



**Universidad Rey Juan Carlos**

**Máster Oficial en Visión Artificial**

**Curso Académico 2012/2013**

**Detección y reconocimiento de texto**

**Trabajo de Fin de Máster**

**Autor:** Roberto Valle Fernández

**Tutor:** José Miguel Buenaposada Biencinto

8 de junio de 2013



# Agradecimientos

En el plano personal, me gustaría dar las gracias a las siguientes personas por el apoyo ofrecido durante el desarrollo del trabajo fin de máster:

- En primer lugar, quiero agradecer a mi tutor José Miguel Buenaposada Biencinto la oportunidad brindada para realizar de nuevo el proyecto a su lado. Porque siempre ha tenido una respuesta amable a todas las preguntas que surgían, y sobre todo por la confianza depositada en mí desde el principio. Por estos 4 años de trabajo contigo muchas gracias.
- Dar las gracias a todos mis compañeros de universidad, con los cuales he compartido momentos increíbles durante los últimos 7 años de facultad. Con un recuerdo especial para Sergio, David, París, Esther, Moisés y Miguel.
- Gracias de corazón a mis amigos de toda la vida, por la felicidad de cada día pasado, por la constancia y por estar siempre ahí cuando era necesario. Aunque sea evidente, sois motivo de orgullo para mí y por eso os llevo en mi corazón.
- Finalmente, quería dedicarle el título de ingeniero superior en informática, así como el máster en visión artificial a mi familia, por el ánimo que me ha dado. En particular, esto va para mi padre, mi madre y mi hermano, por haberme educado así y por creer en mis posibilidades en todo momento. Como primer miembro de la familia que llega a la universidad quería reservar este último hueco de la memoria para vosotros.

Gracias.

# Resumen

El presente trabajo fin de máster detalla el interés que suscita procesar imágenes para proceder a la lectura del texto natural sin restricciones. De hecho, identificar con acierto el posible vocabulario visible en la escena, resulta de gran utilidad para multitud de registros del acontecer cotidiano: digitalizar el contenido de la imagen (pasaporte, matrícula, ...), lectura del texto adaptada a personas invidentes, traducción automática del texto a otros idiomas, soporte a la navegación de vehículos, etc.

La propuesta del trabajo pretende dar una solución válida al problema de la detección y el reconocimiento de texto natural. Para ello, primero es importante precisar la posición exacta de cada carácter visible en la escena por separado. Acto seguido, conviene estimar la sucesión de caracteres de interés, para detectar el vocabulario por palabras. Por último, la idea será proceder a reconocer el contenido de cada palabra, tras clasificar cada carácter de manera individual.

Como resultado, esta aproximación a estudiar pretende detectar el texto de una escena que introduce la imagen o vídeo de entrada, localizando cada carácter potencial mediante la técnica de umbralización adaptativa. Así, el algoritmo debe prescindir de los contornos extraídos que no sean dígitos o letras, mediante un clasificador SVM. Por otra parte, cabe destacar el uso de la técnica RANSAC a la hora de estimar la secuencia de contornos que forma cada palabra (caracteres alineados). No obstante, la escasa tasa del 52 % de acierto al localizar el léxico de la colección de imágenes *Street View Text*, refleja la pobre calidad del texto natural de la base de datos.

Durante el desarrollo de la aplicación C++, interesa la construcción de la base de datos sintética, para caracteres del alfabeto latino (0-9, A-Z, a-z). Así, para reconocer con éxito el contenido de cada palabra, el objetivo es identificar la clase de cada carácter por separado, mediante el clasificador bayesiano óptimo, al reducir la dimensión inicial del problema con PCA-LDA (tasa del 99 % de acierto), para presentar al usuario por pantalla el vocabulario visible sobre la escena.

# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Posibles problemas . . . . .	2
1.2. Objetivos . . . . .	3
<b>2. Estado del arte</b>	<b>5</b>
2.1. Descripción de los trabajos anteriores . . . . .	5
2.1.1. End-to-End Scene Text Recognition [1] . . . . .	5
2.1.2. Stroke Width Transform [2] . . . . .	6
2.2. Análisis de las bases de datos . . . . .	8
2.2.1. ICDAR . . . . .	9
2.2.2. Natural Street Pictures . . . . .	9
2.2.3. Chars74K . . . . .	10
2.2.4. Street View Text . . . . .	11
<b>3. Algoritmos propuestos</b>	<b>14</b>
3.1. Localización del texto . . . . .	14
3.1.1. Segmentación de contornos . . . . .	14
3.1.2. División de caracteres unidos . . . . .	16
3.1.3. Clasificador binario de texto . . . . .	16
3.2. Reconocimiento de líneas . . . . .	19
3.2.1. Estimar líneas candidatas . . . . .	19
3.2.2. Clasificador de caracteres . . . . .	21
<b>4. Resultados experimentales</b>	<b>25</b>
4.1. Detección de caracteres . . . . .	25
4.1.1. Construcción de la base de datos . . . . .	25
4.1.2. Experimentos de clasificación . . . . .	26
4.2. Reconocimiento de palabras . . . . .	31
4.2.1. Construcción de la base de datos . . . . .	31
4.2.2. Experimentos de clasificación . . . . .	33
<b>5. Descripción informática</b>	<b>39</b>
5.1. Metodología y plan de trabajo . . . . .	39
5.2. Captura de requisitos . . . . .	40
5.2.1. Requisitos funcionales . . . . .	41
5.2.2. Requisitos no funcionales . . . . .	41
5.3. Análisis y diseño . . . . .	42
5.4. Implementación . . . . .	47

<b>6. Conclusiones</b>	<b>50</b>
6.1. Análisis de los resultados . . . . .	50
6.2. Trabajos futuros . . . . .	52

# Índice de figuras

1.1.	Estimar el grado de dificultad asociado al texto natural a recuperar. . . . .	1
1.2.	Traducir al instante el vocabulario extraído de la imagen. . . . .	2
1.3.	Imágenes naturales adecuadas para el OCR por el contraste con el fondo. . .	3
2.1.	Esquema que detalla el proceso de reconocimiento de palabras. . . . .	5
2.2.	Extracción del ancho del trazo y localización del texto. . . . .	7
2.3.	Calcular el ancho del trazo que delimita el segmento $[p, q]$ . . . . .	7
2.4.	Detección y reconocimiento de texto sobre imágenes del ICDAR 2003. . . . .	9
2.5.	Sucesión de imágenes naturales para la colección NSP. . . . .	10
2.6.	Reconocer los caracteres asociados a la base de datos Chars74K. . . . .	10
2.7.	Extraer palabras de la colección de imágenes Street View Text. . . . .	11
3.1.	Esquema que detalla el funcionamiento de la aplicación propuesta. . . . .	14
3.2.	Contornos a extraer mediante la umbralización adaptativa propuesta. . . . .	15
3.3.	Dividir los contornos de ancho excesivo para evitar posible texto unido. . .	16
3.4.	Reducir el número de contornos aplicando el clasificador de texto. . . . .	17
3.5.	Construcción del hiperplano mediante los vectores soporte $z_i$ . . . . .	17
3.6.	Transformar el espacio de entrada a otro de alta dimensionalidad. . . . .	18
3.7.	Establecer las líneas candidatas a partir de los caracteres a reconocer. . . .	19
3.8.	Separar grupos de contornos distantes asociados a la misma línea. . . . .	20
3.9.	Reconocimiento de caracteres sobre las líneas candidatas extraídas. . . . .	21
3.10.	Clasificar los caracteres normalizados de la línea candidata. . . . .	21
4.1.	Diferentes contornos de interés para entrenar el clasificador de texto. . . .	25
4.2.	Determinar el <i>kernel</i> y $C$ adecuado para recuperar el texto de la imagen. .	26
4.3.	Hipótesis de distribución gaussiana que minimiza la probabilidad de error.	29
4.4.	Determinar el $\sigma$ óptimo asociado al kernel RBF. . . . .	30
4.5.	Procesar los caracteres sintéticos desde FreeType. . . . .	31
4.6.	Dígitos y letras sintéticos para entrenar el clasificador de caracteres. . . .	33
4.7.	Sucesión de caracteres de altura similar en línea horizontal. . . . .	34
4.8.	Rendimiento del clasificador bayesiano al reconocer los caracteres sintéticos.	35
4.9.	Matriz de distancias asociada al clasificador de caracteres. . . . .	36
4.10.	Reconocimiento de texto natural aplicado a la sucesión de caracteres. . . .	37
5.1.	Diagrama de clases de diseño que permite construir cada clasificador. . . .	42
5.2.	Perspectiva UML asociada a la detección y reconocimiento de texto. . . . .	43
5.3.	Diagrama de clases de diseño que presenta la interfaz al usuario. . . . .	44
5.4.	Diagrama de secuencia UML que detalla la ejecución básica del programa.	45
5.5.	Diagrama de clases de diseño que presenta el texto en la interfaz. . . . .	46

