ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA CIVIL E INDUSTRIAL

Trabajo Fin de Grado

SISTEMA DE DETECCIÓN Y ALMACENAMIENTO DE VARIABLES VISUALIZADAS POR UN DISPOSITIVO MEDIANTE PANTALLA O DISPLAY A TRAVÉS DE UN SISTEMA DE VISIÓN DE BAJO COSTO.

Titulación: Grado en Ingeniería Electrónica Industrial y Automática

Alumno: Javier Hernández Afonso
Tutor: Santiago Torres Álvarez

Julio, 2014

A mi Abuela,

por recordarme día a día

que el que lucha, gana.

Y que el que quiere, lucha.

ESCUELA SUPERIOR

DE INGENIERÍA

CIVIL E INDUSTRIAL

TITULACIÓN: Grado en Ingeniería Electrónica Industrial y Automática

Memoria

Trabajo Fin de Grado

TÍTULO

SISTEMA DE DETECCIÓN Y ALMACENAMIENTO DE VARIABLES VISUALIZADAS POR UN DISPOSITIVO MEDIANTE PANTALLA O DISPLAY A TRAVÉS DE UN SISTEMA DE VISIÓN DE BAJO COSTO.

AUTOR

Javier Hernández Afonso

TUTOR

Santiago Torres Álvarez

ABSTRACT

The objective of this project is to develop a detection system, recognition and storage of variables displayed on screen by a camera or other viewing device, in order to integrate it into a control algorithm. To this end, there will be necessary to develop an optical character recognition system. Efforts are on the way for the development of efficient and effective OCR system, so that the latency time does not exceed the levels allowed by the control algorithms.

The optical character recognition stages are pre-processing, segmentation, feature extraction and classification. In this project, we will try to describe the different stages of our optical character recognition system, as well as we justify the proposed solution over other alternatives, exposing the results of the testing stage. Moreover, it will describe the methods followed to adapt the system for real-time execution, as well as the integration in the control algorithm.

Índice:

Índice de figuras:

1. ASPECTOS GENERALES

1.1 Objeto

1.1.1 Objetivos técnicos

Desarrollar un sistema de reconocimiento y almacenamiento de variables visualizadas por un dispositivo mediante pantalla o display a través de un sistema de visión de bajo costo.

1.1.2 Objetivos administrativos

Aprobar la asignatura "Trabajo Fin de Grado" para la consecución del título de Graduado en Ingeniería Electrónica Industrial y Automática.

1.2 Descripción

Se pretende desarrollar un sistema de detección, reconocimiento y almacenamiento de variables visualizadas en pantalla mediante una cámara u otro dispositivo de visión, desarrollando para ello un sistema de reconocimiento de caracteres OCR, valorándose:

- Su eficacia: se realizarán pruebas y testeos sobre el sistema y se valorarán los resultados obtenidos.
- Su eficiencia: se tratará de que el sistema de reconocimiento sea lo más rápido posible, de forma que cuando se realice su implantación en un sistema de ejecución dinámica, los tiempos de latencia no superen el periodo de ejecución requerido por el sistema principal, sin que esto perjudique la eficacia del sistema de reconocimiento.

Se expondrá la solución propuesta para la implementación y se justificará frente a otras alternativas.

1.3 Motivación

En la implementación de lazos de control, muchas veces se dispone de dispositivos sensores que monitorizan, y visualizan por pantalla o display, el estado de una o varias variables de interés y, sin embargo, no se puede extraer información de los mismos (bien porque no dispone de puerto de salida para el envío de datos, o bien porque el protocolo de

envío de dichos datos es desconocido o el fabricante no lo facilita). Un ejemplo de ello son los dispositivos que monitorizan el estado de un paciente sometido a una intervención quirúrgica. El objetivo de este proyecto es el diseñar e implementar un sistema de visión de bajo coste que detecte en la pantalla del dispositivo la variación de las variables de interés, a la vez que almacene de forma conveniente dichos datos para ser utilizados por los algoritmos de control. Hay dos claros beneficios del resultado de este proyecto. Por un lado, no se perdería tiempo y esfuerzo en desenmascarar el protocolo de comunicación del dispositivo con otros dispositivos (si esa conexión existiese). Por otro lado, el producto final sería lo suficientemente versátil como para ser empleado en multitud de aplicaciones, y no sólo de control.

1.4 Antecedentes

Actualmente, la Universidad de La laguna se encuantra desarrollando el proyecto ANESPLUS. El proyecto pretende diseñar un controlador multivariable que regule tanto el nivel de hipnosis como el de analgesia en humanos sometidos a intervenciones quirúrgicas en quirófano con anestesia intravenosa.

Desde un punto de vista médico, la variable de interés en la anestesia no es sólo la hipnosis sino también la analgesia y relajación muscular. Por tanto, además de la hipnosis el anestesista debe garantizar un nivel satisfactorio de analgesia para evitar sufrimiento al paciente y también garantizar una correcta relajación muscular. Actualmente existe un gran interés en el ámbito médico en el desarrollo de una herramienta que incorpore estas otras variables para optimizar el proceso anestésico y este proyecto pretende dar respuesta a esta necesidad.

Desde el punto de vista del desarrollo, el objetivo es dotar al anestesista de una herramienta de ayuda en el proceso anestésico que permita monitorizar y controlar las principales variables que intervienen en el proceso. Esto supondría un avance considerable que mejoraría el rendimiento de la operación en términos de tiempos de recuperación del paciente, tiempos de uso de quirófanos y salas de recuperación, ajuste de la dosis infundidas a las necesidades reales de cada paciente y reducción global de costos de cada operación.

El presente proyecto pretende detectar, mediante un sistema de reconocimiento dinámico, las variables de interés en el proceso anestésico de una intervención quirúrgica y

almacenarlas para que así puedan ser utilizadas por los algoritmos de control del sistema principal. El proceso de desenmascaramiento de los protocolos de comunicación de los sistemas de anestesia con otros dispositivos (si existiese) es un proceso largo y engorroso. Éste proyecto pretende dar una propuesta de sistema de reconocimiento universal, de forma que no sea necesario recurrir al desenmascaramiento de dichos protocolos.

2. RECONOCIMIENTO: SISTEMA OCR

2.1 Introducción

2.1.1 Concepto de reconocimiento

El Reconocimiento Óptico de Caracteres (OCR del inglés *Optical Character Recognition*), o generalmente denominado reconocimiento de caracteres, es un proceso dirigido a la digitalización de textos, los cuales identifican automáticamente a partir de una imagen símbolos o caracteres que pertenecen a un determinado alfabeto, para luego almacenarlos en forma de datos. Así podremos interactuar con estos mediante un programa de edición de texto o similar. Las aplicaciones de los sistemas de reconocimiento de caracteres son de lo más variopintas, desde el reconocimiento de texto manuscrito hasta la indexación de bases de datos.

Para el ser humano es relativamente fácil reconocer las principales características de un texto, mientras que para un ordenador se exigen una serie de etapas que, a pesar de que requieren un tiempo de ejecución relativamente corto, es necesario que tengan un alto nivel de precisión. Por ésta razón es necesario identificar las características más adecuadas para conseguir una clasificación eficaz durante el reconocimiento.

El objetivo del reconocimiento de caracteres ópticos (OCR) es identificar patrones dentro de una imagen digital como caracteres alfanuméricos. El proceso de OCR involucra varias etapas, siendo las principales las de segmentación, extracción de patrones y, por ultimo, clasificación.

2.1.2 Modelo general de los sistemas OCR

En el presente apartado se van a describir las distintas etapas en las que se puede dividir un sistema OCR, así como los principales métodos de clasificación empleados en el reconocimiento.

2.1.2.1 Binarización

Es el proceso de transformación de una imagen de intensidades (grises) a una imagen binaria (blanco y negro). Consiste en comparar cada pixel de la imagen con un determinado umbral (valor límite que determina si un pixel será de color blanco o negro). Los valores de la imagen que sean mayores que el umbral toman el valor 255, mientras que el conjunto de pixeles cuyo valor inicial no supere el umbral tomarán el valor 0. Uno de los principales es mediante el estudio del histograma de la imagen donde se muestra el número de pixeles para cada nivel de grises que aparece a la imagen. A la hora de realizar la binarización hemos de escoger un umbral adecuado, de forma que todas las regiones de interés puedan ser identificadas (queden en blanco) tras el proceso. Por ésta razón, en ocasiones es necesario invertir la imagen binarizada.

Figura 2.1: Ejemplo de imagen digital binarizada

La gran mayoría de sistemas parten como base de una imagen binaria (dos colores) por lo tanto es conveniente convertir una imagen de escala de grises, o una de color, en una imagen en blanco y negro, de tal forma que se preserven las propiedades esenciales de la imagen.

2.1.2.2 Segmentación

La segmentación es el proceso por el cual se extrae de la imagen cierta información adyacente para su posterior uso. Mediante este proceso se divide la imagen en las partes u objetos que la forman. El nivel al que se realiza esta subdivisión depende de la aplicación en particular, es decir, la segmentación terminará cuando se hayan detectado todos los objetos de

interés para la aplicación. Los algoritmos de segmentación de imagen generalmente se basan en dos principios fundamentales de los niveles de gris de la imagen: discontinuidad y similitud. Dentro de la primera categoría se intenta dividir la imagen basándonos en los cambios bruscos en el nivel de gris. Las áreas de interés en esta categoría son la detección de puntos, de líneas y de bordes en la imagen. Las áreas dentro de la segunda categoría están basadas en las técnicas de umbrales, crecimiento de regiones, y técnicas de división y fusión. Para el reconocimiento de caracteres, nos basaremos en la segmentación orientada a regiones.

En general, la segmentación automática es una de las tareas más complicadas dentro del procesado de imagen. La segmentación va a dar lugar en última instancia al éxito o fallo del proceso de análisis. En la mayor parte de los casos, una buena segmentación dará lugar a una solución correcta, por lo que, se debe poner todo el esfuerzo posible en la etapa de segmentación.

2.1.2.3 Extracción de características

Una vez realizada la segmentación, se tiene una imagen normalizada en la que se encuentra la información susceptible de ser "reconocida". El siguiente paso sería seleccionar aquellas características con las que consigamos un mayor desacoplamiento entre clases, y así realizar una clasificación satisfactoria.

Desde el punto de vista del reconocimiento de formas, la matriz bidimensional se ve como un vector de tantas dimensiones como componentes tiene la matriz. La dimensión de estos vectores (el número de componentes) es normalmente elevado, lo que supone un gran coste computacional a la hora de procesar el mismo. Y no sólo eso, y más importante aún, es que está comprobado que al intentar clasificar ("reconocer") vectores de este tamaño aparece un efecto, llamado maldición de la dimensionalidad, que provoca que los resultados, independientemente del método de clasificación utilizado, no sean satisfactorios. Por ello se han desarrollado multitud de técnicas, denominadas "técnicas de selección y extracción de características", mediante las cuales es posible obtener una representación del objeto a reconocer más eficiente. Eficiencia, en este caso, significa que con una representación más compacta se consigue un poder discriminativo igual o superior al que se tenía con la representación original. Esto no es sólo importante por el ahorro de espacio en el almacenamiento de las muestras, sino que durante el proceso de reconocimiento reduce los costes computacionales, debido a la reducción en el volumen de información procesado.

La extracción de las características es una de las fases más difíciles en los sistemas de reconocimiento de caracteres, puesto que la elección de las características más adecuadas para la clasificación es una tarea altamente compleja y depende de la aplicación final del sistema. Para que una característica pueda considerarse como buena para su extracción:

- Sus valores deberán ser correctamente diferenciables entre clases.
- Deben ser independientes, es decir, las características deben estar no correladas unas de otras.

El número de características deberá ser lo más pequeño posible para asegurar la rapidez y facilidad de clasificación, siempre y cuando no perjudique la eficiencia del sistema. Las características más comúnmente utilizadas pueden ser clasificadas en dos categorías:

- Geométricas: momentos, histogramas o direcciones (códigos de cadena).
- Estructurales: descriptores de Fourier, métodos correlativos o propiedades topológicas del esqueleto o contorno de regiones.

2.1.2.4 Clasificación

Proceso fundamentado en el reconocimiento de patrones cuyo objetivo es la asociación de un conjunto de objetos a sus correspondientes categorías o clases. El problema que en esta etapa se plantea consiste en desarrollar algún método que sea capaz distinguir la clase a la que pertenece un objeto entre un conjunto limitado de clases posibles. Este planteamiento general del reconocimiento de formas, se traduce, en el caso de la aplicación de OCR, en asignar un carácter hipótesis a una imagen de entrada. Históricamente los dos enfoques en el reconocimiento de patrones han sido el estadístico (teoría de la decisión) y el sintáctico (o estructural), aunque en la actualidad la utilización de redes neuronales ha cobrado gran importancia en los sistemas de reconocimiento de caracteres. Los clasificadores más comúnmente utilizados en los sistemas OCR son:

- Clasificador KNN: o clasificador de los "k vecinos más próximos". El método funciona como se explica: dado un conjunto de objetos prototipo de los que ya se conoce su clase (es decir, dado un conjunto de muestras de entrenamiento) y dado un nuevo objeto cuya clase no conocemos (imagen de un carácter a reconocer) se busca entre el conjunto de prototipos los "k" más parecidos al nuevo objeto. A este se le asigna la clase más numerosa entre los "k" objetos prototipo seleccionados.

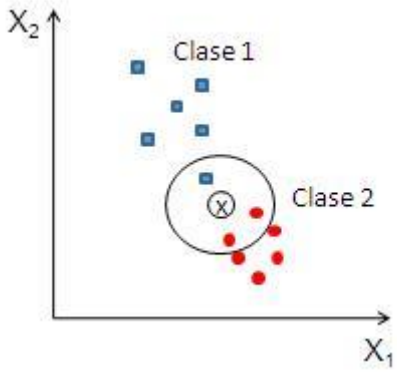

Figura 2.2: Ejemplo de clasificador knn para k=3. Fuente: Wikipedia

La elección del número adecuado de vecinos k es una tarea que no tiene una procedimiento fijo, y en ocasiones su búsqueda se vuelve un proceso iterativo.

- Árboles de decisión: como su nombre lo indica el modelo sigue una estructura tipo árbol, donde los nodos no terminales son condiciones sobre las variables de entrada y los nodos terminales (o nodos hoja) son las clases asociantes.

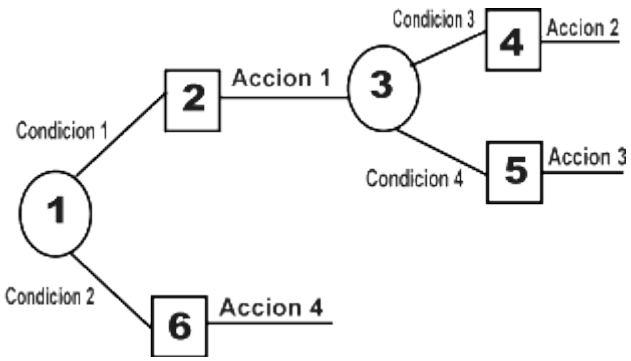

Figura 2.3: Estructura de un árbol de decisión. Fuente:wikispaces.com

Se trata de sistemas de fácil diseño y sencillamente interpretables por el ser humano. Son especialmente útiles en problemas de entradas discretas, y su velocidad de clasificación es alta si está correctamente balanceado.

- Redes neuronales artificiales: Las redes neuronales artificiales son un paradigma de aprendizaje ampliamente tratado en la literatura de la inteligencia artificial. Las redes neuronales artificiales (RNA) intentan imitar la naturaleza de la neuronas del cerebro humano.