5.6. Estructura jerárquica que gestiona el contenido del producto software. . . . .	47
6.1. Imágenes naturales inadecuadas para evaluar la precisión del sistema. . . . .	51
6.2. Distintas aplicaciones a considerar al hacer uso de la aplicación. . . . .	52



# Capítulo 1

## Introducción

En la actualidad, prevalece el auge de dispositivos móviles, consolas o tablets de última generación, dotados con una infraestructura tecnológica que permite soportar el empleo de cámaras fotográficas, vídeo, acceso a redes, GPS, etc. Como resultado, es habitual el uso de estas cámaras digitales sobre diferentes entornos donde pueda aparecer texto de interés.

Con el tiempo, el creciente interés que suscita digitalizar ese posible texto de la imagen (conservar el vocabulario presente en la escena natural), aumenta la necesidad de extraer automáticamente su contenido, usando el OCR adecuado (*Optical Character Recognition*). Sin embargo, la naturaleza del problema y las diversas vertientes que establece, hacen que sea un proceso en continua investigación.

La capacidad para detectar e identificar con éxito el vocabulario de la escena, depende de la calidad del texto, la resolución de esa imagen y las características del entorno natural (iluminación, posibles occlusiones, patrones repetitivos de aspecto similar al carácter, . . .). No obstante, la claridad del texto, a priori inferior para escenas sin restricciones y de baja resolución, tipo *Google Maps*, determina el rendimiento de la aplicación. En la Figura 1.1 se presenta la apariencia del léxico a recuperar de estas imágenes, con respecto al análisis de documentos escaneados o códigos CAPTCHA.



Figura 1.1: Estimar el grado de dificultad asociado al texto natural a recuperar.

Asimismo, al construir cierto OCR para recuperar con éxito el texto natural, es factible procesar la escena al completo para localizar cada carácter visible y reconocer su contenido de manera individual. A continuación, cabe especificar un listado con posibles aplicaciones a considerar de interés:

- *Gestión de información.* Digitalizar el texto de la escena, con el objetivo de procesar el léxico extraído desde el sistema, gestionando ese contenido de manera conveniente (conservar la información, reconocer documentos de identidad, lectura de matrículas de vehículos, corrección gramatical, . . .).

- *Lectura de texto adaptada a personas ciegas.* Evitar el uso del alfabeto braille para personas invidentes o con cierta discapacidad visual. La aplicación podría ayudar a recuperar el vocabulario que recoge la cámara para proceder a convertir su contenido en el audio que escucha el usuario.
- *Soporte a la navegación.* Asistencia a la conducción autónoma de vehículos o robots móviles tras estimar la trayectoria a seguir, en función de las señales de circulación que podría reconocer el automóvil durante el recorrido.
- *Traducción instantánea del texto.* Procesar el vocabulario de la escena para traducir el texto extraído a cualquier otro idioma. La Figura 1.3 ilustra el funcionamiento de la aplicación, tras traducir el texto y superponer el resultado sobre la misma imagen de entrada (realidad aumentada).



Figura 1.2: Traducir al instante el vocabulario extraído de la imagen.

## 1.1. Posibles problemas

No obstante, conviene considerar la cantidad de problemas que implica recuperar con éxito cada dígito o letra de cada escena, a partir de las características de la propia imagen (legibilidad del texto, nitidez de los caracteres, contraste con el fondo, etc.). A priori, basta con evitar la presencia de ciertas condiciones adversas durante la adquisición de la imagen, para mejorar la tasa de acierto del OCR sobre texto natural:

- *Presencia de occlusiones.* Evitar la presencia de objetos cerca de la escena que puedan ocultar parte del texto, con el fin de identificar el vocabulario al completo. De hecho, la posible pérdida parcial de caracteres de la misma palabra, afecta a la localización (mínimo de caracteres exigido por palabra).
- *Iluminación no uniforme.* Controlar la legibilidad correcta del texto, producto de la iluminación de la escena y el contraste suficiente con el fondo. La ausencia de brillos deslumbrantes cerca del texto facilitan la lectura cómoda del vocabulario.
- *Desenfoque por movimiento.* Como la posición del texto en la escena es desconocida, es necesario cuidar el enfoque y evitar la presencia de borrones durante la adquisición a causa del movimiento de la cámara.
- *Dimensión del texto.* La percepción adecuada de los caracteres depende de la distancia que separa al propio texto de la cámara digital. Por norma general, la dificultad a la hora de reconocer el texto es mayor al reducir el tamaño de los caracteres.

- *Claridad de la tipografía.* La amplia variedad de tipografías presentes para el texto natural, limita la precisión del OCR. Potenciar el empleo de tipografías más legibles (trazos simples de ancho notable) podría ayudar a recuperar con éxito los dígitos y letras del alfabeto latino.

Así, la idea del trabajo es recuperar todo el texto natural de escenas con características similares a las que presenta la Figura 1.3, donde la legibilidad de los caracteres es correcta.



Figura 1.3: Imágenes naturales adecuadas para el OCR por el contraste con el fondo.

## 1.2. Objetivos

El objetivo del trabajo fin de máster es proceder a estudiar el problema de la detección y reconocimiento del texto visible desde cada escena natural, a partir de la imagen o vídeo de entrada. A continuación, cabe presentar la lista de objetivos parciales a alcanzar, para poder lograr el objetivo final (construir el OCR en imágenes naturales):

- *Construir la base de datos.* Generar las distintas bases de datos necesarias para realizar los experimentos que establecen el rendimiento de cada clasificador propuesto. Destaca el clasificador de texto que detecta cada dígito o letra de la imagen, así como el clasificador de caracteres que identifica el contenido de cada carácter extraído.
- *Localizar la posición exacta del texto.* Determinar la ubicación de los caracteres de la escena natural, a ser posible en tiempo real. Estudiar al inicio la opción de recorrer la imagen para encontrar dígitos o letras a distinta escala, o bien procesar la escena para detectar el texto en función de las características típicas de los caracteres.
- *Reconocer el contenido del texto.* Desarrollar el OCR para clasificar correctamente cada carácter extraído, como una de las 62 clases del alfabeto latino (0-9, A-Z, a-z). Proporcionar el resultado por palabras (sucesión de caracteres alineados).
- *Implementar el sistema.* Trasladar los algoritmos que evalúa el trabajo fin de máster, al lenguaje de programación C++ y estudiar el rendimiento de la propuesta.



# Capítulo 2

## Estado del arte

El estado del arte hace referencia a la investigación de carácter documental que aporta conocimiento previo relacionado con la temática a tratar. A lo largo de este Capítulo 2, el objetivo es recopilar el estado actual de la investigación, para presentar las bases asociadas a la detección y el reconocimiento de texto natural y poner en contexto el trabajo realizado.

### 2.1. Descripción de los trabajos anteriores

Las características de este problema y las diversas vertientes que presenta, hacen que sea un proceso en continua investigación. Como punto de partida, para empezar a formular el problema, es interesante la lectura crítica y analítica de los artículos más significativos realizados sobre este tema.

#### 2.1.1. End-to-End Scene Text Recognition [1]

Este trabajo aborda el estudio del reconocimiento de texto, desde la detección del área ocupada por el propio texto dentro de la imagen, hasta su extracción y clasificación en cadenas de texto. Para determinar la posición de los caracteres en la escena, los autores del artículo proponen un algoritmo de clasificación basado en la forma y la textura de los caracteres (gradiente de los bordes y contraste adecuado con el fondo).

Respecto a la formación de palabras a partir de los caracteres previamente detectados, será imprescindible proporcionar un listado  $\mathcal{L}$  con el posible léxico a reconocer, donde se incluya de antemano el vocabulario de interés, junto a otras palabras con sentido para la escena a procesar. La Figura 2.1 detalla el proceso de reconocimiento de texto propuesto usando como léxico: PUFF, STUFF, FUN, MARKET, VILLAS, SMOKE, ...

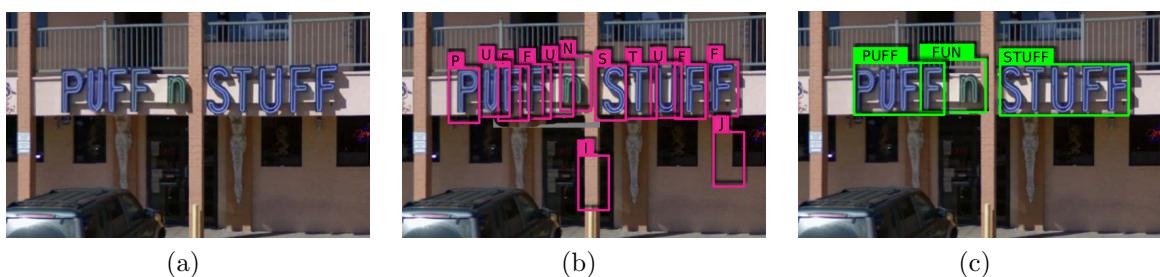


Figura 2.1: Esquema que detalla el proceso de reconocimiento de palabras.

- *Detección de caracteres.* En esta primera etapa el objetivo principal será localizar la posición exacta de los caracteres que aparecen en la escena que se está procesando. La idea propuesta por el artículo de K. Wang et al. [1] permite usar el algoritmo de búsqueda conocido como “ventana deslizante” para iterar sobre la imagen, con una ventana que se desliza a diferentes escalas sobre una determinada región de interés. A priori no se conoce la dimensión de los caracteres en la escena natural, por lo que la ventana deslizante debe ser capaz de cambiar de tamaño para tratar de clasificar todas las posiciones a diferentes escalas.

De esta manera, para detectar todos los caracteres presentes en la escena, mientras la ventana deslizante recorre la imagen entera, se debe aplicar un clasificador biclase (caracter / no caracter) de tal forma que, al ajustar la ventana sobre los caracteres de interés, la puntuación  $u(\ell, c)$  de la Ecuación 2.1 sea positiva. En este punto, aparece el clasificador *Random Ferns* presentado por M. Özuysal et al. [7], que determina la probabilidad  $\mathbf{p}$  de encontrar un carácter  $c$ , sobre una posición  $\ell$  concreta, tras hacer uso del vector de características  $x$  extraído de la subimagen, mediante el descriptor HOG (*Histograms of Oriented Gradients*).

$$u(\ell, c) = \log \left( \frac{\mathbf{p}(c|x)}{\mathbf{p}(c_{bg}|x)} \right) \quad (2.1)$$

A medida que la ventana de clasificación se desliza a diferentes escalas por la imagen es usual que se dispare más de una detección alrededor del objeto buscado. Así, para agrupar las ventanas correspondientes a detecciones solapadas, la idea es aplicar el algoritmo NMS (*Non-Maximal Suppression*).

- *Reconocimiento de palabras.* Para determinar qué posibles palabras encontrar en la imagen procesada, es imprescindible hacer uso del léxico  $\mathcal{L}$  que especifica todo el vocabulario que puede aparecer en la escena. De esta forma, sea  $w = (c_1, c_2, \dots, c_n)$  una palabra con  $n$  caracteres del léxico, el objetivo será encontrar la configuración óptima  $\mathcal{L}^* = (\ell_1^*, \ell_2^*, \dots, \ell_n^*)$  a partir de la puntuación  $u(\ell, c)$  asociada al conjunto de caracteres  $c_i$  y posiciones  $\ell_i$  definidos en la etapa anterior.

Finalmente, para eliminar posibles falsos positivos en la detección, la propuesta de los autores es entrenar un clasificador SVM que reciba ciertas características de los caracteres que forman la detección ( $\mathcal{L}^*$ ,  $\mu$  y  $\sigma$  con las puntuaciones de cada posición  $\ell_i$ , el número de caracteres que forman la palabra, la separación entre los caracteres en cuestión, etc). Respecto a la implementación del algoritmo desde MATLAB [16], el tiempo de ejecución medio que lleva procesar con éxito una imagen de 800 x 1200 píxeles, con el apoyo de un léxico de 50 palabras, es de aproximadamente 15 segundos.

### 2.1.2. Stroke Width Transform [2]

Desde el punto de vista computacional, recorrer las escenas al completo para localizar todo el texto posible a diferentes escalas, limita en gran parte el rendimiento del sistema. La propuesta de los autores de este artículo será recuperar el texto de interés en función de las propiedades asociadas a los caracteres típicos (regiones de píxeles de características similares) que recoja la imagen natural.

Para este trabajo en concreto, como característica discriminante a la hora de reconocer el texto, el algoritmo establecido trata de clasificar las regiones de la escena a partir de su anchura de trazo, de tal forma que al procesar contornos similares a los de la Figura 2.2, es bastante fácil distinguir los caracteres que mantienen su trazo regular.



Figura 2.2: Extracción del ancho del trazo y localización del texto.

- *Procesamiento basado en el ancho del trazo.* El objetivo será localizar la posición exacta de los caracteres que aparecen en la escena natural a partir del ancho del trazo que caracteriza la tipografía en cuestión. La técnica SWT (*Stroke Width Transform*) evalúa la vecindad del contorno asociado a cada píxel  $p$  de la imagen de entrada para sustituir la intensidad (nivel de gris) por el ancho del trazo asociado respectivamente.

En definitiva, la idea será recorrer cada píxel borde  $p$  e ir proyectando una recta  $d_p$  en la dirección del gradiente correspondiente. De tal manera que, cuando la recta  $r = p + k \cdot d_p, \forall k > 0$  respectiva, cruce con un píxel borde  $q$  con gradiente  $d_q$  en sentido contrario (píxel opuesto del contorno), será posible calcular la distancia  $\|p - q\|$  que determina el ancho del trazo  $w$  de interés. La Figura 2.3 presenta el algoritmo que calcula la amplitud del segmento  $[p, q]$ .

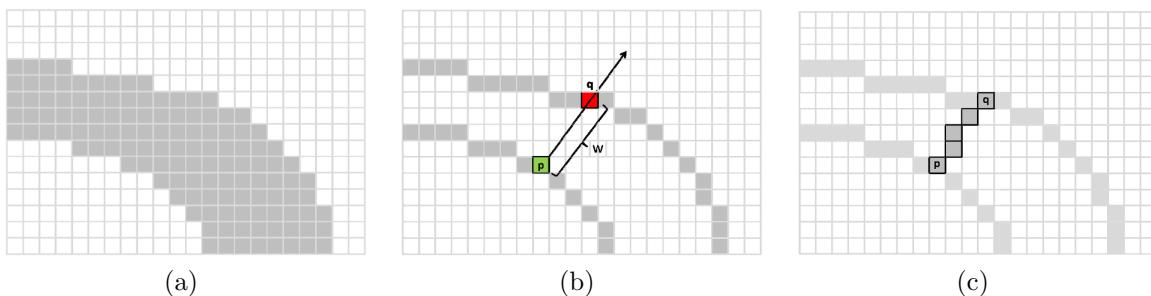


Figura 2.3: Calcular el ancho del trazo que delimita el segmento  $[p, q]$ .