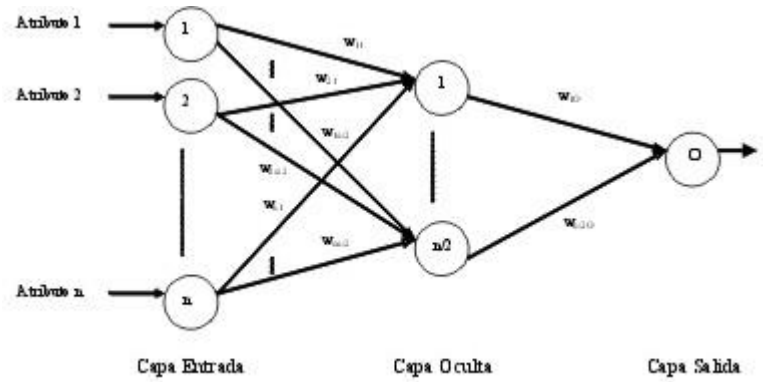

Figura 2.4:Estructura de un perceptrón. Fuente: Electrónica.com.mx

Cada una de las neuronas de un perceptrón multicapa recibe un vector de datos de entrada x, y tiene un vector de pesos sinápticos w y una función de activación f(x). Dado sus entradas, la salida y de una neurona es:

$$y = f(\vec{w}, \vec{x}) \qquad (2.1)$$

El poder clasificatorio de éste método es superior a los mostrados anteriormente, pues matemáticamente se ha demostrado que no exigen ninguna distribución determinada de los datos. Sin embargo, debido a la cantidad de parámetros que se pueden llegar a configurar, estos algoritmos son muy difíciles de afinar para un problema determinado, y son también muy difíciles de interpretar.

2.2 Diseño del sistema OCR

2.2.1 Introducción

En los siguientes apartados se describen las diferentes etapas de procesamiento de las que se componen el sistema OCR desarrollado, exponiendo las teorías que justifican la propuesta, así como describiendo la implementación realizada en el software matemático Matlab.

2.2.2 Pre-procesamiento

Antes de poder analizar la imagen y extraer sus características, es necesario realizar una serie de procesos de adecuación de la imagen. En el sistema desarrollado, dichos procesos son el filtrado de la imagen, su umbralización y su erosionado.

2.2.2.1 Filtrado

En ocasiones, nuestra imagen presentará ciertas cantidades de ruido que dificultarán el procesamiento eficiente de la imagen. Para reducir éste ruido y suavizar la imagen, suele ser común la utilización de filtros de suavizado de imagen. Para nuestro sistema OCR, emplearemos el filtro de la mediana. Se hace pasar una rejilla nxn por la imagen, sustituyendo el valor del conjunto de pixeles contenidos en la rejilla por la mediana del mismo conjunto. En conjuntos de pixeles de un solo valor de intensidad, como los números, dicho valor suele ser el más repetido.

Figura 2.5: Funcionamiento del filtro de mediana. Fuentes imagenes-uisrael.blogspot.com

Éste filtro de la ventaja de que el valor final del pixel es un valor real presente en la imagen y no un promedio, reduciendo el efecto borroso que producen otros tipos de filtros (como los filtros promedio). Además el filtro de la mediana es menos sensible a valores

extremos, y tiene la cualidad de preservar los bordes de las regiones de imágenes (*edge-preserving*), por lo que no perjudica el posterior análisis de los objetos.

- *Implementación en Matlab:*

La función *medfilt2* aplica el filtro de mediana de rejilla nxn (por defecto 3x3) a una imagen de intensidades.

```
>Im = medfilt2(imRuido);
```

Figura 2.6: Ejemplo de aplicación del filtro de mediana

2.2.2.2 Binarización

La imagen será binarizada mediante la técnica de detección de umbral. El uso de umbrales en el análisis de imágenes constituye una de las principales técnicas en los sistemas de visión artificial para la detección de objetos.

Suponiendo que se quiere aislar un objeto del entrono mediante umbralización, será necesario seleccionar un umbral adecuado que separe los dos tonos de intensidad. Podemos considerar la fijación del umbral como una operación que implica pruebas con respecto a una función T de la forma,

$$T = T[x, y, p(x, y), f(x, y)] \qquad (2.2)$$

siendo *f(x,y)* la intensidad en el punto *(x,y)* y *p(x,y)* la intensidad media de una región cerrada. Así, se creará una imagen binaria *g(x,y)* definiendo,

$$g(x,y) = \begin{cases} 0 \; si \; f(x,y) > T \\ 1 \; si \; f(x,y) \leq T \end{cases} \qquad (2.3)$$

Dependiendo de la imagen, la umbralización asignará un 0 o 1 al entorno y un 1 o 0 al objeto de interés. Normalmente, el procesamiento que se realiza tras el umbralizado suele recaer en el objeto, por lo que nos interesa que dicha región se le asigne un 1 y al entorno un 0. En ocasiones suele ser necesario invertir la imagen para cumplir éste requerimiento.

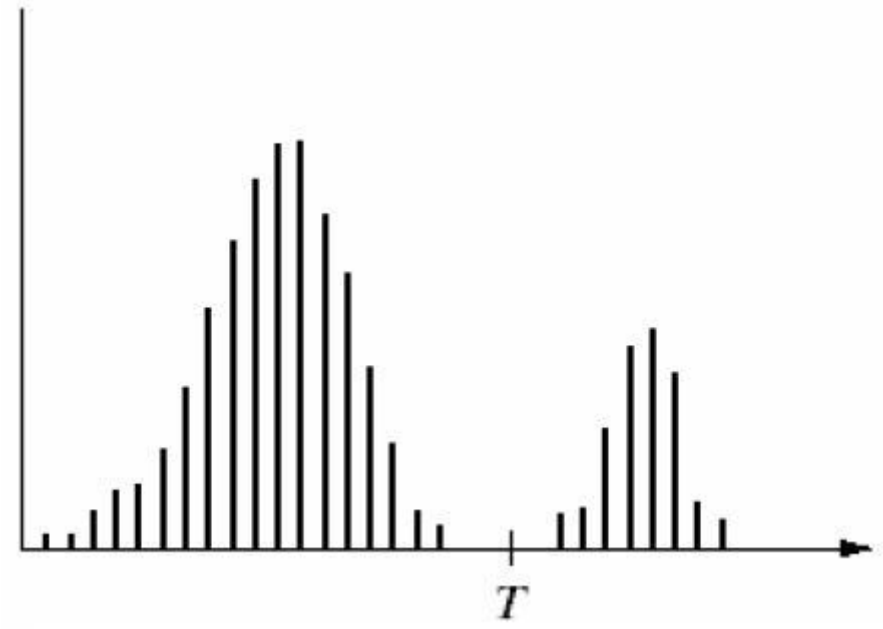

Figura 2.7: Umbralización del histograma. Fuente:digitimagen.blogspot.com

Para obtener dicho umbral, uno de los métodos más utilizado es el conocido método de Otsu (Nobuyuki Otsu, 1979), que utiliza métodos estadísticos para la resolución de dicho problema. Consideremos dos a *G1* y *G2* como dos grupos de pixeles divididos por un umbral *T*. La varianza dentro de los grupos será,

$$G_w^2 = n_1(T) \cdot G_1^2(T) + n_2(T) \cdot G_2^2(T) \qquad (2.4)$$

y la varianza entre grupos,

$$G_b^2 = n_1(\mu_1 - \mu)^2 + n_2(\mu_2 - \mu)^2 \qquad (2.5)$$

para:

$$\mu = n_1\mu_1 + n_2\mu_2 \qquad (2.6)$$

siendo:

μ_i: media del conjunto G_i.

n_i: fracción total de pixeles del conjunto G_i.

Obtenidas ambas varianzas, obtendríamos el cociente siguiente:

$$J = \frac{G_b^2}{G_w^2} \qquad (2.7)$$

Por tanto, el objetivo sería encontrar un umbral *T* que maximice el cociente de varianzas *J*, es decir, maximizar la varianza entre clases y minimizar la varianza "intra-clases".

- *Implementación en Matlab:*

La función *graythresh* de Matlab calcula el umbral *T* mediante la aplicación del método de Otsu, introduciendo una imagen de intensidades *img* como parámetro de entrada. Además, la función *im2bw* binariza la imagen con un umbral *T* previamente hallado, obteniendo una imagen binaria *bw*. Así, el código para la binarización de una imagen RGB sería:

```
>img = rgb2gray(im)
>umb=graythresh(img);
>bw(:,:,i)=im2bw(img, umb);
```

Empleamos la rejilla por defecto 3x3. Una rejilla mayor podría provocar que dos regiones no conectadas terminasen conectándose por efecto del filtrado. Si quisiésemos invertir la imagen, bastaría con el uso del comando *not* sobre una imagen binaria.

Figura 2.8: Proceso de binarización de una imagen

2.2.2.3 Erosionado

Por culpa de una mala umbralización, o por efecto del proceso de filtrado, en ocasiones pueden quedar pixeles no conectados a las regiones en las imágenes binarizadas. Para que éstos pixeles no provoquen una posterior extracción de características defectuosas, es conveniente aplicar un proceso de erosionado sobre la imagen binarizada.

La erosión de la imagen binaria A por un elemento estructurante B está definida por:

$$A \ominus B = \{Z \in E | \, B_Z \subseteq A\} \qquad (2.8)$$

siendo B_z la traslación de B por el vector Z:

$$B_Z = \{b + z | b \in B\}, \forall z \in E \qquad (2.9)$$

A partir de las expresiones anteriores, se puede entender la erosión del conjunto de pixeles (u objeto en imágenes binarias) A por el elemento estructurante B como el lugar geométrico de los puntos alcanzados por el centro del elemento B cuando éste se mueve por A.

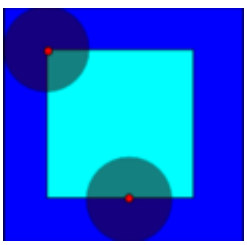

Figura 2.9: Erosionado con elemento estructurante tipo disco. Fuente: Wikipedia.com

- *Implementación en Matlab*

La función *strel* permite crear elementos estructurantes para su uso en operaciones morfólogicas sobre imágenes. Nosotros utilizaremos un elemento cuadrado 2x2. Un elemento de mayor tamaño podría llegar a "cortar" regiones, lo que provocaría posteriores errores en el reconocimiento. La función *imerode* realiza la erosión con el uso de un elemento estructurante previamente creado. El código implementado es el siguiente.

```
>se = strel('square',2);
>erodedBw = imerode(Bw,se);
```

(A) *(B)*

Figura 2.10: Resultado de la erosión (B) sobre la imagen binaria(A)

2.2.3 Segmentación y Normalización

2.2.3.1 Segmentación

Debido a que la cámara permanecerá fija durante la grabación de las variables de interés, la segmentación de dicha variable la realizaremos mediante un calibrado manual sobre la imagen RGB tomada por la cámara.

- *Implementación en Matlab*

La función *cali*pecghrmite seleccionar 4 puntos de una imagen de entrada para poder extraer sus coordenadas. Por otra parte, la función *cort* permite recortar una imagen pasando como parámetros de entrada 4 coordenadas previamente calculadas. Este calibrado deberá realizarse antes de comenzar la ejecución. El código implementado es el que sigue:

```
>[coor] = cali(im);
>im = cort(im, coor);
```

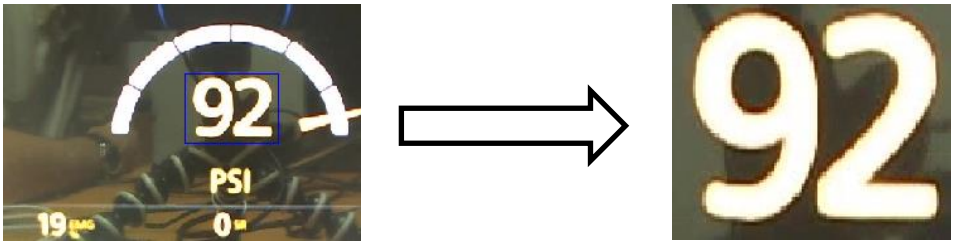

Figura 2.11: Operación de corte de imagen

Nota: El código de las funciones cali y cort puede verse en el anexo I: código fuente, en la sección "sistema OCR principal"

2.2.3.2 Extracción de regiones

Una vez la imagen cortada haya sido pre-procesada, el primer paso será extraer las regiones que contenga la misma. En nuestro sistema, estas regiones serán cifras (y puntos decimales en algunos casos).

El objetivo consiste en etiquetar todas las componentes conexas de forma que el resultado final sea una región. Se entiende por componente conexa como todos aquellos pixeles de valor 1 que están conectados entre sí por un camino o conjunto de pixeles de valor 1. Para el etiquetado de componentes existen multitud de algoritmos, todos ellos basados en el tipo de conectividad (conectividad-4 o conectividad-8).

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 2 \\ 1 & 0 & 2 \\ A & 2 & 2 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

(A) *(B)* *(C)*

Figura 2.11: Ejemplo de etiquetado de la imagen (A)

En el ejemplo de la figura, A representaría una imagen, B la imagen etiquetada mediante un algoritmo basado en conectividad-4 (2 objetos), y C la imagen etiquetada mediante un algoritmo basado en conectividad-8 (un objeto).

- *Implementación en Matlab*

La función *regionprops* permite extraer distintas propiedades de cada una de las regiones que se detecten por el algoritmo interno de la función, tales como su centroide, área, esquinas,… Una de estas propiedades es el "*boundingbox*" que es la caja rectangular de perímetro mínimo que contiene la región etiquetada. La función devuelve las coordenadas de dicha caja, por tanto, podríamos "recortar" cada una de las regiones para extraer las características de cada una de forma individual. El código implementado es el que sigue:

```
>props=regionprops(bw, 'boundingbox');
>A=size(props)
>A=A(1);
>
>for i=1:A
>    coor=props(1).BoundingBox;
>    Num=cort(bw, coor);
>
.
. (Código de post-procesado y extracción)
.
>end
```

Ejecutamos un bucle *for* en el que las imágenes cortadas por el "*boundingbox*" son analizadas una a una para su extracción de características (código de extracción). A será igual al número de regiones conectadas indexadas por la función *regionprops*.

Figura 2.12: Número cortado con el uso del "boundingbox"

2.2.3.3 Asignación de pesos

Durante el reconocimiento, tenemos que contemplar el número de cifras que compone el número que da valor a la variable. Si queremos reconocer el número 92, nos interesa que el sistema nos devuelva 92, no un 9 y un 2.

- *Implementación en Matlab*

La función *regionprops* indexa las regiones en el sentido de lectura de la misma, siendo éste es de derecha izquierda. Bastaría con contar el número de cifras (regiones indexadas por Matlab) y asignar el peso máximo a la primera región. Éste sería:

$$P_{MAX} = 10^{N-1} \qquad (2.10)$$

siendo N el número de cifras reconocidas. Sin embargo, esta expresión no nos serviría en el caso de que existiese decimal. En ese caso, sería necesario encontrar la posición relativa del decimal en la imagen respecto a las demás regiones. En ese caso, la expresión sería:

$$P_{MAX} = 10^{D-2} \qquad (2.11)$$

Siendo D la posición del decimal respecto a las demás regiones, contando de izquierda a derecha. La dificultad radicaría en saber cuándo hay decimal. Para ello hemos implementado el siguiente algoritmo en Matlab.

```
> [L, N]=bwlabel(bw);
> R=regionprops(bw, 'Area');
> A=0;
>
>     for i=1:N
>        A(i)=R(i).Area;
>     end
>
> M=max(A);
> [m, in]=min(A);
>
>    if N>2&&M>(10*m)
>        D=1;
>    else
>        D=0;
>    end
>
>    if D==1
>
>        P=10^(in-2);
>    else
>        P=10^(N-1);
>    end
```

Una de las funciones del comando *bwlabel* es obtener el número de regiones conectadas que hay en una imagen, sin necesidad de utilizar la función *regionprops* en casos no necesarios. Mediante el uso de *regionprops* podemos obtener las áreas de cada una de las regiones. Así, establecemos que habrá decimal si la relación entre el área máxima y mínima es 10:1 (en ese caso, *D=1*). Además, se ha considerado que si no hay más de dos regiones conectadas, no puede haber decimal. Extrayendo el índice *in* del área mínima podemos obtener la posición relativa del posible decimal. Los pesos se calculan con una de las ecuaciones expuestas anteriormente, en función de si se obtiene decimal o no.

2.2.3.4 Normalización

En la extracción de características en sistemas OCR, la fuente cobra un papel fundamental, pues parámetros como la anchura, espesor o elongación del número pueden hacer variar las características extraídas. Por tanto, a veces es necesario realizar un proceso de normalización sobre las regiones segmentadas.

Un buen sistema de normalización es forzar a la región una resolución, fija para cada una de las entradas impedientemente del tamaño de la fuente. Posteriormente, se realiza una esqueletización de la región que consiste en hacer adelgazar al objeto hasta que se conforme de una única línea de píxeles conectados.

Pajares y de la Cruz (2001) define el adelgazamiento de una imagen como una técnica basada en la transformación morfológica conocida como *añadir o eliminar*. Ésta operación puede describirse como la acción de un elemento estructural compuesto B=(B_1,B_2) sobre un conjunto de píxeles X.

$$X * B = \{x: B_1 \subset X \; y \; B_2 \subset X^c\} \qquad (2.13)$$

Esto implica que para cualquier punto x que pertenece al conjunto resultante, se cumplen dos condiciones simultáneamente. En primer lugar, la parte B1 que tiene su punto representativo en x estaría contenida en X, y por otra parte, la parte B2 del elemento estructural estaría contenida en Xc (entendiéndose Xc como el conjunto de pixeles complementario a X). La transformación *añadir o eliminar* puede expresarse utilizando erosiones y dilataciones como sigue:

$$X * B = (X \oplus B_1) - (X \oplus B_2) \qquad (2.14)$$

Como hemos dicho, una aplicación de la transformada añadir o eliminar es la de adelgazamiento de regiones, y sigue la siguiente expresión:

$$X \rhd B = X - (X * B) \qquad (2.15)$$

Así, cuando se produce un adelgazamiento, una parte del borde del objeto es sustraída de él por la operación diferencia de conjuntos.

- *Implementación en Matlab:*

Siendo *bw* una imagen de una región segmentada, la función *imresize* permite redefinir la resolución de la imagen. Por otro lado, la función *bwmorph* permite realizar distintas operaciones morfológicas sobre imágenes binarias, entre ellas el adelgazamiento (*thinning*). A continuación se presenta el código implementado

```
> bw=imresize(bw, [33, 33]);
>
> se = strel('square', 1);
> bw = imopen(bw,se);
>
> bwOUT = bwmorph(bw,'thin', Inf);
```

Con *Inf* indicamos que repita el proceso hasta que la imagen quede esqueletizada. Observar que también realizamos un proceso de apertura sobre la imagen redimensionada (erosión + dilatación), pues en ocasiones, al redimensionar, pueden producirse desconexiones ("cortes") sobre una región conectada. Hacemos éste proceso para cada una de las cifras de la variable.

(A) *(B)* *(C)*

Figura 2.13: Proceso de redimensionamiento (B) y esqueletización(C) de una región (A)

Nota: Si resultase confusa la comprensión del código, éste puede verse en su contexto en el Anexo I: código fuente.

2.2.4 Extracción de características

En la etapa de extracción de características de nuestro sistema, hemos decidido calcular el número de Euler y los nueve momentos centrales normalizados de orden 0, 1, 2 y 3 para cada región. A continuación se exponen los fundamentos teóricos y el procedimiento seguido para cada proceso.

2.2.4.1 Número de Euler

En imágenes digitales, el número de Euler se conoce como el número de regiones conectadas de una imagen menos el número total de agujeros. Así, podríamos pre-clasificar los números en función de su número de Euler: el número de Euler de 0, 4, 6 y 9 sería 0; el de 1, 2, 3, 5, 7 el 1; y el del 8, -1.

- *Implementación en Matlab:*

La función *regionprops* permite obtener el número de Euler de una región conectada. El código a implementar sería el siguiente:

```
> props=regionprops(bw, 'EulerNumber');
> E=props.EulerNumber;
```

2.2.4.2 Momentos Centrales

Siendo *f(x,y)* la intensidad del punto *(x,y)* en una región, el momento de orden $p+q$ de una imagen de intensidades (de una región en caso de imágenes binarias) digital sería,

$$m_{pq} = \sum_{x} \sum_{t} x^p y^q f(x,y) \qquad (2.16)$$

donde el sumatorio se realiza sobre todas las coordenadas espaciales de puntos de la región. El momento central de orden $p+q$ viene dado por,

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y) \qquad (2.17)$$

donde,

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

Los momentos centrales normalizados de orden $p+q$ se definen como:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\Upsilon}}, \quad donde \quad \Upsilon = \frac{p+q}{2} + 1 \quad para\ (p+q) \geq 2$$

2.2.4.3 Momentos invariantes de Hu

Ming-Kuei HU postuló en el año 1962 un conjunto de 7 momentos invariantes a la rotación, traslación y cambios de escala. Son los conocidos momentos invariantes a continuación y se exponen a continuación.

$$\Phi_1 = \eta_{20} + \eta_{02}$$

$$\Phi_2 = (\eta_{20} - \eta_{02})^2 + 4{\eta_{11}}^2$$

$$\Phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

$$\Phi_4 = (\eta_{30} - \eta_{12})^2 + (\eta_{21} - \eta_{03})^2$$

$$\Phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2] + \qquad (2.17)$$

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

$$\Phi_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} - \eta_{03})^2] +$$

$$4\eta_{11}(\eta_{30}+\eta_{12})(\eta_{21}+\eta_{03})$$

$$\Phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2] +$$

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

- Implementación en Matlab:

Se estudió tomar como características de las regiones los momentos centrales, los momentos centrales normalizados (de orden 3 e inferior) y los momentos invariantes de Hu. Más adelante, se descartaron los momentos invariantes de Hu, pues el proceso de normalización expuesto anteriormente hace que pierda sentido calcular un momento invariante a la rotación, traslación y cambio de escala. Se estudiaron los resultados con los momentos centrales y normalizados, y se comprobó que fueron mejores con los normalizados. A continuación se muestra un pequeño fragmento del cálculo de los momentos centrales normalizados normalizados. Para ver el código completo, diríjase al Anexo I: Código Fuente, sección "Sistema OCR principal".

A) Cálculo de momento m_{00}

```
> for x=1:m
>    for y=1:n
>        m00=m00+(x^0)*(y^0)*R(x,y);
>
>     end
> end
```

B) Cálculo de momento central μ_{00}

```
> xm=m10/m00;
> ym=m01/m00;
> for x=1:m
>     for y=1:n
>         u00=u00+((x-xm)^0)*((y-ym)^0)*R(x,y);
>     end
> end
```

C) Cálculo del momento central normalizado η_{00}

```
> y15=1.5;
> n10=u10/(u00^y15);
```

2.2.5 Clasificación

Para la clasificación, decidimos implementar un conjunto de clasificadores Knn o de "k vecinos más próximos", en combinación con un árbol de decisión simple. Utilizamos el árbol de desición para pre-clasificar el dígito en función de su número de Euler, para después llevarlo al clasificador Knn-1 correspondiente: clasificador-1 para dígitos de número de Euler 1, y el clasificador Knn-0 para número de Euler 0. Dentro de los clasificadores Knn, se clasificarán los números en función de los momentos centrales normalizados. El único dígito con número de Euler -1 es el 8, por lo que no será necesario enviarlo a ningún clasificador Knn que estudie los momentos del mismo.

Como habíamos comentado en el Apartado "Modelo general de un sistema OCR", los árboles de decisión siguen una estructura tipo árbol, donde los nodos no terminales son condiciones sobre las variables de entrada y los nodos terminales (o nodos hoja) son las clases asociantes. Entre los nodos podemos encontrar cuadros de acción, que realizan las operaciones necesarias para el tránsito entre nodos. En nuestro caso, estos cuadros de acción serán el cálculo del número de Euler *E*, y los clasificadores *Knn-1* y *Knn-2*, en función del parámetro anterior. En el siguiente esquema puede observarse la estructura del árbol implementado:

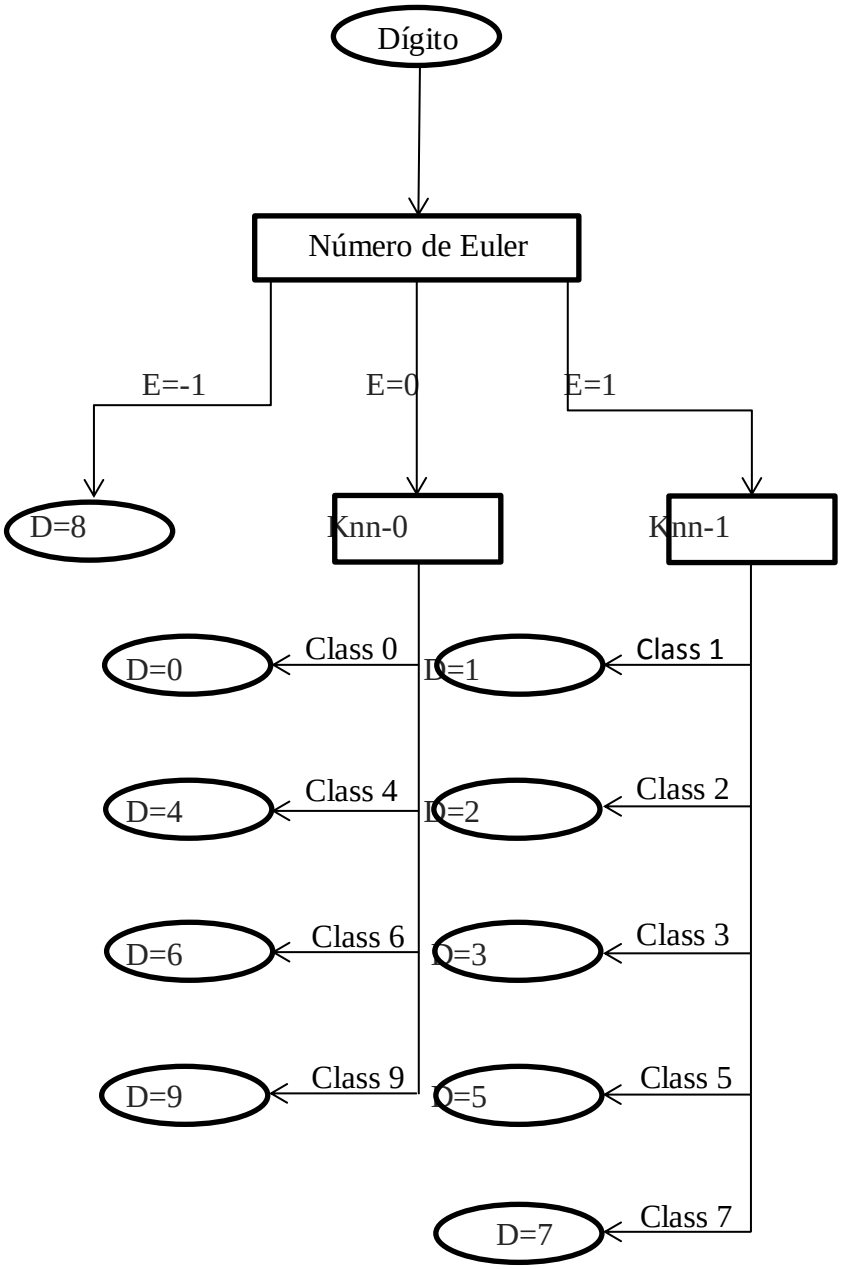

Figura 2.14: árbol de decisión del sistema OCR

Comentábamos que el clasificador Knn funcionaba buscando, entre el conjunto de protoripos, los "k" vecinos más próximos de un objeto de entrada de clase desconocida, teniendo un conjunto de objetos prototipo de los que sí se conoce. Un potencial problema de éste clasificador es que asigna a todos los objetos entrantes una clase, pues siempre habrá un vecino más próximo. Esto puede provocar asignaciones de una clase de número a un objeto que no es un número. Sin embargo, se considera que con un correcto calibrado del dispositivo

de video, y con la ayuda de los procesos de filtrado y erosionado, nunca se presentarán en pantalla objetos que no sean dígitos. Para la búsqueda de los K vecinos más próximo, los clasificadores Knn se basan en el cálculo de la distancia euclidiana ente el objeto entrante y los objetos de muestra, y selecciona los vecinos en función de la mínima distancia calculada. La distancia euclidiana entre un objeto entrante x_i y un objeto de muestra x_j es:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{p} (x_{ir} - x_{jr})^2} \qquad (2.\ 16)$$

Siendo:

X_{ir}: característica r del objeto entrante x_i.

X_{jr}: característica r del objeto entrante x_j.

p: número total de características del objeto (en nuestro caso los 9 momentos centrales normalizados de orden menor e igual a 3).

El proceso de aprendizaje de este clasificador consiste en almacenar en generar unos vectores de entrenamiento a partir de una base de datos de características de objetos de muestra. Dicho proceso lo realizaremos manualmente. Así, ante un nuevo objeto a clasificar, se calcula la distancia euclideana con respecto a los n ejemplos almacenados en los vectores de entrenamiento, y se considerarán los k más cercanos. El nuevo objeto será clasificado en la clase mayoritaria de los k ejemplos más cercanos. El caso más sencillo es para K=1. El estudio del clasificador para distintos valores de K puede verse en el capítulo "Resultados y Valoraciones".

- *Implementación en Matlab*

El árbol de decisión lo realizaremos mediante sentencias if/else. Por otro lado. La función knnclassify permite clasificar un objeto entrante o "sample" mediante el método de los K-vecinos próximos, pasándole cómoparámetros los ventores de entrenamiento ("*trainning*") y de grupo ("*group*"). Éste último vector contiene la clase asociada a las muestras de entrenamiento (momentos) del vector "trainning". Él código implementado es el siguiente:

```
> load traininv
>
> if E==-1
>          Num=8;
>
> elseif E==0
>          Num=knnclassify(double(Mom), double(training0), group0);
>
> else
>          Num=knnclassify(double(Mom), double(training1), group1);
>
> end
```

Traininv es un fichero de datos que contiene los vectores de entrenamiento y grupo que se introducen al clasificador. El procedimiento seguido para la generación de éstos vectores es el que se describe a continuación.

- *Implementación en Matlab (2): Entrenamiento de la máquina de vectores*

En un esfuerzo por crear un sistema de reconocimiento multi-fuente, creamos una imagen con 6 fuentes distintas para cada número. Al cargar la imagen en el programa, procesaremos cada imagen de la misma forma que para las imágenes de entrada. Tras esto, Matlab mostrará al programador un mensaje en pantalla:

```
>N=input('¿Qué número es este? ');
```

A la vez que muestra el número procesado. Introduciendo el valor del objeto mostrado, el siguiente paso será calcular los momentos centrales normalizados del mismo. Destinamos una matriz de almacenamiento para cada número, de forma que si introducimos en la ventana de comandos de Matlab un 5, las características extraídas del objeto serán almacenadas en el vector de momentos Mom(5):

```
>MomC=sevenHU(bwC);

...

> if N==5
>       training5(Ind(6), :)=MomC;
>       Ind(6)=Ind(6)+1;
> end
```

Ind, marca la posición de la matriz en la que se almacenarán el nuevo vector de momentos y se incrementa iteración a iteración. Al finalizar de procesar la imagen, se crean los vectores finales de grupo y entrenamiento de los clasificadores *knn-1* y *knn-0*:

A) Vectores de entrenamiento

```
> trainingUNO=[training1; training2; training3; training5;...
>             training7];
> trainingCERO=[training0; training4; training6;...
>              training9];
```

B) Vectores de grupo

```
> groupCERO=[0*ones(size(training0,1),1);...
>            4*ones(size(training4,1),1);...
>            6*ones(size(training6,1),1);...
>            9*ones(size(training9,1),1)];
>
>
> groupUNO=[1*ones(size(training1,1),1);...
>           2*ones(size(training2,1),1);...
>           3*ones(size(training3,1),1);...
>           5*ones(size(training5,1),1);...
>           7*ones(size(training7,1),1)];
```

2.2.6 Modos de reconocimiento

Tras probar el software prototipo sobre varios paneles de sistemas anestésicos se comprobó que, en algunos casos, las cifras están demasiado próximas, llegando incluso a solaparse, imposibilitando el reconocimiento múltiple de cifras. Por ese motivo, se decidió dar la posibilidad de seleccionar un modo de reconocimiento: múltiple o individual. En el reconocimiento individual, se seleccionan (con la función *cali*) todas las zonas que contengan una sola cifra. Para ello, el programador recibirá, en la ventana de comandos, una pregunta relacionada con el número de cifras (y de decimales) a reconocer. Se deberá especificar en la misma ventana el número exacto de cifras, para después realizar el calibrado individual de las zonas de reconocimiento de las mismas. Una vez obtenidos los parámetros propios de la configuración del sistema, el programa funcionará de forma análoga en ambos modos de reconocimiento (el programa ha sido configurado de tal forma que realice un procesamiento

por cada grupo de coordenadas recibidas). Recordamos que el programa completo puede verse en el Anexo I.