Por el contrario, si la recta no encuentra ningún píxel borde  $q$  con gradiente en sentido contrario, el valor establecido será la mediana del ancho del trazo asignado a la vecindad del segmento  $[p, q]$ .

- *Agrupación de candidatos.* Tras terminar de procesar la escena con SWT, la imagen como resultado almacena en cada píxel la amplitud asociada al trazo en cuestión. Consecuentemente, el objetivo será agrupar píxeles vecinos con un ancho de trazo similar, para formar contornos con el aspecto de los caracteres. Así pues, prescindir de las regiones que tienen asociada una varianza importante dentro del propio trazo, implica reducir sustancialmente los contornos que determinan la presencia de texto en la escena natural.

Para tratar de eliminar el resto de contornos de trazo regular que no forman parte del texto a localizar, primeramente la idea es aplicar un filtrado geométrico para limitar la relación de aspecto asociada a cada carácter en cuestión. Llegados a este punto, es posible reconocer las líneas de texto en la escena, determinando la presencia de regiones teóricamente próximas entre sí con características similares (altura y ancho del trazo común en cada palabra).

Finalmente, para separar por palabras las cadenas de texto detectadas y proceder al reconocimiento de cada carácter de manera individual, la propuesta del artículo es hacer uso de una heurística capaz de estimar el umbral óptimo que diferencia entre la distancia media entre caracteres (*intra-word*) y la distancia entre palabras (*inter-word*) asociada a cada cadena de texto. Como resultado, el rendimiento mejora sustancialmente con respecto al obtenido al recorrer la imagen con la ventana deslizante a diferentes escalas. Según los autores, el algoritmo es capaz de detectar la presencia de texto en la escena con un tiempo de ejecución de hasta un frame por segundo aproximadamente.

## 2.2. Análisis de las bases de datos

Antes de evaluar la eficiencia y la eficacia de las técnicas propuestas desde los artículos más significativos relacionados con el tema, cabe destacar el nivel de dificultad que supone extraer de manera automática el texto natural de cada base de datos pública que sirve de referencia. Respecto a las medidas de evaluación del rendimiento, la puesta en común de los autores es establecer una relación de compromiso entre la precisión y el recall.

- *Precisión.* Determinar la proporción existente entre la cantidad de texto recuperado y reconocido con éxito (*true positives*), con respecto a la información total extraída (*true positives + false positives*). Aumentar la precisión implica reducir el riesgo de clasificar erróneamente contornos como caracteres que no son.
- *Recall.* Estimar la proporción existente entre la cantidad de texto natural reconocido con éxito (*true positives*), con respecto a la información total disponible en la escena (*true positives + false negatives*). Aumentar la sensibilidad implica reducir el riesgo de rechazar texto que no debería de pasar desapercibido.
- *$F_\beta$  - Measure.* Media armónica ponderada entre la precisión y el recall detallada en la Ecuación 2.2.

$$F_\beta = \frac{(1 + \beta^2) \cdot Precision \cdot Recall}{\beta^2 \cdot Precision + Recall} \quad (2.2)$$

Cuando el objetivo sea hacer una ponderación equivalente de precisión y recall basta con asignar  $\beta = 1$ . Teóricamente,  $\beta < 1$  busca darle mayor importancia a la precisión mientras que  $\beta > 1$  prioriza el recall.

Llegados a este momento, es interesante estudiar el rendimiento de las propuestas más significativas de la literatura en función de las bases de datos de referencia.

### 2.2.1. ICDAR

Para el ámbito de la localización y reconocimiento automático de texto en imágenes naturales, las bases de datos públicas más relevantes son las asociadas al congreso bienal de referencia ICDAR (*International Conference on Document Analysis and Recognition*). Celebrado desde 1991, el ICDAR presenta para cada edición los últimos avances técnicos logrados en la materia. No obstante, las imágenes detalladas en la Figura 2.4 forman parte de la colección propuesta para la competición del ICDAR 2003.

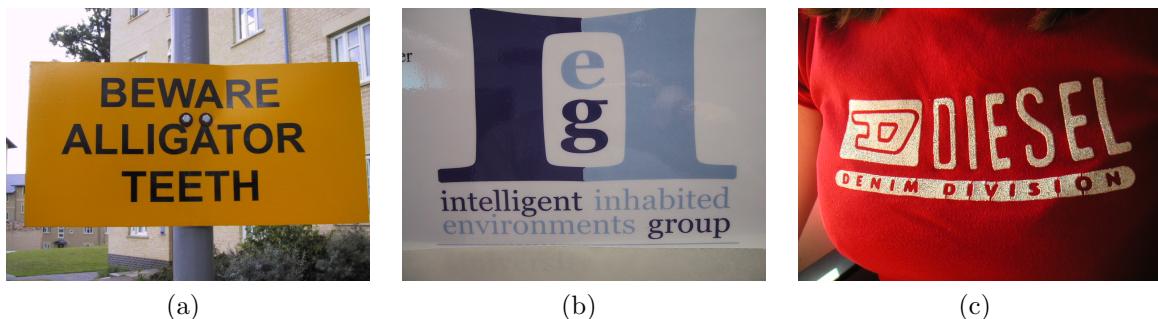


Figura 2.4: Detección y reconocimiento de texto sobre imágenes del ICDAR 2003.

Como resultado, el artículo de S. Lucas et al. [4] detalla el rendimiento de los algoritmos más significativos a la hora de localizar la posición exacta del vocabulario de cada escena natural de la colección del ICDAR. El análisis del Cuadro 2.1 refleja el nivel de dificultad que supone extraer cada palabra (sucesiones de contornos, pérdida de caracteres, etc).

Algoritmo	Precisión	Recall	F-Measure	Tiempo
SWT [2]	0.73	0.60	0.66	0.94
Hinnerk Becker	0.62	0.67	0.62	14.4
Alex Chen	0.6	0.6	0.58	0.35
Ashida	0.55	0.46	0.5	8.7
HWDavid	0.44	0.46	0.45	0.3

Cuadro 2.1: Detección de texto natural para la colección del ICDAR.

### 2.2.2. Natural Street Pictures

A la hora de localizar con éxito el vocabulario de cada escena natural, la propuesta de B. Epshtain et al. [2] es extraer los posibles contornos de ancho similar (tipografía regular). Para corroborar el rendimiento del algoritmo SWT, los autores del artículo facilitan otra colección de 307 imágenes de exterior que detalla la Figura 2.5, donde aumenta de manera notable la dificultad de recuperar todo el texto en escenas de calidad y resolución superior (amplia distancia desde la cámara digital).



Figura 2.5: Sucesión de imágenes naturales para la colección NSP.

Como es lógico, el rendimiento del algoritmo para localizar la posición exacta del texto, depende de la legibilidad de los caracteres de interés (inferior en imágenes de exteriores). Así, los resultados que presenta el Cuadro 2.2 reflejan el grado de dificultad que aporta la colección NSP (presencia de vegetación, patrones repetitivos desde edificios, etc) respecto a la calidad del texto asociada al ICDAR 2003.

Algoritmo	Precisión	Recall	<i>F</i> -Measure
SWT [2]	0.54	0.42	0.47

Cuadro 2.2: Detección de texto natural para la colección NSP.

### 2.2.3. Chars74K

Asimismo, cabe identificar con éxito el contenido de cada palabra extraída de la escena. El artículo de T. de Campos et al. [3] estudia el proceso de reconocimiento de caracteres en imágenes naturales similares a las que presenta la Figura 2.6. El objetivo de los autores es evaluar el rendimiento de cada propuesta, con esta colección de en torno a 74000 imágenes (*Chars74K*) donde están incluidos los caracteres del alfabeto a considerar (0-9, A-Z, a-z). Por otra parte, destaca el empleo del alfabeto canarés (una de las 22 lenguas oficiales de la India y lengua oficial del estado de Karnataka), con la idea de aumentar la cantidad de clases y dificultar la tarea del clasificador.



Figura 2.6: Reconocer los caracteres asociados a la base de datos Chars74K.

El análisis del Cuadro 2.3 evalúa el rendimiento de cada clasificador de caracteres que estudia la literatura. Según el artículo de K. Wang et al. [1], la tasa de acierto es óptima para el algoritmo basado en Random Ferns, cuando la variabilidad entre los caracteres de entrenamiento y test es evidente (una mayor similaridad entre caracteres para la colección Chars74K, facilita la labor del vecino más cercano).

Base de datos	Ferns [1]	MKL [3]	HOG+NN
Chars74K (English)	54 %	55.26 %	57.5 %
ICDAR (2003)	64 %	-	52 %

Cuadro 2.3: Tasa de acierto relativa al reconocimiento de caracteres.

#### 2.2.4. Street View Text

Para contrastar el rendimiento que ofrece el clasificador Random Ferns al reconocer el vocabulario asociado a cada escena natural, el artículo de K. Wang et al. [1] proporciona otra colección de 350 imágenes extraídas directamente del Street View de *Google Maps*, similares a las que detalla la Figura 2.7. No obstante, la escasa resolución de cada imagen natural así como la variabilidad entre caracteres de baja calidad, dificultan clasificar con éxito el texto de cada marquesina o rótulo de la escena, con respecto al texto del ICDAR. Por consiguiente, para facilitar el reconocimiento de las palabras, la base de datos SVT proporciona un léxico  $\mathcal{L}$  con 50 palabras para cada imagen, donde especifica el vocabulario que puede aparecer en la escena.



Figura 2.7: Extraer palabras de la colección de imágenes Street View Text.

Tras suponer que la detección de palabras sea ideal (extraer todo el texto disponible), la idea es evaluar el rendimiento de la propuesta que reconoce el contenido de cada palabra. De esta manera, el análisis del Cuadro 2.4 refleja la dificultad de reconocer el vocabulario de cada escena, incluso tras hacer uso del léxico  $\mathcal{L}$  de 50 palabras por imagen. De hecho, al prescindir del léxico, la tasa de acierto de la colección ICDAR desciende un 15 %.

Base de datos	Ferns+ $\mathcal{L}$ [1]
ICDAR (2003)	76 %
Street View Text	57 %

Cuadro 2.4: Tasa de acierto relativa al reconocimiento de palabras.

Finalmente, el objetivo es evaluar de manera conjunta la detección y el reconocimiento de palabras, para comparar el resultado con el artículo de L. Neumann et al. [6]. Por ende, según el análisis del Cuadro 2.5, el algoritmo de la propuesta localiza y clasifica con éxito cerca del 50 % del vocabulario.

Base de datos	Precisión	Recall	$F_2$ -Measure
ICDAR (2003)	0.45	0.54	0.51
Street View Text	0.50	0.31	0.38

Cuadro 2.5: Detección y reconocimiento de texto natural.



# Capítulo 3

## Algoritmos propuestos

El objetivo del trabajo fin de máster es desarrollar el algoritmo que permita determinar la presencia de texto de cada escena natural, desde cada imagen o vídeo de entrada, para proceder a localizar la posición exacta del vocabulario y reconocer el contenido de cada sucesión de caracteres alineados de interés. Por consiguiente, cada apartado del Capítulo 3 detalla el conjunto de técnicas que describen el funcionamiento del sistema que presenta la Figura 3.1. Como resultado, el usuario recibirá por pantalla la lectura del texto a procesar (Canto Do Brasil, RESTAURANT y 6268727)



Figura 3.1: Esquema que detalla el funcionamiento de la aplicación propuesta.

### 3.1. Localización del texto

Como punto de partida, interesa localizar los caracteres por separado, de las palabras que aparecen en la escena. Atendiendo al análisis de los artículos del Capítulo 2, conviene evitar la aproximación de Kai Wang et al. [1], por el elevado coste computacional asociado a la ventana deslizante que recorre la escena a distintas escalas.

Por tanto, la propuesta del trabajo actual es localizar el vocabulario en función de las características típicas de los caracteres a extraer (contornos cerrados que contrastan con el fondo de manera clara). No obstante, la calidad de los caracteres así como el contraste con el fondo de la escena, es esencial para detectar contornos de iluminación regular.

#### 3.1.1. Segmentación de contornos

A priori, los caracteres son regiones cerradas que contrastan con el fondo claramente. La segmentación es la técnica que permite etiquetar cada píxel de la imagen, de manera que los dígitos o letras de interés terminen con un valor distinto al del fondo (el objetivo será encontrar estas componentes conexas).

La primera aproximación a estudiar es la propuesta de Jiri Matas et al. [8] que emplea la técnica denominada MSER (*Maximally Stable Extremal Regions*). El algoritmo MSER analiza la imagen binarizada para todos los umbrales posibles (de 0 a 255 niveles de gris). Así, la región extrema más estables hace referencia a cada zona o vecindad de la imagen que permanece sin cambios tras aplicar una serie de umbrales consecutivos.

El problema radica a la hora de procesar ciertas imágenes en las que no existe suficiente contraste entre los caracteres que se desea segmentar y el fondo no homogéneo de la escena respectiva. Realmente, es habitual que en cada imagen natural donde la iluminación no es uniforme, ocurra que píxeles del mismo carácter a segmentar tengan niveles de gris muy diferentes. Como alternativa a la técnica MSER, la umbralización local o adaptativa puede resolver este problema haciendo que el valor del umbral  $U_k$  varíe según las características locales del entorno a evaluar. A continuación cabe especificar el algoritmo de umbralización adaptativa a utilizar:

**Paso 1.-** Dividir la imagen original  $I(i, j)$  en subimágenes  $I_k(i, j)$  donde se supone que los cambios de iluminación no son tan fuertes.

**Paso 2.-** Determinar independientemente un umbral  $U_k$  para cada subimagen  $I_k(i, j)$ .

**Paso 3.-** Si en alguna subimagen no se puede determinar su umbral, calcularlo mediante la interpolación de valores de los umbrales de subimágenes vecinas.

**Paso 4.-** Procesar cada subimagen con respecto a su umbral local  $U_k$ .

El proceso de segmentación basado en la detección de contornos cerrados, hace uso de la información que proporciona la frontera de los objetos de interés que aparecen dentro de cada imagen binaria, producto de la umbralización adaptativa (b). Parece lógico pretender localizar los caracteres, sin más que encontrar estas regiones conexas respectivamente (c). Así, cuando los caracteres muestran un color uniforme y distinto del fondo, debe mejorar la nitidez de cada componente conexa a extraer. En la Figura 3.2 se detalla el funcionamiento del algoritmo de detección de texto que propone el artículo de S. Suzuki y K. Abe [9].

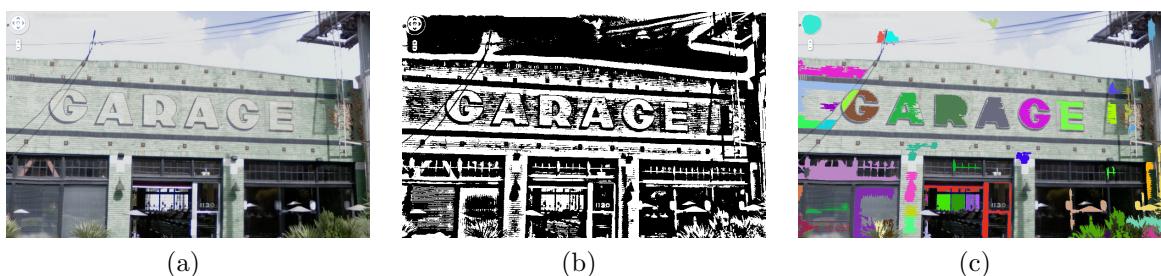


Figura 3.2: Contornos a extraer mediante la umbralización adaptativa propuesta.