2.3 Sistema de detección para 7 segmentos

Antes de desarrollar el sistema expuesto en el apartado 2.2, se realizó un pequeño sistema de detección de dígitos en 7 segmentos. Dado que la mayoría de monitores de variables del proceso anestésico escanean sus valores en fuentes tipográficas modernas, este sistema termino por descartarse. Sin embargo, consideramos interesante describir el sistema desarrollado, pues consideramos que puede ser de utilidad en otras aplicaciones.

En los siguientes apartados se describen las diferentes etapas de procesamiento de las que se componen el sistema OCR desarrollado para la detección de dígitos en 7 segmentos, así como la implementación realizada en el software matemático Matlab.

2.3.1 Pre-procesamiento

2.3.1.1 Binarización

Utilizaremos la técnica de la detección de umbral, basada en el método de Otsu presentado en la sección 2.2.1.2

2.3.1.2 Dilatación

Partimos de la base de que un número en 7 segmentos está compuesto por 7 regiones conectadas independientes. Para que el reconocimiento sea posible, debemos intentar formar una única región conectada para cada cifra. Para esto, utilizaremos la operación morfológica de la dilatación.

En la sección 2.2.2.3 definíamos la erosión como el lugar geométrico de los puntos alcanzados por el centro de un elemento estructurante en un conjunto conectado de píxeles, provocando el adelgazamiento de las componentes afectadas. La dilatación es el fenómeno contrario, es decir, la expansión de un conjunto de pixeles conectados, provocada por la transformación producida por un elemento estructurante sobre los pixeles vecinos al objeto. En ocasiones, este fenómeno se denomina rellenado o crecimiento.

La dilatación de un conjunto de pixeles A por un elemento estructurante B está definida por:

$$A \oplus B = \{Z \in E | \, (B^s)_Z \cap A \neq \emptyset\} \qquad (2.17)$$

Pretendemos dilatar cada una de las regiones conformadas por los 7 segmentos del display, reiteradamente, hasta que el conjunto de segmentos forme una sola región conectada.

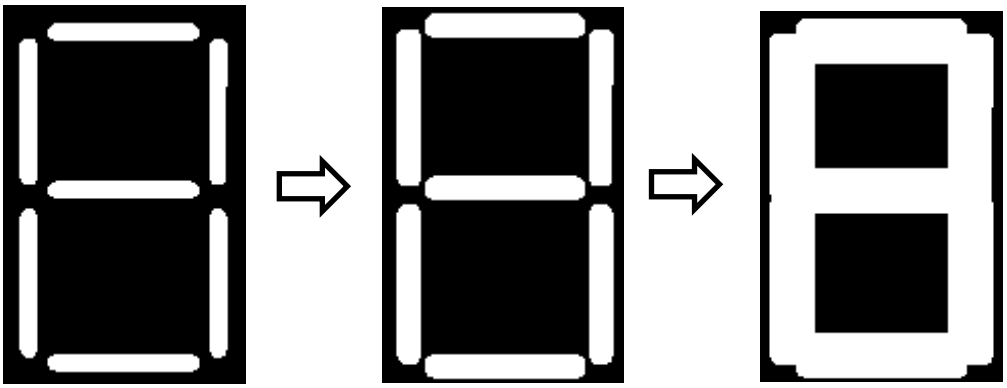

Figura 2.15: Dilatación de componentes conexas en números de 7 segmentos

- *Implementación en Matlab*

Para la formación de una única región conectada, desarrollamos el siguiente bucle:

```
> while true
>
> s=s+1;
> se=strel('square', s);
> BWx=imdilate(BW, se);
> [L,NUM] = BWLABEL(BW);
>
> if NUM==1
>      BW=BWx;
>      break
> else
>      BW=BWx;
> end
> pause
> end
```

Realizamos dilataciones sucesivas sobre una imagen binaria entrante de 7 segmentos, mediante el comando *imdilate* (combinado con la formación del elemento estructurante mediante *strel*), usando una sentencia *while* para el bucle. Utilizaremos un elemento estructurane cuadrado para preservar las líneas rectas de la imagen, aumentando su lado en una unidad cada iteración. Después de cada dilatación, obtenemos el número de regiones

conectadas que contiene la imagen, de forma que en el momento de detectar 1, interrumpimos el bucle y tomamos como imagen de salida la correspondiente a la última dilatación (*BWx*).

2.3.1.3 Erosión

De acuerdo a la descripción realizada en 2.2.1.3 y a las ecuaciones planteadas en dicho apartado, nuestro objetivo ahora es hacer adelgazar a la región para facilitar la posterior extracción de características.

- *Implementación en Matlab*

Nuevamente, desarrollamos un bucle para realizar la operación, salvo que en ésta ocasión, erosionamos con *imerode* y rompemos el bucle en el momento en el que dejemos de tener una región conectada, quedándonos con la imagen anterior a la última erosión:

```
> while true
> s=s+1;
> se=strel('square', s );
>
> BWx=imerode(BW, se);
>
> [L,NUM] = BWLABEL(BWx);
>
> if NUM==1
>     BW=BWx;
> else
>     break;
> end
>
> end
```

2.3.2 Extracción de características

2.3.2.1 Extracción de líneas mediante la transformada de Hough

La característica extraída de las regiones, una vez procesadas, será la forma de los segmentos que la componen (concretamente, su longitud y orientación), una vez haya sido pre-procesada (unión de segmentos). Para ello, utilizaremos la transformada de Hough.

Como se comenta en Pajares y de la Cruz (2001), La transformada de Hough es una técnica propuesta por Paul Hough (1962) para encontrar la ecuación de una línea que pase por un conjunto de n puntos en el plano *xy*, expresada de la forma,

$$y = ax + b \qquad (2.18)$$

Considerando un punto de partida (x_i, y_i), podemos pensar que existen infinitas líneas que pasan por ese punto y que cumplan con la ecuación 2.18, para valores variables de a y b. Sin embargo, cuando se tienen rectas verticales, los parámetros de la recta a y b se indefinen. Por esta razón es mejor usar los parámetros que describen una recta en coordenadas polares, ρ y θ. Redefinimos la ecuación de la recta en sus coordenadas polares:

$$y = \left(\frac{cos\theta}{sin\theta}\right) \cdot x + \left(\frac{P}{sin\theta}\right) \qquad (2.19)$$

La transformada de Hough se fundamenta en considerar las características de una recta en función de sus parámetros, y no como puntos de una imagen. Así, podemos reescribir la ecuación 2.19 en:

$$\rho = x * \cos\theta + y \cdot sin\theta \qquad (2.20)$$

Entonces, para un punto arbitrario en la imagen de coordenadas (x_0, y_0), las rectas que pasan por ese punto son los pares (ρ, θ), con ρ (distancia entre la línea y el origen) dependiente de θ. En el espacio de coordenadas polares de Hough, esto corresponde con una sinusoide única en ese punto. Si las curvas correspondientes a dos puntos se intersectasen, dicho punto se correspondería con una línea en el espacio imagen que une esos dos puntos.

El algoritmo de la transformada de Hough se basa en el uso de una matriz acumuladora, de dimensión igual al número de parámetros desconocidos, en el que cada celda representa una figura cuyos parámetros se pueden obtener a partir de la posición de la misma celda. Por cada punto en la imagen, se buscan todas las posibles figuras a las que puede pertenecer ese punto. Esto se logra buscando todas las posibles combinaciones de valores para parámetros que describen la figura. Si es así, se calculan los parámetros de esa figura, y después se busca la posición en el acumulador correspondiente a la figura definida, y se incrementa el valor que hay en esa posición. Así, la línea correspondiente a los puntos estudiados se corresponde a los parámetros definidos por la celda de máximo valor del acumulador. Veamos la implementación de este problema en Matlab para observar un ejemplo gráfico.

- *Implementación en Matlab*

Supongamos que queremos encontrar las líneas contenidas pos la región correspondiente a un 8 pre-procesada (binarizada, dilatada y erosionada). Si aplicamos el comando *Hough* sobre la imagen, éste nos devolverá una especie de histograma correspondiente al espacio polar de Hough. Esto es:

```
> [H, THETA, WHO]= HOUGH(BW);
```

Tras plotear la mátriz H devuelta, obtenemos el siguiente histograma:

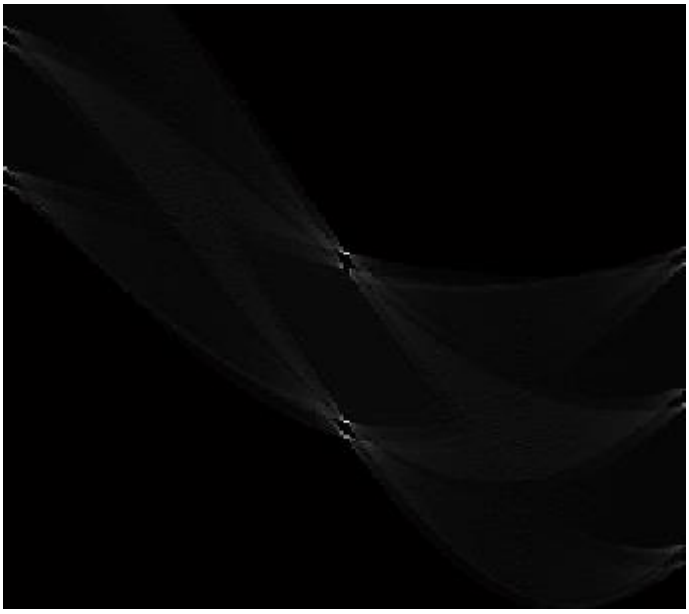

Figura 2.16: Espacio polar de Hough

En la imagen pueden apreciarse las curvas de las que hablábamos anteriormente. La función *Hough* traza, para cada punto de la región, todos los pares (ρ, θ) candidatos a formar una línea con dicho punto. Si las sinusoides correspondientes a dos pares se cortasen, significaría que esos puntos forman una línea.

Trabajamos con imágenes de 7 segmentos, por lo tanto, nos interesa extraer únicamente 7 líneas como máximo. Las líneas de mayor dimensión se corresponden a los puntos de mayor intensidad del histograma. Si quisiésemos extraer las líneas correspondientes a esos puntos, bastaría con utilizar la función *houghpeaks*:

```
> peaks=houghpeaks(H, 7 ,  'threshold',63);
```

Con esta función, extraeríamos los "picos" (indicamos un máximo de 7) del histograma que tengan un nivel de intensidad superior al umbral *"threshold"*, definido en la función (nosotros escogimos 0.63).

Nótese que por cada segmento de la región, tendríamos 2 líneas: las correspondientes con el lado interior y exterior del segmento. Para evitar extraer más de una línea por segmento, la función *houghpeaks* incorpora una función rechazadora de vecinos, *NHoodSize*, que discrimina los puntos próximos a un pico concretamente dentro de un rejilla definida por el programador. Nosotros elegimos una rejilla 99x99, y la sentencia de la función quedaría de la siguiente forma:

```
> peaks=houghpeaks(H, 7 ,  'threshold',63, 'NHoodSize', [99 99]);
```

Para extraer las lineas, utilizamos la función *houghlines*:

```
> lines=houghlines(BW, THETA, WHO, peaks);
```

El comando nos devolverá una estructura con las coordenadas iniciales y finales de cada línea extraída. Podemos pintar dichas líneas sobre la imagen inicial:

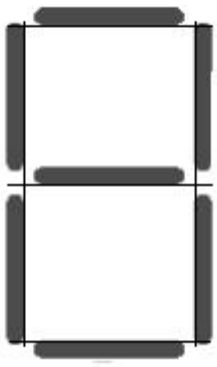

Figura 2.17: Lineas extraídas de un número en 7 segmentos

2.3.2.1 Puntos finales

Algunas regiones no podemos clasificarlas en función de las líneas extraídas. Éste es el caso de los números 2 y 5, por su simetría; y de los números 6 y 9, porque básicamente se trata de la misma región en ambos casos, pero con distinta orientación. En éstos casos nos vemos obligados a extraer una característica adicional, que serán los puntos finales de las regiones. En el caso del 6, su punto final se encuentra por encima del centroide de la región, al contrario del 9. En el caso del 2, su punto final superior queda a la izquierda del centroide, al contrario del 5.

- *Implementación en Matlab:*

Para extraer los puntos finales de una región, basta con esqueletizar la región mediante la función "*infinite thin*" del comando *bwmorph*, y extraer los puntos finales del esqueleto, usando la función *endpoints* del mismo comando. El código sería el siguiente.

```
> BWThinned = bwmorph(BW,'thin',Inf);
> EP=bwmorph(BWThinned, 'endpoints');
```

2.3.3 Clasificación

Emplearemos un árbol de decisiones para realizar la clasificación de las regiones entrantes, en función del tipo de líneas detectadas y de los puntos finales de la región. Distinguimos 3 tipos de líneas:

- Líneas horizontales: líneas correspondientes a los segmentos horizontales de los números.
- Líneas verticales simples: Líneas correspondientes a los segmentos verticales de los números.
- Líneas verticales dobles: líneas correspondientes a dos segmentos verticales que quedasen unidos tras el proceso de dilatación.

Con esto, y con la configuración de los puntos finales correspondiente a los números 2, 5, 6, 9. Siendo la configuración de líneas verticales L=[Lineas dobles verticales, líneas simples verticales, líneas horizontales], se ha desarrollado un árbol de decisión que sigue la estructura de la figura:

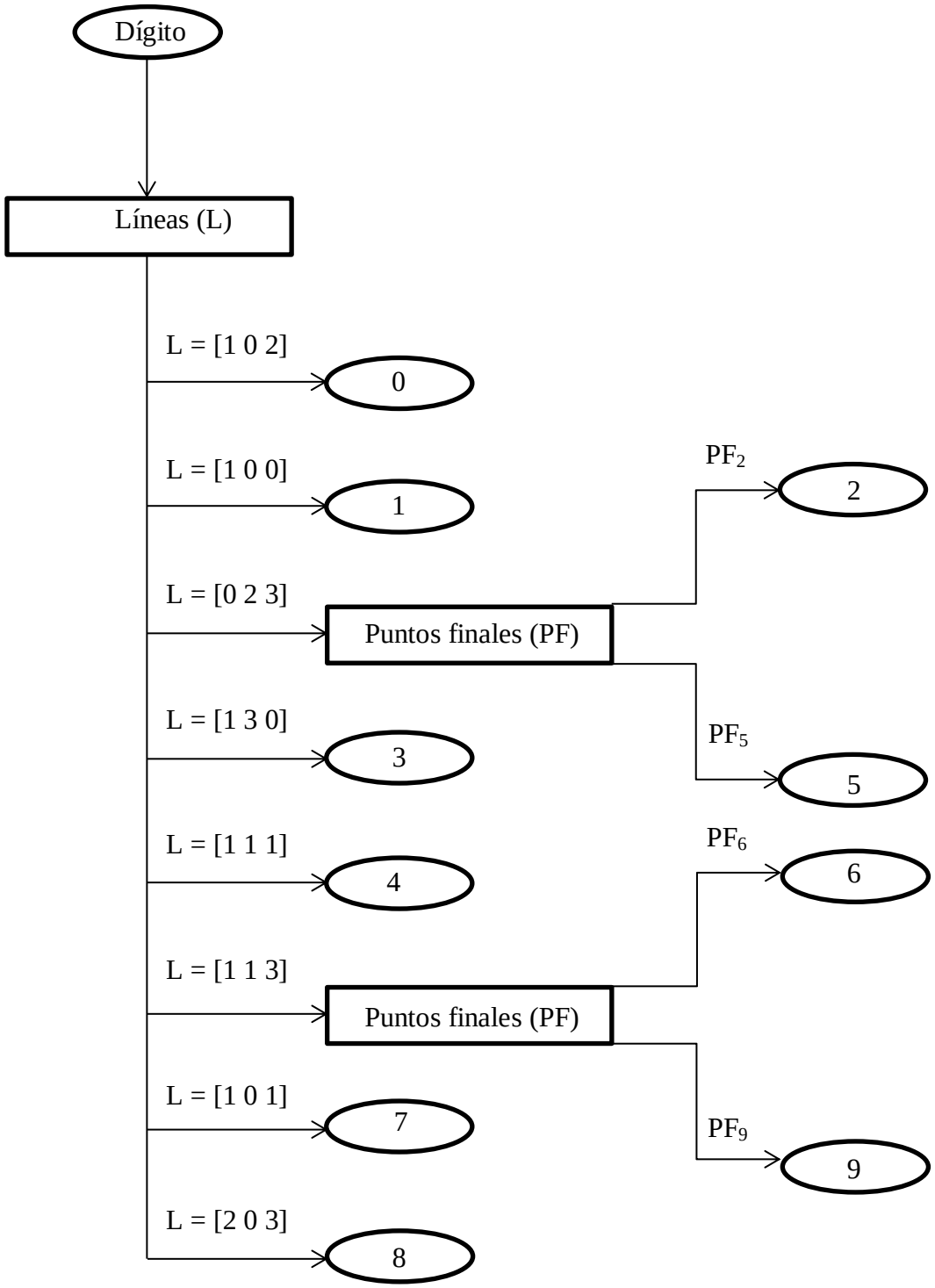

Figura 2.18: árbol de decisión para la clasificación en 7 segmentos

- *Implementación en Matlab*

Realizamos el árbol de decisión mediante sentencias *if/else*. A continuación se muestra un fragmento del mismo:

```
> if L==2&&V==0&&H==2
>     Numero=0;
>
> elseif L==1&&V==0&&H==0
>     Numero=1;
>
> elseif L==1&&V==0&&H==3
>     Numero=3;
>
> elseif L==1&&V==1&&H==1
>     Numero=4;
>
> elseif L==2&&V==0&&H==2
>     Numero=0;
>
>end
```