Llegados a este punto, puede ser interesante reducir el número de contornos detectados, con el objetivo de mantener solo las componentes conexas relativas al texto. Por tanto, la idea es proceder a aplicar el siguiente filtro geométrico que elimine las regiones en función de la dimensión asociada al rectángulo que se ajusta a cada contorno (bounding box):

- Establecer la dimensión mínima para cualquier región extraída ( $14 \times 20$  píxeles) con la idea de excluir las componentes conexas de cada escena, que no alcancen el ancho y alto necesario.

- Verificar que el área asociada al bounding box de cada contorno no supera el máximo establecido para un carácter normal. Así, el área que ocupa el texto depende en cierta medida del tamaño de la imagen.
- Limitar la relación de aspecto asociada al rectángulo que se ajusta a cada contorno y descartar regiones de alto desmedido. Respecto al ancho del contorno, cabe soportar la presencia de posibles caracteres unidos.

### 3.1.2. División de caracteres unidos

La segmentación basada en la detección de caracteres por separado, no es buena cuando la proximidad entre caracteres es grande y el contraste sobre el fondo es escaso (b). Antes de proceder a reconocer cada contorno de la escena, es importante segmentar con éxito el vocabulario por caracteres (c). De hecho, aún detectando todos los caracteres de manera individual, será necesaria esta etapa para separar fuentes de tipografía continua (véase el texto manuscrito). La Figura 3.3 presenta la separación óptima de los posibles contornos con caracteres ligados, que resisten el filtro geométrico previo.



Figura 3.3: Dividir los contornos de ancho excesivo para evitar posible texto unido.

El algoritmo de segmentación dividirá en partes iguales cada contorno extraído, para explotar la relación de aspecto asociada al bounding box, y separar cada región en función de la relación de aspecto ideal para cualquier carácter ( $ancho/alto = 1,428$ ). Por tanto, la separación mejora cuanto menor sea la cantidad de caracteres ligados y más parecidos sean entre sí (cuidar caracteres de ancho distinto como ‘l’ y ‘w’). Como alternativa, cabe estudiar el empleo de los valles de separación entre caracteres o la proyección horizontal del texto, para separar el léxico de manera precisa.

### 3.1.3. Clasificador binario de texto

Después de separar con éxito cada contorno en función de la relación de aspecto ideal de cualquier carácter, conviene verificar como la cantidad de contornos extraídos desde las imágenes naturales, resulta aún demasiado elevada, incluso tras limitar el tamaño de las componentes conexas con el filtrado geométrico de la propuesta. Para proceder a la etapa de formación de palabras, es importante reducir la cantidad de contornos que produce la técnica de umbralización local o adaptativa, para facilitar la labor del algoritmo que trata de agrupar cada sucesión de caracteres alineados (los falsos positivos dificultan estimar el vocabulario por palabras).

Como consecuencia, para desechar la mayor parte de los contornos restantes que no sean dígitos o letras de interés, la idea es entrenar un clasificador de texto biclase que reciba la subimagen de cada contorno en escala de grises (región que se ajusta al bounding box) y devuelva como salida la predicción adecuada (caracter / no caracter). Para determinar el rendimiento del clasificador de texto, cabe destacar la importancia del recall a la hora de evitar rechazar algún carácter de manera errónea (falsos negativos). La Figura 3.4 detalla el efecto del clasificador de texto a implementar (b), donde es preferible no rechazar texto válido, a tratar de eliminar por completo los falsos positivos de la escena (c).

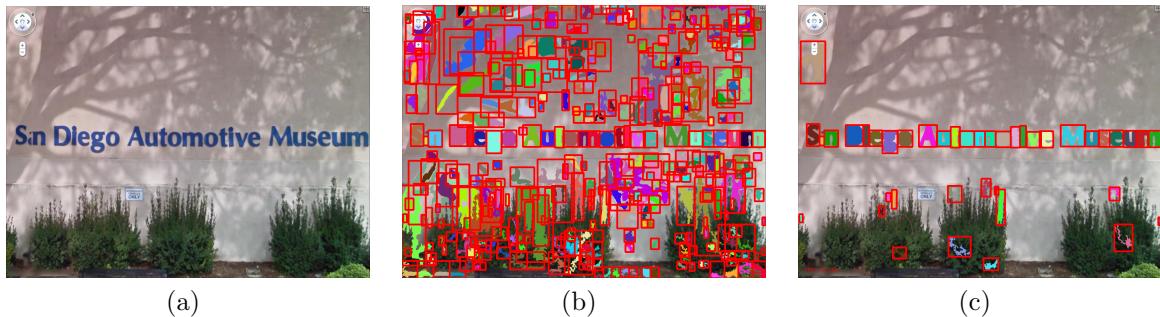


Figura 3.4: Reducir el número de contornos aplicando el clasificador de texto.

Tras realizar los experimentos necesarios del Apartado 4.1.2, la propuesta del trabajo es hacer uso del clasificador SVM (*Support Vector Machines*) que ofrece el mejor rendimiento. Así, las máquinas de vectores soporte desarrolladas por V. Vapnik [10] hacen referencia a un tipo de clasificador supervisado capaz de construir el hiperplano óptimo que separa las distintas clases asociadas al espacio del problema. La idea del modelo basado en SVM de la Figura 3.5 es generar el hiperplano que proporciona un margen geométrico máximo entre los elementos de las dos categorías a diferenciar. Las muestras más cercanas al hiperplano (vectores soporte  $z_i$ ) son las encargadas de formar la función discriminante  $f(x)$ .

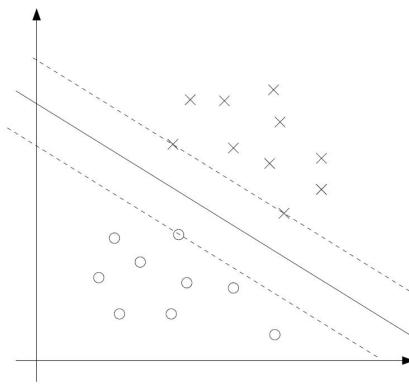


Figura 3.5: Construcción del hiperplano mediante los vectores soporte  $z_i$ .

El SVM es un algoritmo de aprendizaje supervisado que busca dividir de manera lineal las muestras de entrenamiento. No obstante, cuando esta colección de entrenamiento no es linealmente separable (aspecto parecido entre caracteres y falsos positivos), el clasificador no es capaz de generalizar el problema correctamente. La alternativa será transformar los datos de entrada a un espacio de mayor dimensión (estimador no lineal).

Sin ir más lejos, la Figura 3.6 presenta la transformación no lineal  $\phi(x)$  que implica la correspondencia hacia un espacio  $\mathbb{R}^3$  donde construir el hiperplano óptimo que separa las clases de ejemplo. Por consiguiente, para establecer dicha correspondencia  $\phi(x)$ , conviene conocer el denominado *kernel trick*.

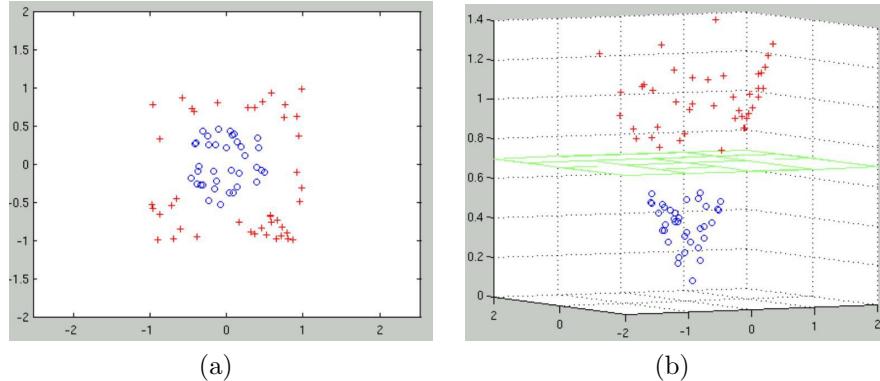


Figura 3.6: Transformar el espacio de entrada a otro de alta dimensionalidad.