Como hemos dicho, para diferenciar los números 2 y 5; y los números 6 y 9; estudiamos la posición relativa de los puntos finales de la región respecto del centroide. Para el caso del 2 y el 5, estudiamos la posición relativa de la coordenada X del primero punto final respecto a la coordenada X del centroide. En las siguientes líneas se muestra el código implementado.

```
>  elseif L==0&&V==2&&H==3
>    Numero=2;
>    EP=bwmorph(BWP, 'endpoints');
>    [A B]=find(EP==1);
>
>    C=regionprops(BWP, 'centroid');
>    C=C.Centroid(1);
>
>    if B(1)<C(1)
>        Numero=2;
>    else
>        Numero=5;
>    end
```

Nota: Para ver el código en su contexto, diríjase a la sección "Detección para 7 segmentos" del Anexo I: código fuente

3. RESULTADOS

3.1 Introducción

En el presente capítulo se describen las diferentes pruebas de validación realizadas sobre el sistema OCR desarrollado, así como la exposición de los resultados obtenidos y sus correspondientes conclusiones.

3.2 Criterios de evaluación del sistema

Como se comentó en el apartado anterior, el sistema fue entrenado con 6 fuentes distintas para cada número. Para el proceso de validación, se imprimieron varios grupos de caracteres en papel, concretamente las mismas fuentes con las que se entrenó el sistema, pero a alturas diferentes (100, 80, 60 y 40), contando con un total de 240 muestras para analizar. Los folios fueron fotografiados con una cámara de 5 megapíxeles, en buenas condiciones de iluminación natural y sin sombras. Para evaluar el correcto funcionamiento del sistema, analizaremos el comportamiento el sistema cuando el dígito que espera es el mismo que el del conjunto de muestras que se le envía, pero también en el caso contrario. Así tendremos cuatro posibilidades de reconocimiento:

- Verdadera reconocida (VR): el dígito de las muestras de entrada es el que espera el sistema y lo reconoce.
- Verdadera no reconocido (VN): el dígito de las muestras de entrada es el que espera el sistema, pero no lo reconoce.
- Falsa reconocida (FR): el dígito de las muestras de entrada no es el que espera el sistema, pero lo reconoce como bueno.
- Falsas no reconocidas (FR): el dígito de las muestras de entrada no es el que espera el sistema y no lo reconoce.

Así, podemos obtener la tasa de éxito de reconocimiento como el número de muestras verdaderas reconocidas sobre el total de muestras verdaderas

$$ER = \frac{VR}{V}, \quad para\ V = VR + VN \qquad (3.1)$$

Además, evaluaremos la tasa éxito de "no fallo" como el número de muestras falsas no reconocidas sobre el número de muestras falsas total.

$$EN = \frac{MR}{M}, \quad para \ M = MR + MN \qquad (3.2)$$

Así, podemos definir la tasa de éxito global como la media de las tasas de éxito de reconocimiento de cada clase:

$$E = \frac{1}{N}\sum_{i=1}^{N} ER_i \qquad (3.4)$$

De forma que el reconocimiento y el falso reconocimiento tengan el mismo peso en el éxito global.

3.3 Resultados obtenidos

Durante la etapa de desarrollo del sistema OCR se estudiaron los resultados extrayendo como características momentos invariantes de Hu, momentos centrales y momentos centrales normalizados. En los siguientes apartados se exponen los resultados obtenidos para cada uno de ellos.

3.3.1 Momentos Invariantes de HU

Uno de los métodos más extendidos en el reconocimiento óptico de caracteres es el estudio de los momentos invariantes de HU de los objetos de número. La primera evaluación del sistema se realizó extrayendo dichos momentos. Obtendremos la tasa de éxito de reconocimiento y de "no fallo" para cada clase del clasificador (cada número), así como la tasa de éxito global final. En la siguiente tabla se muestran los resultados para K=1 (vecino más próximo).

Clase	VR	VN	FR	FN	ER(%)	EN(%)
0	24	0	0	216	100.00	100.00
1	19	5	0	216	79.17	100.00
2	16	8	14	202	70.00	94.61
3	23	1	8	208	95.65	97.12
4	21	3	0	216	86.96	100.00
5	9	15	13	202	65.22	95.19
6	5	19	10	206	17.39	96.15
7	24	0	0	216	100.00	100.00
8	23	1	6	211	95.83	98.07
9	16	8	14	202	66.67	91.30
Éxito global (%)					**77.67**	

Tabla 3.1: Resultado con momentos de HU

3.3.3.1 Diferenciación entre 6's y 9's

Debido a que los momentos de Hu son invariantes de Hu, el clasificador tiene serios problemas para diferenciar el 6 del 9, teniendo un porcentaje de éxito de reconocimiento bastante bajo para sus respectivas clases. Esto se solucionó estudiando la posición relativa del punto final del esqueleto de la región respecto al centroide de la misma, de forma que si el punto final está por encima del centroide el número se clasificaría como 6, y en el caso contrario, como 9. A continuación se muestra el algoritmo implementado:

```
> if Num==6||Num==9
>
>     A = bwmorph(bw5,'endpoints');
>     [P b]=find(A==1);
>
>     C=regionprops(bw5, 'centroid');
>     C=C.Centroid(1);
>
>     if P>C
>         Num=9;
>     else
>         Num=6;
>     end
>
> end
```

Mediante la función *bwmorph* obtenemos la matriz de puntos finales de una posible región. Almacenamos la posición de los puntos finales en una variable y lo comparamos con la componente y del centroide obtenido por *regionprops* mediante sentencias *if/else*.

A continuación se muestran los resultados de la clasificación mediante la extracción de momentos invariantes de Hu para los distintos valores de K, tras haber implementado el algoritmo de distinción entre 6's y 9's, para K=1.

Clase	VR	VN	FR	FN	ER(%)	EN(%)
0	24	0	0	216	100.00	100.00
1	19	5	0	216	79.17	100.00
2	16	8	14	202	70.00	94.61
3	23	1	8	208	95.65	97.12
4	21	3	0	216	86.96	100.00
5	9	15	13	202	65.22	95.19
6	5	19	10	206	78.26	100.00
7	24	0	0	216	100.00	100.00
8	23	1	6	211	95.83	98.07
9	16	8	14	202	83.33	99.52
Éxito global (%)					**77.67**	

Tabla 3.2: Resultados con momentos de Hu con discriminación de 6's y 9's

Se puede que las clases 6 y 9 están mucho más desacopladas después de implementar el algoritmo discriminatorio.

Avanzado el desarrollo del sistema, se llegó a la conclusión de que la extracción de momentos invariantes de Hu para la clasificación carecía de lógica. Durante la grabación de las variables, la cámara permancerá fija, por lo que la imagen capturada tendrá siempre las mismas propiedades de orientación y traslación. Además, el proceso de cortado y normalizado de las regiones hace que no se presenten variaciones en la escala de los números. Tras descartar los momentos invariantes, se decidió estudiar el comportamiento del clasificador extrayendo de la imagen momentos centrales.

3.3.2 Momentos centrales

Tras rechazar la extracción de los momentos invariantes de Hu, se decidió apostar por momentos centrales de la imagen, concretamente los 9 momentos centrales de orden menor e igual a 3. Para un clasificador Knn de K=1, se obtuvieron los siguientes resultados:

Clase	VR	VN	FR	FN	ER(%)	EN(%)
0	22	2	3	213	95.65	99.04
1	23	1	5	211	95.83	98.06
2	16	8	2	214	70.00	99.53
3	24	0	5	211	100.00	98.08
4	24	0	2	214	100.00	99.52
5	24	0	2	214	100.00	99.52
6	19	5	0	216	82.61	100.00
7	24	0	0	216	100.00	100.00
8	23	1	5	211	95.83	98.07
9	21	3	2	214	87.50	99.52
Éxito global (%)					**92.74**	

Tabla 3.3: Resultados con momentos centrales

Decir que el algoritmo diferenciador de 6's y 9's no se ha implementado en éste caso, puesto que los momentos centrales de imágenes digitales no poseen invariancia ante la rotación.

Pese a contar con una tasa de éxito global bastante superior, observamos que las tasas de reconocimiento de algunas clases son relativamente bajas. Esto es debido a que los momentos de orden 3 tienen mucho más peso que los de orden 0 y 1. Recordemos que los clasificadores Knn funcionan calculando la distancia euclidea entre todos los parámetros de los objetos de entrada y los de muestra. Por tanto, es lógico pensar que los momentos de orden 3 tendrán un poder de decisión en la clasificación mucho mayor a la de los momentos de orden 1 o 2. Por ésta razón, decidimos recurrir a los momentos centrales normalizados para la clasificación.

3.3.3 Momentos centrales normalizados

Como se expuso en la ecuación 2.16 del capítulo "Reconocimiento de caracteres", el momento central normalizado de orden $n+m$ es el cociente entre el momento central de orden $n+m$ y el momento central de orden 0, elevado a un coeficiente φ dependiente del orden del momento superior. Con ésta normalización conseguimos que el poder de decisión de los momentos en la clasificación se equilibre, independientemente del orden de los mismos. A continuación se muestran los resultados obtenidos para un clasificador Knn para K=1.

Clase	VR	VN	FR	FN	ER(%)	EN(%)
0	22	2	0	216	95.65	100.00
1	21	3	0	216	87.50	100.00
2	24	0	0	216	100.00	99.53
3	24	0	3	213	100.00	99.04
4	24	0	0	216	100.00	100.00
5	24	0	7	209	100.00	97.12
6	21	3	2	214	91.30	99.52
7	24	0	0	216	100.00	100.00
8	23	1	5	211	95.83	98.07
9	21	3	2	214	87.50	99.52
Éxito global (%)					**95.77**	

Tabla 3.4: Resultados con momentos centrales normalizados para k=1

Pese a obtener unos resultados algo inferiores a los esperados, se pudo comprobar que tanto la tasa de éxito global como la particular de cada clase se vieron mejoradas con la normalización de momentos.

Cuando definimos a los clasificadores Knn o de "K vecinos más próximos, comentábamos que la clasificación dependía del parámetro K escogido. Se obtenían los K vecinos más cercanos y se clasificaba al objeto entrante a la clase más repetida. Como no hay una regla escrita para la obtención del valor óptimo de K, decidimos estudiar el comportamiento del clasificador para K={3,5,7}, además del ya estudiado K=1. A continuación se exponen los resultados obtenidos para cada valor de K:

K=3						
Clase	**VR**	**VN**	**FR**	**FN**	**ER(%)**	**EN(%)**
0	22	2	0	216	95.65	100.00
1	15	9	0	216	62.50	100.00
2	22	2	5	211	95.00	98.10
3	24	0	3	213	100.00	99.04
4	24	0	0	216	100.00	100.00
5	24	0	8	208	100.00	96.63
6	21	3	2	214	91.30	99.52
7	24	0	0	216	100.00	100.00
8	23	1	5	211	95.83	98.07
9	21	3	2	214	87.50	99.52
Éxito global (%)					**92.78**	

Tabla 3.4: Resultados con momentos centrales normalizados para k=3

K=5						
Clase	**VR**	**VN**	**FR**	**FN**	**ER(%)**	**EN(%)**
0	22	2	0	216	95.65	100.00
1	15	9	0	216	62.50	100.00
2	20	4	5	211	85.00	98.10
3	24	0	3	213	100.00	98.56
4	22	2	25	191	91.30	88.46
5	24	0	8	208	100.00	96.63
6	21	3	2	214	91.30	99.52
7	24	0	0	216	100.00	100.00
8	23	1	5	211	95.83	98.07
9	22	2	2	214	91.30	99.52
Éxito global (%)					**91.30**	

Tabla 3.5: Resultados con momentos centrales normalizados para k=5

K=7						
Clase	**VR**	**VN**	**FR**	**FN**	**ER(%)**	**EN(%)**
0	22	2	0	216	95.65	100.00
1	15	9	0	216	62.50	100.00
2	20	4	5	211	85.00	98.10
3	24	0	6	210	100.00	97.60
4	15	9	0	216	62.50	100
5	21	3	7	209	91.30	96.15
6	21	3	2	214	91.30	99.52
7	24	0	0	216	100.00	100.00
8	23	1	5	211	95.83	98.07
9	22	2	2	214	87.50	99.52
Éxito global (%)					**87.16**	

Tabla 3.6: Resultados con momentos centrales normalizados para k=7

En general, se observó que a medida que incrementábamos el valor de K, la tasa de reconocimiento decrementaba, tanto la global como la particular de cada clase. Esto es debido a que, cuando evaluamos el clasificador con el mismo tipo de muestras con el que se entrenó, el vecino más cercano siempre será él mismo, y cuanto mayor sea el número de vecinos del que depende la decisión, habrá mayor probabilidad de fallo.

Se decidió entonces ver el verdadero potencial del sistema analizando su comportamiento frente a la entrada de caracteres de fuente distinta a la de las muestras de entrenamiento. En total, se realizó el reconocimiento de 20 fuentes distintas por número, a tamaño único de 80, contando con un total de 200 muestras de evaluación. Los resultados obtenidos para K=1 fueron los siguientes.

Clase	VR	VN	FR	FN	ER(%)	EN(%)
0	20	0	0	180	100	100.00
1	17	3	0	180	85.00	100.00
2	20	0	5	175	100.00	98.71
3	19	1	0	180	95.00	100.00
4	20	0	0	180	100.00	100.00
5	17	3	5	175	85.00	97.12
6	19	1	0	180	95.00	100.00
7	19	1	4	176	95.00	98.72
8	20	0	0	180	100.00	100.00
9	20	0	0	180	100.00	100.00
Éxito global (%)					**95.83**	

Tabla 3.7: Resultados para varias fuentes k=1

Decir que la razón de que la tasa de éxito para múltiples fuentes sea superior que en la evaluación para las fuentes de entrenamiento es debido a que las imágenes que se tomaron para el reconocimiento no fueron fotografías, sino directamente editadas en aplicaciones gráficas. Aun así, se puede comprobar que la tasa de éxito es bastante elevada, teniendo en cuenta la cantidad de fuentes con las que se evaluó el programa.

Es en éste caso cuando sí cobra sentido evaluar el comportamiento del clasificador para distintos valores de K, pues no usamos las mismas fuentes en la evaluación que en el entrenamiento. En las siguientes tablas se muestran los resultados para K={3, 5, 7}.

K=3						
Clase	**VR**	**VN**	**FR**	**FN**	**ER(%)**	**EN(%)**
0	20	0	0	180	100	100.00
1	13	7	0	180	65.00	100.00
2	17	3	5	175	85.00	97.42
3	19	1	0	180	95.00	100.00
4	20	0	0	180	100.00	100.00
5	17	3	94.44	100	100.00	95.57
6	19	1	0	180	95.00	100.00
7	19	1	4	176	95.00	98.72
8	20	0	0	180	100.00	100.00
9	20	0	0	180	100.00	100.00
Éxito global (%)					**93.50**	

Tabla 3.8: Resultados para varias fuentes k=3

K=5						
Clase	**VR**	**VN**	**FR**	**FN**	**ER(%)**	**EN(%)**
0	19	1	0	180	95.00	100.00
1	13	7	0	180	65.00	100.00
2	16	4	5	175	80.00	97.42
3	19	1	0	180	95.00	100.00
4	16	4	0	180	80.00	100.00
5	17	3	94.44	100	100.00	95.57
6	19	1	0	180	95.00	100.00
7	19	1	4	176	95.00	98.72
8	20	0	0	180	100.00	100.00
9	20	0	2	178	100	99.38
Éxito global (%)					**90.50**	

Tabla 3.8: Resultados para varias fuentes k=5

K=7						
Clase	**VR**	**VN**	**FR**	**FN**	**ER(%)**	**EN(%)**
0	19	1	0	180	95.00	100.00
1	13	7	0	180	65.00	100.00
2	16	4	5	175	80.00	97.42
3	19	1	0	180	100.00	100.00
4	16	4	0	180	80.00	100.00
5	17	3	6	174	100.00	95.57
6	18	2	15	165	90.00	91.77
7	19	1	4	176	95.00	98.72
8	20	0	0	180	100.00	100.00
9	18	2	3	177	90.00	98.77
Éxito global (%)					**89.5**	

Tabla 3.9: Resultados para varias fuentes k=7

En ésta ocasión sí se pudo observar que las tasas de éxito en el reconocimiento de algunas clases oscilaban en función de K, pero se siguió observando que la tasa de mayor éxito se seguía correspondiendo con K=1.

Finalmente, el clasificador implementado en nuestro sistema OCR fue un Knn para K=1, entrenado para recibir los momentos centrales normalizados de los objetos entrantes.

3.4 Conclusiones

Al comenzar éste trabajo, la primera meta que nos pusimos como objetivo era la de construir un sistema OCR de reconocimiento multi-fuente. Como ya hemos definido, el objetivo de éste proyecto es el de diseñar un sistema de reconocimiento de variables universal, y que gracias a esto sea compatible con el mayor número de sistemas de anestesia posible. Hemos entrenado al sistema con 6 fuentes distintas de números, y somos conscientes de que si incrementamos el número de muestras, el número de fuentes tipográficas reconocibles incrementará aún más. La cantidad de muestras utilizadas por nuestro sistema, frente a las comúnmente utilizadas por sistemas de reconocimiento óptico de caracteres manuscritos, es ridícula.

Sin embargo, de poco nos sirve diseñar un sistema OCR capaz de leer gran cantidad de fuentes de caracteres, si la tasa de reconocimiento sobre la fuente objetivo se decrementa. Estaríamos descentralizando el problema. Por esa razón, hemos decidido entrenar a nuestro sistema con las fuentes tipográficas más comunes en paneles digitales, con el fin de asegurar una tasa de reconocimiento alto sobre est4os dispositivos.

4. IMPLEMENTACIÓN EN TIEMPO REAL

4.1 Introducción

El presente capitulo describe los procedimientos necesarios para la implementación dinámica del sistema OCR desarrollados. Así mismo, se describen los problemas encontrados durante la adaptación del software a un sistema de captación en tiempo real, así como los métodos utilizados para integrar el mismo en el programa *ancyc.m*, del proyecto ANESPLUS.

4.2 Principios del procesamiento de video en Matlab

Para poder realizar la adquisición de imágenes a partir de un sistema de visión instalado en el equipo desde Matlab, es necesario crear un objeto de video asociado al mismo. Para ello, podemos introducir la siguiente línea de código:

```
> obj = videoinput('winvideo',1);
```

De esta forma "conectamos" una cámara instalada en nuestro ordenador a Matlab, pudiéndose alterar distintos parámetros configuración del dispositivo, parámetros como la resolución, el tratamiento de la señal de video, el número de fotografías por disparo (*framesPerTrigger*), etc.

Para la adquisición de imágenes, Matlab ofrece dos alternativas: mediante capturas, o mediante grabación o extracción de *frames*. Se ha comprobado que la mayoría de cámaras tardan un tiempo en adaptarse a la luminosidad del entorno, una vez iniciado el proceso de captura. Este hecho, unido a que el tiempo de fotografiado será siempre mayor que al de la extracción de *frames* de video, nos hace descartar la primera opción. Para la extracción de frames, es necesario iniciar la grabación con el objeto de video. Esto puede realizarse mediante la función *start* de Matlab.

```
> start(obj);
```

Posteriormente, previsualizamos el objeto de video mediante el comando *preview*:

```
> preview(obj);
```

Para la extracción de un *frame* a tiempo real, utilizamos el comando *getsnapshot* sobre el objeto:

```
> getsnapshot(obj);
```

Es muy importante utilizar este comando después de haber inicializado la previsualización del objeto, para que la cámara pueda adaptarse a las condiciones de luminosidad del entorno. De lo contrario, la calidad de la imagen será muy pobre.

4.3 El problema del solapamiento de dígitos

Al realizar las primeras grabaciones sobre los monitores de variables del proceso anestésico, pudimos apreciar que, en ocasiones, durante el escaneado continuo de dígitos en pantalla se llega a producir solapamiento. Esto supone que, cuando la variable cambia de valor, el nuevo dígito es escaneado antes de producirse el borrado del anterior. Este fenómeno puede apreciarse en la siguiente imagen.

Figura 4.1: Solapamiento entre dígitos

En la imagen puede apreciarse el solapamiento producido durante el tránsito de díditos, concretamente de un 93 a un 92, en el índice PSI. Aunque se aprecia que las intensidades no son equivalentes, no es difícil imaginar que una mala umbralización podría provocar fallos garrafales en el reconocimiento de la región. Es por eso que intentamos evitar que la región a procesar pudiera llegar a ser como la de la imagen mostrada.

Para evitar procesar una imagen de dígitos solapados, la primera medida a tomar era tomar varias imágenes por disparo. Se estimó que el tiempo entre el escaneado de un número

y el borrado del anterior no superaba los 0.05 segundos. Teniendo en cuenta que el video con el que se trabajaría no superaba una tasa de 15 imágenes por segundo, se dedujo que era imposible obtener la misma transición en dos imágenes seguidas.

Se tuvo la idea de tomar 5 imágenes para cada procesamiento, transformarlas a imágenes binarias y hacer una imagen promedio de las mismas. A continuación se puede ver una serie de 5 imágenes consecutivas binarizadas:

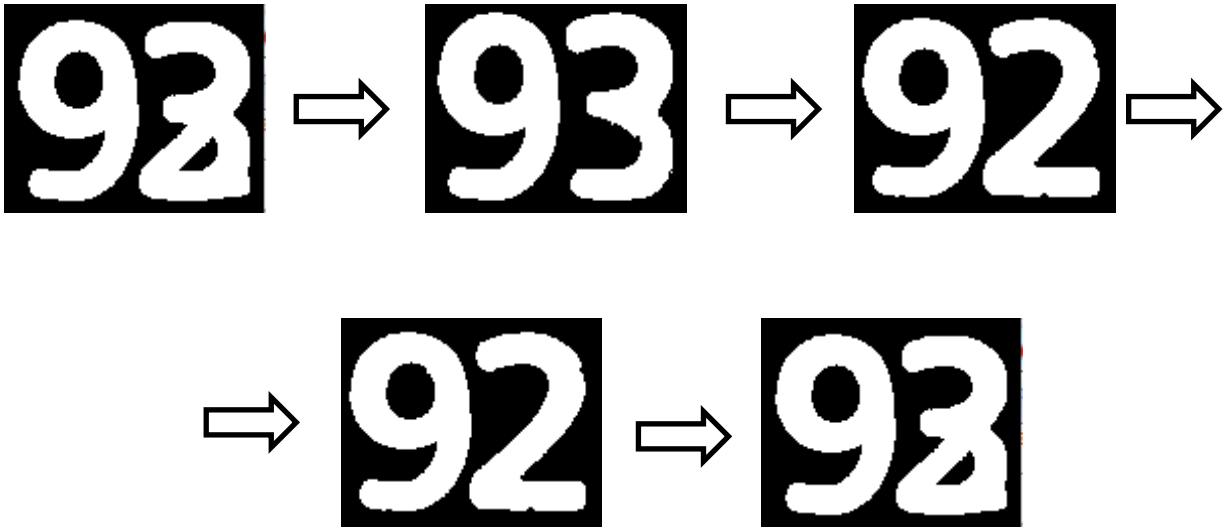

Figura 4.2: secuencia binariazada de 5 imágenes consecutivas

Tras binarizar el array de imágenes, obtendríamos la imagen promedio, calculando la media de intensidades por pixel. Dicha imagen tendría la siguiente forma:

Figura 4.3: Promedio de una secuencia de imágenes

Nuestro objetivo es encontrar el número más repetido en el conjunto de imágenes (no la imagen más repetida). Éste es un problema más complejo del que cabría suponer.

En un principio se intentó binarizar la imagen promedio, usando como umbral la intensidad máxima de la imagen (255). Esto es un error, pues la intensidad máxima corresponde a las regiones coincidentes entre números, y no estaríamos segmentando el número que nos interesa. A continuación, se pensó que si buscamos el umbral que nos diese, en la imagen promedio, el mismo número de regiones conectadas que tiene cada una de la imagen promedio, se conseguiría "separar" el número más repetido de la imagen. Esto nos conduce a un error pues el conjunto de regiones coincidentes entre números no tiene por qué ser una región fragmentada (pese a que en la imagen anterior si lo parezca). Se concluyó que es imposible obtener la región más repetida sobre la imagen promedio, a no ser que obtengamos una característica de cada imagen de la secuencia. Es ahí donde entra en juego la correlación.

La correlación entre imágenes indica el grado de semejanza entre dos imágenes binarizadas, de mismas dimensiones NxN, determinada a partir de la siguiente expresión:

$$c = I_1 * I_2 = \frac{1}{K}\sum_{i=1}^{N}\sum_{j=1}^{N} I_1(i,j)I_2(i,j) \qquad (4.1)$$

donde K es un coeficiente de normalización. Cuanto mayor sea c, más similares serán las imágenes. Para saber si un número está contenido en una región solapada, se estudia el nivel de correlación que tiene la imagen de un número consigo misma (correlación máxima), y después se compara dicho nivel de correlación con la que tiene la imagen de un número con la imagen solapada. Si los niveles son similares (aproximadamente al 95%), consideramos que la región está repetida. Veamos un ejemplo:

Supongamos que queremos saber si la región ocupada por el 92 está contendida en el solapamiento entre el 92 y el 93.

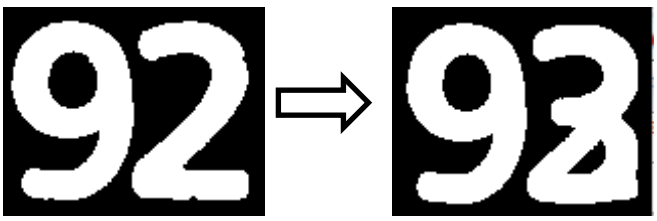

Figura 4.4: Estudio de correlación entre regiones.

Estudiando el nivel de correlación que tiene el 92 consigo mismo, y comparándolo con el nivel que se tiene con la imagen solapada, se obtendría que la región ocupada por el 92 esta "contenida" en la imagen solapada. Lo interesante de éste análisis es que si se hiciese a la inversa, no obtendríamos los mismos resultados.

Lo que hacemos es comparar la región contenida en una imagen con todas las regiones de la secuencia, para todas las imágenes. Observaríamos que tendríamos un total de 3 regiones diferentes, la ocupada por el 92, la ocupada por el 93, y la ocupada por el conjunto solapado. Si encontramos la región más repetida, podríamos obtener el umbral con el que binarizar la imagen promedio, a partir de la siguiente expresión:

$$Umb = \frac{N_{MAX}}{N} \quad (4.2)$$

Siendo:

N_{max}: número de repeticiones de la región más repetida.

N: número total de imágenes de la secuencia.

Binarizamos con dicho umbral la imagen promedio y obtendremos la región correspondiente al número más repetido.

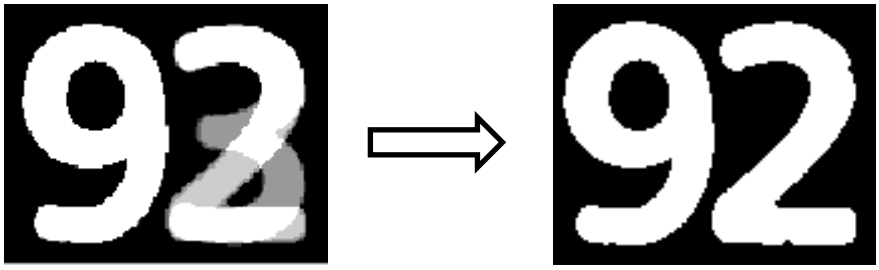

Figura 4.5: obtención del número más repetido.

- *Implementación en Matlab*

Contando con un conjunto de imágenes ya binarizadas en una array de tamaño K, podemos obtener la imagen promedio *bwSum* mediante el siguiente bucle for:

```
> for i=1:K
>
```

```
>   if i==1
>     bwSum=bw(:,:,1)/K;
>   else
>     bwSum=bwSum+bw(:,:,i)/K;
>   end
>
> end
```

Estudiamos el nivel de correlación de todas las imágenes consigo mismas y entre sí:

```
> for i=1:K
>        corP=correlAbs(bw(:,:,i),bw(:,:,i));
>
>     for j=1:K
>      cor=correlAbs(bw(:,:,i),bw(:,:,j));
>
>      if cor/corP>0.95;
>          puntos(i)=puntos(i)+1
>      else
>          puntos(i)=puntos(i)
>      end
>
>    end
> end
```

Si la relación de los niveles de correlación entre imágenes supera el 95%, damos un punto a esa imagen, de forma que obtengamos un vector de puntuación. Para obtener el nivel de correlación entre dos imágenes binarias, hemos diseñado la función *correlAbs*, que contiene el siguiente código:

```
> [n, m]=size(bw1);
>  c=sum(reshape(bw1.*bw2, n*m, 1));
```

Mediante la función *reshape*, obtenemos en forma de vector el producto escalar de las dos imágenes binarias entrantes, de longitud *nxm*. Mediante el comando *sum* hacemos la suma de todos los elementos del vector que devuelve *reshape*, para así obtener la correlación absoluta c, tal y como expresa la ecuación (4.1).

Para obtener la imagen promedio, obtenemos el valor máximo del vector de puntuación generado anteriormente, y calculamos el umbral como en la ecuación (4.2)

```
>  MaxRep=max(puntos);
>  umb=MaxRep/K;
>  bwF=bwSum>0.95*umb;
```

4.4 Ejecución dinámica: uso de *timers*

Matlab ofrece la posibilidad de utilizar objetos temporizadores (*timer objects*) para la ejecución de funciones de llamada de retorno (*callbacks*) asociadas al mismo en distintos instantes. Para nuestro sistema, sería de utilidad utilizar un temporizador para ejecutar del programa del sistema OCR desarrollado.

El bucle de control de Ancyc.m realiza una lectura de las variables de interés del proceso anestésico cada 5 segundos. Decir que, una vez depurado el código de cada una de las funciones de nuestro sistema, el tiempo de procesamiento es, en media, de 1.8 segundos, por lo que cumplimos con los requerimientos de eficiencia del programa principal.