El núcleo o kernel es la función  $K$  tal que para todo vector  $x, z$  tiene asociada la tupla  $K(x, z) = \langle \phi(x), \phi(z) \rangle$  (producto escalar de dos vectores multidimensionales), siendo  $\phi$  la transformación a un espacio de características de mayor dimensión donde las clases sí sean linealmente separables. No obstante, para resolver el problema con éxito no será necesario conocer explícitamente el valor de la transformación  $\phi$ , pues el kernel solo depende de los datos a través de productos escalares sobre el espacio de características transformado.

La clave del enfoque es encontrar esa función kernel  $K$  que permita evaluar de manera eficiente cada contorno extraído. Así, la propuesta es emplear el kernel estándar gaussiano (*Radial Basis Function*) de la Ecuación 3.3, donde las variables  $x$  y  $z$  representan el vector de características y el vector soporte respectivamente.

$$K(x, z) = e^{-\frac{\|x-z\|^2}{2\sigma^2}} \quad (3.1)$$

Como resultado, el objetivo será generar el mejor clasificador no lineal, tras seleccionar el valor óptimo para el parámetro  $\sigma$  asociado al kernel RBF. Asimismo, cabe incluir en el problema una variable de relajación que permita manejar un cierto error en la estimación. La constante  $C$  trata de compensar el error de entrenamiento, para modificar la holgura del margen del hiperplano y reducir la cantidad de vectores soporte, al tolerar datos mal clasificados (evitar situaciones de excesivo sobreajuste a los datos).

Finalmente, la Ecuación 3.4 presenta la función discriminante que clasifica cada vector de características de entrada  $x$ , dentro de las clases del problema (caracter / no caracter), siendo  $\alpha_i$  el peso asociado a cada vector soporte  $z_i$  del clasificador.

$$f(x) = \sum_{i=1}^N \alpha_i \cdot K(x, z_i) \quad (3.2)$$

## 3.2. Reconocimiento de líneas

Después de detectar la posición exacta de cada contorno, el objetivo es reconocer cada posible carácter restante de manera individual, para generar el listado con el contenido del vocabulario extraído. La propuesta será estimar de antemano qué caracteres forman cada palabra (sucesión de contornos similares alineados), para presentar el texto reconocido de manera ordenada, facilitando la lectura de izquierda a derecha.

### 3.2.1. Estimar líneas candidatas

Hasta el momento, la aplicación es capaz de detectar la ubicación de los contornos que forman el vocabulario de cada escena por separado. Sin embargo, para realizar la lectura adecuada del texto, será necesario agrupar por palabras la colección de contornos extraída (b). El algoritmo RANSAC (*Random Sample and Consensus*) propuesto por M. Fischler et al. [11] permitirá estimar de manera robusta el conjunto de palabras que aparece en la escena natural (c). La Figura 3.7 presenta las etapas del algoritmo RANSAC a la hora de calcular las líneas candidatas con el texto (HERITAGE, FORUM, HERITAGEFORUM, sArtand y AnnAragon).

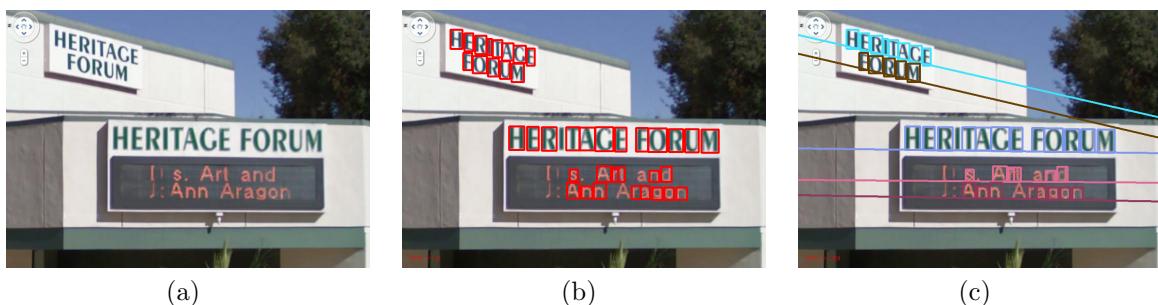


Figura 3.7: Establecer las líneas candidatas a partir de los caracteres a reconocer.

Dentro del campo de la visión artificial, conviene emplear RANSAC para la estimación robusta de modelos geométricos (rectas, homografías, etc) como técnica tolerante a datos atípicos, es decir a muestras que no pertenecen al modelo en cuestión. Por consiguiente, el principio del algoritmo es bastante simple y eficiente. Respecto a la etapa de selección de muestras, la idea es seleccionar de manera aleatoria el número mínimo de datos necesarios para definir el modelo  $l_{best}$  en cuestión. Como el objetivo es establecer rectas mediante los caracteres, basta con escoger solo dos contornos para definir la línea candidata a estudiar (elección aleatoria con la misma probabilidad de selección para cada muestra).

Acto seguido, hay que comprobar de las muestras restantes, cuántos contornos encajan con el modelo hipótesis  $l_{best}$ , o lo que es igual, qué cantidad de contornos detectados están alineados con la línea candidata a evaluar (número de inliers). Así, al aumentar la cantidad de inliers que encajan con el modelo, crece la probabilidad de encontrar el vocabulario de interés. Para terminar, tras realizar  $K$  iteraciones este proceso de selección de muestras y evaluación del modelo, la idea es elegir la línea candidata  $l_k$  de más inliers. Respecto al número  $K$  de iteraciones, dependerá de la probabilidad a priori de seleccionar el conjunto adecuado (elegir dos muestras de la misma palabra).

**Bucle.-** Comprobar la disponibilidad de una sucesión de contornos similares alineados para definir una nueva línea candidata  $l_{best}$ .

**Bucle.-** Realizar  $K$  iteraciones hasta encontrar la línea candidata  $l_{best}$  con el mayor número de inliers posible  $i_1, i_2, \dots, i_N$ .

**Paso 1.-** Seleccionar aleatoriamente dos contornos de los disponibles (datos) para formar cada línea candidata  $l_k$  a evaluar (modelo).

**Paso 2.-** Corroborar que la relación de aspecto de los contornos asociados a la línea candidata  $l_k$  sea aproximadamente similar.

**Paso 3.-** Calcular el número de inliers de la escena cuya distancia a la línea candidata  $l_k$  sea mínima (encajan con el modelo).

**Paso 4.-** Verificar que los inliers  $i_1, i_2, \dots, i_N$  que forman el texto relativo a la línea  $l_{best}$  están lo suficientemente próximos entre sí. En caso contrario, separar los inliers distantes dentro del mismo modelo (palabras diferentes).

**Paso 5.-** Eliminar los inliers  $i_1, i_2, \dots, i_N$  asociados a la línea de interés evaluada  $l_{best}$  para proceder a reconocer el texto restante. En principio, las palabras de la escena no deberían compartir caracteres.

En definitiva, la propuesta es reconocer el vocabulario de la escena natural, a partir de las secuencias de contornos alineados estimadas. De hecho, lo más lógico atendiendo a la implementación, será determinar primero la presencia de las palabras con mayor cantidad de caracteres (línea candidata  $l_{best}$  de más inliers).