En Matlab, los *timers* admiten 3 tipos de ejecución:

- "FixedRate" (o de tiempo fijo): El tiempo del periodo entre ejecuciones se inicia inmediatamente después de que la llamada a la función "callback" del temporizador se agregue a la cola de ejecución.
- "FixedDelay (o de retardo fijo): El tiempo del periodo entre ejecuciones comienza cuando la función "callback" comienza a ejecutarse, después de cualquier tiempo de retraso debido a los retrasos en la cola de ejecución.
- "FixedSpacing" (o de espaciado fijo): El tiempo del periodo entre ejecuciones inicia la cuenta justo cuando acaba la ejecución de la función "callback" asociada al timer.

En el siguiente esquema puede apreciarse las diferencias principales entre los distintos modos de ejecución:

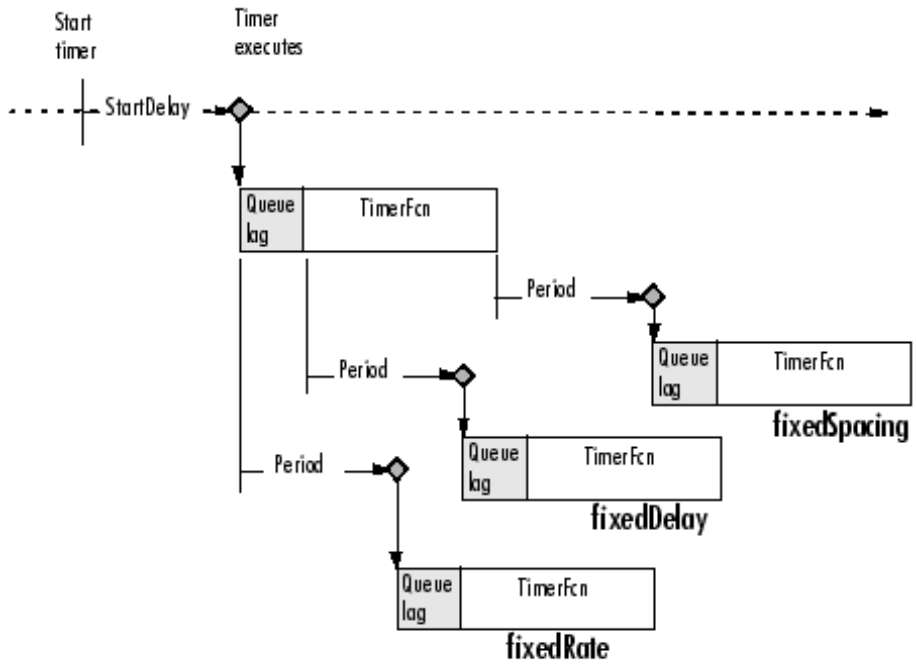

Figura 4.7: Diferencias entre los modos de ejecución de la función "timer". Fuente: Mathworks.com

En vistas a los requerimientos del sistema principal, decidimos que el modo de ejecución óptimo para el *timer* asociado a nuestro sistema era "*fixedRate*" o de tiempo fijo. La función de retorno de la llamada o "*callback*" asociada al *timer* será el script principal de nuestro sistema OCR, "Reconocer.m". En la siguiente línea de código puede verse la declaración del *timer*, así como la selección del modo de ejecución.

```
> timer_OCR=timer('TimerFcn', 'Variable=reconocer()', 'period',5);
> set(timer_OCR,'ExecutionMode', 'fixedRate');
```

Para la activación del timer:

```
> start(timer_OCR)
```

Para la desactivación del timer:

```
> stop(timer_OCR)
```

Una vez definida la ejecución dinámica del programa, el último paso será la integración en el programa Ancyc.m.

4.5 Integración en Ancyc.m: construyendo la clase *camara*

Ancyc.m es el programa principal del proyecto ANESPLUS. Para dar por finalizado nuestro proyecto, hemos de realizar la integración de nuestro sistema OCR en el código principal, para que sirva como función de reconocimiento en tiempo real.

Ancyc.m está desarrollado en lenguaje Matlab orientado a objetos, siguiendo las directrices de dicha programación para las versiones anteriores a 2008. Para realizar la integración construiremos la clase *cámara*, a la que irán asociados todas las funciones y parámetros de nuestro sistema.

En Matlab, una clase consta de los siguientes elementos:

- Atributos: variables que almacenan el estado de un objeto de esa clase. Se accede a ellos como a los campos de una estructura de Matlab, y sólo se permite acceder a los mismos mediante métodos propios de la clase.

- Métodos: Son las únicas funciones que están autorizadas a acceder directamente a los atributos de un objeto, y se construyen como funciones comunes de Matlab, cada una en un fichero.m. Reciben como primer parámetro un objeto de la clase correspondiente, pudiendo el resto ser variables de estado del objeto u otros parámetros generados por métodos anidados.

En el lenguaje clásico de Matlab, para realizar una clase lo primero que ha de hacerse es un directorio con el nombre de la clase precedido del carácter arroba *(@Nombre_Clase*). Dentro de este directorio integraremos todos los métodos asociados a la clase (en nuestro caso, los scripts desarrollados para nuestro sistema OCR) y el constructor de la misma. El constructor consiste en la función responsable de crear objetos de clase, a partir de una estructura cuyos campos constituyen los futuros atributos de estado del objeto. El nombre del constructor deberá de coincidir con el nombre de la clase (en este caso, el nombre del directorio) cuyo nombre debe coincidir con el de la clase. Los atributos de estado pueden pasarse como parámetros de entrada al constructor, o bien declararse en el interior del constructor.

En nuestro caso, el único de parámetro de entrada será la calibración inicial de la zona de captura. El resto de parámetros seguirán una configuración por defecto, que será

modificable a partir de un método asociado a la clase *cámara*. Como decíamos, declaramos las variables de estado como si de una estructura se tratase (para un objeto *cam*, *cam.Nombre_Variable*) En el siguiente código puede verse la estructura del constructor de la clase:

```
> function cam=camara(coor)
>
> cam.coor=coor;
> cam.modo=1;
> cam.cifras=Nan;
> cam.decimales=Nan;
>
> timer_OCR=timer('TimerFcn', 'Variable=reconocer(cam)', 'period',5);
> set(timer_OCR,'ExecutionMode', 'fixedRate');
> cam.t=timer_OCR;
>
> obj = videoinput('winvideo',1, 'YUY2_320x240');
> set(obj, 'ReturnedColorSpace', 'RGB');
> cam.obj=obj
> start(obj);
>
> end
```

Para los métodos, la estructura será la misma que la diseñada para el sistema OCR, con la salvedad de que ahora, al utilizar Matlab orientado a objetos, el primer parámetro de todas las funciones será el objeto creado de la clase *cámara*.

Para modificar las variables de estado de la clase, y así poder cambiar la configuración del sistema de reconocimiento, la única forma de realizarlo sería devolviendo el objeto modificado como resultado. Esto es debido al fenómeno de la inmutabilidad, que provoca la imposibilidad de modificar parámetros internos de una función.

Una solución alternativa que evita el problema de la inmutabilidad consiste en el uso de una técnica alternativa de implementación de los atributos conocida como *closure*, basada en el uso de funciones anidadas. Sin embargo, este sistema no llego a implementarse, pues implicaba la modificación de todas las funciones ya diseñadas del sistema OCR. Sin embargo, se estudiará la posibilidad de implementar este método en futuras mejoras del software.

En el siguiente código puede observarse un extracto del método propuesto para el cambio de configuración:

```
> function cam=cambiar_configuración(cam)
> Modo=input('Selección de modo: Reconocimiento...
> múltiple(1),Reconocimiento > individual(0):');
>
>        if Modo==1
```

```
>              %im=getsnapshot(obj);
>              im=imread('video1.jpg');
>              [coor, Fconv] = cali(im);
>              %Ir al timer
>              [PSI, bw]=Reconocer(Modo, coor, 1,  1, 1);
>         else
>
.
.
.
> end
```

Nota: esto es sólo un fragmento del método implementado. Si se quisiese ver en su contexto, diríjase al Anexo I: código fuente

4.5.1 Activación del timer

Puede observarse que la declaración del timer del sistema se realiza en el constructor de la clase *cámara*. Para poder activar el timer y que se inicie la ejecución en tiempo real del Sistema OCR, es necesario diseñar un método como el siguiente:

```
> function tempOn(cam)
>
>     start(cam.t)
>
> end
```

La función no devuelve ningún dato, sino que inicia la ejecución periódica del script principal del sistema OCR desarrollado (Puede verse el programa completo en el Anexo I de esta memoria, así como una descripción detallada de cada script), el cual actualiza cada 5 segundos el valor de la variable a reconocer. Así mismo, es necesario un método adicional para detener el temporizador cuando finalice el reconocimiento. Este método también sirve para cerrar la previsualización de la salida del objeto de video, así como para detenerlo.

```
> function tempOff(cam)
>
>     stop(cam.t)
>     stop(cam.obj)
>     close(cam.obj)
>
> end
```

5. CONCLUSION

The main objective of this project was to design a detection and storage system for monitored variables, which would be adaptable to any type of monitor, and therefore to any type of character printed by these devices. This involved the development of a multi-font optical character recognition system, which had a high recognition rate with multiple fonts. We believe that we have met the expectations successfully. We have not adapted the system for recognition of a large number of fonts, but we prefer to ensure a high recognition rate on the devices on which the software is used. We would have liked to improve the recognition rate of the system even more, but we consider that the rate obtained is quite.

In fact, we would like to say that I feel sorry to leave the project. Aspects like the development of a GUI (graphical user interface), the application integration in Ancyc program, or simply improve the developed system are tasks that we wish to implement, but while we considered it unfeasible due to the time available.

Strictly speaking the formative nature of the final project, it has gotten I start to see the world of work in a different way than the saw before, arousing my interest in areas such as medical robotics and biomedical engineering. I hope to enjoy my future work in the same way as I did making it work.

6. BIBLIOGRAFÍA

- A. Barrientos et al., 2007. Fundamentos de Robótica (2a ed). McGraw-Hill.
- Abdelmalik Moujahid, Iñaki Inza y Pedro Larrañaga, 2000. Clasificadores K-NN. Departamento de Ciencias de la Computación e Inteligencia Artificial Universidad del País Vasco.
- Arturo de la Escalera, 2001. Visión por Computador: Fundamentos y Métodos, Ed. Prentice Hall.
- Marcos Martín, 2002. Descriptores de imagen. Disponible en la URL: http://poseidon.tel.uva.es/~carlos/ltif10001/descriptores.pdf
- E. Trucco, A. Verri, 1998. Introductory Techniques for 3-D Computer Vision. Ed. Prentice Hall.
- Gonzalo Pajares, Jesús M. de la Cruz, 2001. Visión por computador: imágenes digitales y aplicaciones. RA-MA Editorial.
- Luciano da Fontoura Costa, Roberto Marcondres Cesar Jr, 2000. Shape Analysis and Classification: Theory and Practice. CRC PRESS.
- Método del valor umbral. Wikipedia. Disponible en la URL: http://es.wikipedia.org/wiki/M%C3%A9todo_del_valor_umbral
- Ming-Kuei-Hu, 1962. "Visual Pattern Recognition by Moment Invariants". Ire Transactions on Information Theory, pp 179-187.
- Rafael C. González, Richard E. Woods, 2002. Digital Image Processing (second edition). Ed. Prentice Hall.
- Transformada de Hough. Wikipedia. Disponible en la URL: http://es.wikipedia.org/wiki/Transformada_de_Hough.
- Transformada de Hough. Wikipedia. Disponible en la URL: http://es.wikipedia.org/wiki/Transformada_de_Hough.

ESCUELA SUPERIOR

DE INGENIERÍA

CIVIL E INDUSTRIAL

TITULACIÓN: Grado en Ingeniería Electrónica Industrial y Automática

Anexo I: Código Fuente

Trabajo Fin de Grado

TÍTULO

SISTEMA DE DETECCIÓN Y ALMACENAMIENTO DE VARIABLES VISUALIZADAS POR UN DISPOSITIVO MEDIANTE PANTALLA O DISPLAY A TRAVÉS DE UN SISTEMA DE VISIÓN DE BAJO COSTO.

AUTOR

Javier Hernández Afonso

TUTOR

Santiago Torres Álvarez

Índice:

A. SISTEMA OCR PRINCIPAL

A.1 Constructor

- *camara.m*

```
> function cam=camara(coor)
>
> cam.coor=coor;
> cam.modo=1;
> cam.cifras=Nan;
> cam.decimales=Nan;
>
> timer_OCR=timer('TimerFcn', 'Variable=reconocer(cam)',
'period',5);
> set(timer_OCR,'ExecutionMode', 'fixedRate');
> cam.t=timer_OCR;
>
> obj = videoinput('winvideo',1, 'YUY2_320x240');
> set(obj, 'ReturnedColorSpace', 'RGB');
> cam.obj=obj
> start(obj);
>
> end
```

A.2 Métodos

- *Reconocer.m*

```
%------------------------BLOQUE PRINCIPAL--------------------------
function [Valor, bw]=Reconocer(cam);



if Modo==1

%ANALISIS MÚLTIPLE-----------------------------------------


[bw puntos]=ObtenerImagen2(coor, Modo, obj);

%Eliminamos los posibles pixeles no conectados en la segmentación.
 se = strel('square',2);
 bw = imerode(bw,se);

%Obtenemos el número de cifras y su peso para el procesamiento multiple
[P D N in]=Pesos(bw);
```

```
props=regionprops(bw, 'boundingbox');
Valor=0;

for i=1:N
    if D==1
        if i~=in
            bwN= opermTest(bw, props, i); %Procesamiento morfológico

            E=Euler(bwN);                 %Extracción de características
            Mom=Momentos(bwN);

            Num=clasif(E,Mom, bwN);       %Clasificación
            Valor=Valor+Num*P;
            P=P/10;
        else
            Valor=Valor;
        end
    else
        bwN= opermTest(bw, props, i);     %Procesamiento morfológico

        E=Euler(bwN);                     %Extracción de características
        Mom=Momentos(bwN);

        Num=clasif(E,Mom, bwN);           %Clasificación

        Valor=Valor+Num*P;
        P=P/10;
    end
end

else %ANALISIS INDIVIDUAL (UN PROCESO PARA CADA CIFRA)
       Valor=0;
       P=10;
       Cifras=2;


    for i=1:Cifras

        [bw]=ObtenerImagen2(coor(:,:,i), Modo, obj);

        se = strel('square',2);
        bw = imerode(bw,se);
        props=regionprops(bw, 'boundingbox');

        bwN= opermTest(bw, props, 1);          %Procesamiento
morfológico

        E=Euler(bwN);                          %Extracción
características
        Mom=Momentos(bwN);

        Num(i)=clasif(E,Mom, bwN);             %Clasificación
```

```
        Valor=Valor+Num(i)*P;
        P=P/10;

    end
end
t=toc
end
```

- *Test.m*

```
%---------------------ENTRENAMIENTO MANUAL-------------------------
%Se recibirá una imagen de 5 muestras por fuente. Se calcularán los 10
%momentos centralizados de orden 3 y se almacenarán manualmente en los
%vectores de entrenamiento, que se pasarán a la máquina de clasificación.

function [trainingUNO, groupUNO, trainingCERO, groupCERO]= test(im, cam)


umb=graythresh(im);
bw=im2bw(im, umb);
bw=not(bw);


[L N]=bwlabel(bw);
props=regionprops(bw, 'boundingbox');

for i=1:11
    Ind(i)=1;
end

%Obtención de los momentos de cada región de muestra
for i=1:N

    bwC= opermTest(bw, props, i)
    imshow(bwC);
    N=input('¿Qué número es este? ');
    close all
    MhuC=sevenHU(bwC);

    if N==0
        training0(Ind(1), :)=MhuC;
        Ind(1)=Ind(1)+1;
    end

    if N==1
        training1(Ind(2), :)=MhuC;
        Ind(2)=Ind(2)+1;
```

```
    end

    if N==2
        training2(Ind(3), :)=MhuC;
        Ind(3)=Ind(3)+1;
    end

    if N==3
        training3(Ind(4), :)=MhuC;
        Ind(4)=Ind(4)+1;
    end

    if N==4
        training4(Ind(5), :)=MhuC;
        Ind(5)=Ind(1)+1;
    end

    if N==5
        training5(Ind(6), :)=MhuC;
        Ind(6)=Ind(6)+1;
    end

    if N==6
        training6(Ind(7), :)=MhuC;
        Ind(7)=Ind(7)+1;
    end

    if N==7
        training7(Ind(8), :)=MhuC;
        Ind(8)=Ind(8)+1;
    end

    if N==8
        Ind(9)=Ind(9)+1;
    end

    if N==9
        training9(Ind(10),:)=MhuC;
        Ind(10)=Ind(10)+1;
    end

end

%Generamos los vectores de entrenamiento.

trainingUNO=[training1; training2; training3; training5; training7];
trainingCERO=[training0; training4; training6; training9];

%Generamos los vectores de grupo.

 groupCERO=[0*ones(size(training0,1),1);...
            4*ones(size(training4,1),1);...
            6*ones(size(training6,1),1);...
            9*ones(size(training9,1),1)];
```

```
 groupUNO=[1*ones(size(training1,1),1);...
           2*ones(size(training2,1),1);...
           3*ones(size(training3,1),1);...
           5*ones(size(training5,1),1);...
           7*ones(size(training7,1),1)];

end
```

- *Temp.m*

```
%-----------------INICIALIZACIÓN DEL TEMPORIZADOR----------------------
function []=temp(cam)

start(cam);

end
```

- *Momentos.m*

```
%-----------------OBTENCIÓN DE MOMENTOS NORMALIZADOS--------------------

function [Mom] = Momentos(cam,R)

[m n]=size(R);

m00=0;
m10=0;
m01=0;

%OBTENCIÓN DE LOS MOMENTOS NECESARIOS PARA CENTRALIZAR (M00, M01 Y M10)

for x=1:m
    for y=1:n
        m00=m00+(x^0)*(y^0)*R(x,y);
        m10=m10+(x^1)*(y^0)*R(x,y);
        m01=m01+(x^0)*(y^1)*R(x,y);
```

```
    end
end

xm=m10/m00;
ym=m01/m00;

u00=0;
u10=0;
u01=0;
u11=0;
u20=0;
u02=0;
u21=0;
u12=0;
u22=0;
u30=0;
u03=0;
u31=0;
u13=0;
u40=0;
u04=0;

%OBTENCIÓN DE LOS MOMENTOS CENTRALIZADOS

for x=1:m
    for y=1:n

        u00=u00+((x-xm)^0)*((y-ym)^0)*R(x,y);
        u10=u10+((x-xm)^1)*((y-ym)^0)*R(x,y);
        u01=u01+((x-xm)^0)*((y-ym)^1)*R(x,y);
        u11=u11+((x-xm)^1)*((y-ym)^1)*R(x,y);
        u20=u20+((x-xm)^2)*((y-ym)^0)*R(x,y);
        u02=u02+((x-xm)^0)*((y-ym)^2)*R(x,y);
        u21=u21+((x-xm)^2)*((y-ym)^1)*R(x,y);
        u12=u12+((x-xm)^1)*((y-ym)^2)*R(x,y);
        u22=u22+((x-xm)^2)*((y-ym)^2)*R(x,y);
        u30=u30+((x-xm)^3)*((y-ym)^0)*R(x,y);
        u03=u03+((x-xm)^0)*((y-ym)^3)*R(x,y);
        u31=u31+((x-xm)^3)*((y-ym)^1)*R(x,y);
        u13=u13+((x-xm)^1)*((y-ym)^3)*R(x,y);
        u40=u40+((x-xm)^4)*((y-ym)^0)*R(x,y);
        u04=u04+((x-xm)^0)*((y-ym)^4)*R(x,y);

    end
end


%MOMENTOS CENTRALES NORMALIZADOS (invarianza ante escalados)
y15=1.5;
y2=2;
y25=2.5;
%Momentos orden 0:
n00=1;
%Momentos orden 1:
```

```
n10=u10/(u00^y15);
n01=u01/(u00^y15);
%Momentos orden 2:

n11=u11/(u00^y2);
n20=u20/(u00^y2);
n02=u02/(u00^y2);

%Momentos orden 3:

n21=u21/(u00^y25);
n12=u12/(u00^y25);
n30=u30/(u00^y25);
n03=u03/(u00^y25);


Mom= [n00 n10 n01 n11 n20 n02 n21 n12 n30 n03];

end
```

- *omermTest.m*

```
%-----------------PROCESAMIENTO MORFOLÓGICO-------------------------
%En el siguiente código se pretende normalizar las variables (números) a
%reconocer, fijándolos en una rejilla 33x33 tras su recorte, y
%efectuandoles un proceso de adelgazamiento.

function [bwOUT]= opermTest(cam, bw, props, i)

A=props(i).BoundingBox;

bw=imcrop(bw, A);
bw=imresize(bw, [33, 33]);

se = strel('square', 1);
bw = imopen(bw,se);

bwOUT = bwmorph(bw,'thin', Inf);

end
```

- *Euler.m*

```
%---------------------NÚMERO DE EULER-------------------------------
%Obtenemos el número de euler de la region de la imagen binaria entrante

function E  = Euler(cam, bw)

props=regionprops(bw, 'EulerNumber');
E=props.EulerNumber;

end
```

- *Pesos.m*

```
%----------------------RECONOCIMIENTO DE CIFRAS-------------------------

%Con éste código se obtienen el número de cifras de la variable a
%reconocer, así como el peso de la primera cifra y la existencia de
%decimales


function [P D N in]=Pesos(cam,bw)

[L, N]=bwlabel(bw);
R=regionprops(bw, 'Area'); %Se analizará la existencia de decimal
                           %estudiando la relación de las areas de la
                           %región mayor respecto a la región menor
A=0;

    for i=1:N              %Obtención del vector de areas
        A(i)=R(i).Area;
    end

M=max(A);
[m, in]=min(A);

    if N>2&&M>(10*m)      %Relación de las areas mayor y menor
        D=1;
    else
        D=0;
    end

    if D==1               %Obtención del peso de la primera cifra, en
                          %función de la existencia de decimal.
        P=10^(in-2);
    else
        P=10^(N-1);
    end

end
```

```
%---------------------------CLASIFICACIÓN-------------------------
%El siguiente código clasifica la región entrante en una de las 10 clases
%posibles, en función de su número de Euler y de los 10 momentos
centrales
%de orden 3 propios. Se utilizará un clasificador tipo knn (k vecinos más
%próximos. La máquina habrá sido entrenada previamente con 5 muestras de
%distinta fuente por clase.

function [Num]=clasif(cam,E,Mom, bw)

load traininv

 if E==-1
         Num=8;

 elseif E==0
         Num=knnclassify(double(Mom), double(training0), group0);

 else
         Num=knnclassify(double(Mom), double(training1), group1);

 end


 end
```

```
%-----------CAMBIO DE CONFIGURACIÓN: SELECCIÓN DE MODO Y CALIBRACIÓN-----

function cam=cambiar_configuración(cam)

      > Modo=input('Selección de modo: Reconocimiento...
      > múltiple(1),Reconocimiento > individual(0):');


        if Modo==1
            im=getsnapshot(obj);
            [coor, Fconv] = cali(im);
             %Ir al timer
             [Num, bw]=Reconocer(Modo, coor, obj,  1, 1);
        else
             Cifras=input('¿Cuántas cifras posee la variable a
reconocer?');
             Decimales=input('¿Cuántas de ellas son decimales?');
             Peso=10^(Cifras-Decimales-1)
             im=getsnapshot(obj);

             for i=1:Cifras
                [coor(:,:,i),Fconv] = cali(im);
             end

                %Ir al timer
             Num=Reconocer(Modo, coor, obj, Cifras, Peso);
        end
end
```

- *ObtenerImagen.m*

```
%--------------------SEPARACIÓN DE DÍGITOS SOLAPADOS---------------------
-
function [bwF puntos]=ObtenerImagen(cam)


K=0;

if Modo==0;              %Definimos un umbral de coincidencia que
                         %establezca que las regiones son semegantes al
                         %estudiar la correlación entre ambas.
    Coincidencia=0.90;
else
    Coincidencia=0.95;
end

for i=1:5
    puntos(i)=0;
    fotos(:,:,:,i)=getsnapshot(obj);
end



for i=1:5

     K=K+1;

     im = cort(fotos(:,:,:,i),coor);
     umb=graythresh(im);
     bw(:,:,K)=im2bw(im, umb);


end

%Estudio de la correlación de las imagenes entre sí

      for i=1:K
          corP=correlAbs(bw(:,:,i),bw(:,:,i));

          for j=1:K
          cor=correlAbs(bw(:,:,i),bw(:,:,j));

            if cor/corP>Coincidencia;
                puntos(i)=puntos(i)+1;
```

```
            else
                puntos(i)=puntos(i);
            end

         end
      end

%Suma de las imagenes

   for i=1:K
       if i==1
           bwSum=bw(:,:,1)/K;
       else
           bwSum=bwSum+bw(:,:,i)/K;
       end

   end


%Separación

MaxRep=max(puntos);
cant=find(puntos==MaxRep);
cant=size(cant);
cant=cant(2);

    if cant>MaxRep
        bwF=bw(:,:,1);
    else
        umb=MaxRep/K;
        bwF=bwSum>0.95*umb;
    end
  end




%----------------------------CORRELACIÓN---------------------------------
%El código obtiene la correlación absoluta entre dos imagenes binarias

function c=correlAbs(cam, bw1, bw2)

[n, m]=size(bw1);
c=sum(reshape(bw1.*bw2, n*m, 1));

end
```

- *correlAbs.m*

```
%--------------------------CORRELACIÓN-------------------------------
%El código obtiene la correlación absoluta entre dos imagenes binarias

function c=correlAbs(bw1, bw2)

[n, m]=size(bw1);
c=sum(reshape(bw1.*bw2, n*m, 1));

end
```

- *Cali.m*

```
%-----------------------------CALIBRADO-------------------------------


function coor = cali(cam, img)

    %
    % Calculo de todo el ejercicio
    %
    % Parámetros de entrada:
    %   obj : Objeto de entrada de video
    %
    %
    % Parámetros de salida:
    %   Pf : Ptos de la posicion de la imgen detectada
    %
    % Creada el 13/11/07, por Santiago&Waldo.
    %



%SINCRONISMO DE ANGULOS Y DISTANCIAS
    imshow(img)

%    axes2_CreateFcn(hObject, eventdata, handles)

    coor = ginput(4);
    d = sqrt( (coor(2,1)-coor(1,1))^2 + (coor(2,2)-coor(1,2))^2 );
    Fconv = 50/d;
```

- *Cort.m*

```
%----------------------------CORTE DE IMÁGENES--------------------------

function [IMGcort] = cort(img,coor)

    %
    % Calculo de todo el ejercicio
    %
    % Parámetros de entrada:
    %   obj : Objeto de entrada de video
    %
    %
    % Parámetros de salida:
    %   Pf : Ptos de la posicion de la imgen detectada
    %
    % Creada el 13/11/07, por Santiago&Waldo.
    %

    XY = [coor(1,1),coor(1,2)];
    tmp = coor(:,1); coor(:,1) = coor(:,2); coor(:,2) = tmp;
    Ymin =  floor( max(coor(1,1),coor(2,1)) );
    Ymax = ceil( min(coor(4,1),coor(3,1)) );
    Xmin =  floor( max(coor(1,2),coor(4,2)) );
    Xmax = ceil( min(coor(2,2),coor(3,2)) );
    %imshow(img)
    hold on
    plot([Xmin Xmax Xmax Xmin Xmin], [Ymin Ymin Ymax Ymax Ymin]);
    hold off
    IMGcort = img(Ymin:Ymax, Xmin:Xmax,:);
```

B. SISTEMA OCR PARA 7 SEGMENTOS

- *DeteccionHough.m*

```
%------------------ DETECCION 7 SEGEMENTOS-----------------------
 %El siguiente script recibe como parámetros de entrada una imagen de un
 %numero de 7 segementos, extrae sus lineas mediante la transformada de
 %hough y lo lo clasifica.
function Numero=DeteccionHough(imagen)

 imageng=rgb2gray(imagen);

 BWP=process(imageng);
 BW=edge(BWP,'sobel'); %marcamos bordes
 figure, imtool(BW);

 [H, THETA, WHO]= HOUGH(BW); %Aplicamos la transformada de Hough sobre la
imagen
 imtool(H, [])
 s=size(H)/50;
 peaks=houghpeaks(H, 7 ,  'threshold',63, 'NHoodSize', [99 99]);
 pause
 %Muestra los 7 valores (máximo) de maxima intensidad del diagrama. Se
 %aplica además una máscara 99x99 que anule los picos vecinos a los picos
 %máximos dentro del rango establecido



 lines=houghlines(BW, THETA, WHO, peaks); % Devolvera una estructura.
Serán
                                          % tantas cajas como "peaks"
                                          % hayamos pasado

 a=size(lines);
 numlineas=a(2);


%Pintamos las lineas detectadas sobre la imagen
 for i=1:numlineas


    imageng(lines(i).point1(2):lines(i).point2(2),
lines(i).point1(1):lines(i).point2(1))=0;


 end



H=0;
L=0;
```

```
V=0;


%Clasificamos las lineas en horizontales, verticales y largas
j=1;
for i=1:numlineas



    if lines(i).point1(1)==lines(i).point2(1)

        tamver(j)=abs(lines(i).point1(2)-lines(i).point2(2));
        j=j+1;
    else

        H=H+1;
        tamref=abs(lines(i).point1(1)-lines(i).point2(1));
    end
end

if numlineas>1
    a=size(tamver);
    for i=1:a(2)
        if tamver(i)>1.5*tamref
            L=L+1;
        else
            V=V+1;
        end
    end
else
    numlineas=numlineas;

end

% Clasificación de las regiones
if L==2&&V==0&&H==3
    Numero=8;

elseif L==1&&V==0&&H==1
    Numero=7;

elseif L==1&&V==0&&H==0
    Numero=1;

elseif L==1&&V==0&&H==3
    Numero=3;

elseif L==0&&V==2&&H==3  %Para el caso del 2 y el 5, estudiamos la
posición
                         %relativa de la coordenada x del primer punto
                         %final con la coordenada x del centroide
    Numero=2;
    EP=bwmorph(BWP, 'endpoints');
    [A B]=find(EP==1);
```

```
    C=regionprops(BWP, 'centroid');
    C=C.Centroid(2);

    if B(1)<C
        Numero=2;
    else
        Numero=5;
    end
elseif L==1&&V==1&&H==1
    Numero=4;

elseif L==2&&V==0&&H==2
    Numero=0;

elseif L==1&&V==1&&H==2  %Para el caso del 6 y el 9, estudiamos la
posición
                         %relativa de la coordenada y del punto
                         %final con la coordenada y del centroide

    EP=bwmorph(BWP, 'endpoints');
    [A B]=find(EP==1);

    C=regionprops(BWP, 'centroid');
    C=C.Centroid(1);

    if B<C
        Numero=6;
    else
        Numero=9;
    end

end

end
```

- *Process.m*

```
%--------------- PROCESAMIENTO 7 SEGMENTOS----------------------

%Función que toma una imagen de un numero 7-seg y devuelve
%el mismo numero como única región conectada

function [imagen1]= process(imagen7)

 umbral=graythresh(imagen7);
 BW=im2bw(imagen7, umbral); %Binarizamos la imagen

 BW=not(im2bw(imagen7, umbral));

 s=1;

 while true  %Dilatamos la imagen hasta obtener un elemento conectado
 s=s+1;
 se=strel('square', s);
 BWx=imdilate(BW, se);
 [L,NUM] = BWLABEL(BW); %Obtención de elementos conectados

 if NUM==1
      BW=BWx;
      break
 else
      BW=BWx;
 end

 end

 s=1;


  while true %Erosionamos la imagen hasta antes de tener más de un
elemento conectado
  s=s+1;
  se=strel('square', s );

  BWx=imerode(BW, se);

  [L,NUM] = BWLABEL(BWx); %Obtención de elementos conectados

  if NUM==1
      BW=BWx;
  else
      break;
  end

  end
   BW2 = bwmorph(BW,'thin',Inf);

   imagen1=BW2;
```

```
end
```