Respecto a la formación de las distintas líneas candidatas, conviene limitar el número de caracteres mínimo por palabra, así como acotar la inclinación de la palabra respectiva (texto horizontal), evitando hacer uso de ciertos falsos positivos que resistan al clasificador de texto. Por último, cabe comprobar que los contornos asociados a cada línea candidata a estimar están lo suficiente próximos entre sí, como para considerar que forman parte de la misma palabra. Así pues, cuando la distancia de los contornos supera la posible pérdida de caracteres, la idea es separar la línea en grupos distintos.

La Figura 3.8 presenta una escena donde el vocabulario de interés, situado a la misma altura, agrupa los caracteres de palabras distintas sobre la misma línea candidata (véase FINE ITALIAN y FRESH MEATS). Como solución, al descartar la presencia de datos atípicos (falsos positivos) y verificar que cada sucesión de contornos distante supera el mínimo de caracteres exigido, es posible separar con éxito cada grupo en palabras.



Figura 3.8: Separar grupos de contornos distantes asociados a la misma línea.

### 3.2.2. Clasificador de caracteres

Llegados a la etapa de reconocimiento óptico de caracteres naturales, la idea es entrenar un clasificador multiclase (0-9, A-Z, a-z) que reciba de entrada la imagen binaria producto de la umbralización adaptativa, que se ajusta a cada contorno a identificar (bounding box). El clasificador debe generar para cada carácter, la probabilidad con respecto a cada dígito y letra que forma el alfabeto latino, para asignar finalmente el carácter más probable. Así, tras reconocer todos los caracteres de cada línea candidata, el sistema presenta al usuario de manera ordenada el texto de interés. La Figura 3.9 detalla el resultado de recuperar el texto de la escena (**Nicks** y **GREEKCORNER**) y ofrecer el léxico en la parte superior derecha de la imagen.



Figura 3.9: Reconocimiento de caracteres sobre las líneas candidatas extraídas.

Antes de entrar a especificar las características del clasificador de caracteres, conviene evaluar el formato de las imágenes de los contornos que forman cada línea candidata. Tras realizar los experimentos del Apartado 4.2.2, la propuesta del trabajo será redimensionar el tamaño de cada imagen a 14 x 20 píxeles y reconocer las imágenes binarias de los contornos similares a los que adjunta la Figura 3.10 (negro sobre fondo blanco respectivamente), para facilitar la labor del clasificador. Análogamente, como la umbralización adaptativa podría detectar caracteres blancos sobre fondo negro de interés, interesa calcular el color del fondo (píxeles de la frontera de la imagen) para reconocer la imagen complementaria cuando el carácter sea blanco y el fondo negro.



Figura 3.10: Clasificar los caracteres normalizados de la línea candidata.

Dada la alta dimensionalidad del problema a la hora de reconocer imágenes de 14 x 20 píxeles (vectores de características de 280 dimensiones), la propuesta es proceder a reducir esta dimensión inicial, mediante el algoritmo LDA (*Análisis Discriminante Lineal*), para eliminar datos que proporcionan información redundante. Asimismo, para diferenciar con éxito las  $C$  clases del problema (alfabeto latino), la idea es encontrar la combinación lineal de vectores adecuada. Por consiguiente, una vez reducido el espacio de características y con una separabilidad óptima entre clases, es posible seleccionar el clasificador.

El objetivo del algoritmo LDA es calcular la distancia entre las medias de cada clase  $\mu_i$  normalizadas por la dispersión de cada clase. Así, la matriz de proyección  $A_{C-1 \times 280}$  será la que maximice la función objetivo  $J(A)$  de la Ecuación 3.3 cuya solución viene dada por el problema de autovalores generalizado  $S_b A = \lambda S_w A$ . Cuando  $S_w$  sea no singular (matriz bien condicionada) será posible resolver el problema mediante  $S_w^{-1} S_b = P \wedge P^T$ . Por tanto, la matriz de proyección será  $A = P_{C-1}^T$ , siendo  $P_{C-1}^T$  una matriz cuyas columnas son los  $C - 1$  autovectores con mayor autovalor.

$$J(A) = \frac{|AS_b A^T|}{|AS_w A^T|} \quad (3.3)$$

Donde  $S_b$  será la matriz de dispersión inter-clase definida en la Ecuación 3.4 y  $S_w$  la matriz de dispersión intra-clase de la Ecuación 3.5.

$$S_b = \sum_{i=1}^C n_i (\mu_i - \mu)(\mu_i - \mu)^T \quad (3.4)$$

$$S_w = \sum_{i=1}^C \sum_{x_j \in c_i} (x_j - \mu_i)(x_j - \mu_i)^T \quad (3.5)$$

El LDA multiclase estudiado tiene asociados una serie de problemas cuando la matriz de dispersión intra-clase  $S_w$  está mal condicionada (no es de rango completo). La solución será aplicar la técnica conocida como PCA (*Análisis de Componentes Principales*) que permite maximizar de inicio la varianza de las características discriminantes. El objetivo será reducir previamente la dimensión de inicio del problema (280 dimensiones) con PCA, para después aplicar LDA sobre el conjunto de vectores de características ya reducido y maximizar el índice de separabilidad de las clases en el espacio transformado.

Para encontrar la matriz de proyección  $A_{K \times 280}$  según la cual los datos queden mejor representados en términos de mínimos cuadrados, en primer lugar se debe de centrar el conjunto original de datos en la media de cada clase respectivamente. Esta matriz  $A_{K \times 280}$  será la que maximice la función objetivo  $J(A)$  de la Ecuación 3.6 cuya solución viene dada desde el problema de autovalores asociado a la matriz de covarianzas  $\sum_{\chi} = \frac{1}{n} \hat{X} \hat{X}^T$  resuelto mediante  $\sum_{\chi} = P \wedge P^T$ . La matriz de proyección será  $A = P_K^T$ , siendo  $P_K^T$  una matriz cuyas columnas son los  $K$  autovectores que retienen la variabilidad seleccionada.

$$J(A) = \sum_{i=1}^K \frac{a_i^T \sum_{\chi} a_i}{\|a_i\|^2}, A = [a_1, a_2, \dots, a_K] \quad (3.6)$$

Ahora sí, en un espacio de características transformado, de dimensión reducida y con esa separabilidad entre clases óptima, será el momento de seleccionar el clasificador que permita reconocer los distintos caracteres. Al suponer un completo conocimiento a priori de la estructura estadística de las clases, el aprendizaje se reduce a la estimación de los parámetros que determinan las funciones de densidad de probabilidad  $p(x|\alpha_i)$  de las clases. La propuesta es hacer uso del clasificador bayesiano de la Ecuación 3.7 para clasificar cada vector de características  $x$ .

$$p(x|\alpha_i) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}} \quad (3.7)$$

No obstante, la probabilidad a priori  $P(\alpha_i)$  de clasificar un vector de características  $x$  como de la clase  $i$ , será la misma para cada dígito y letra asociada al alfabeto a reconocer (0-9, A-Z, a-z). Por otra parte, la función de densidad de probabilidad marginal  $p(x)$  es común para todas las clases, con lo cual se podrá ignorar también. Calcular la probabilidad a posteriori  $p(\alpha_i|x)$  mediante la regla de Bayes de la Ecuación 3.8 se reduce al cálculo de la probabilidad a priori.

$$p(\alpha_i|x) = \frac{p(x|\alpha_i) \cdot P(\alpha_i)}{p(x)} \quad (3.8)$$



# Capítulo 4

## Resultados experimentales

Durante el desarrollo del Capítulo 3, el trabajo fin de máster especifica los algoritmos para detectar y reconocer, a partir de una imagen o vídeo de entrada, el posible vocabulario que aparece en la escena (recuperar el texto de la escena natural). Ahora, el objetivo es proceder con los experimentos de clasificación oportunos (tanto construcción de las bases de datos como elección del clasificador) que determinan el rendimiento de la propuesta.

### 4.1. Detección de caracteres

El sistema debe localizar la posición de los caracteres que aparecen en la escena natural, tras filtrar las componentes conexas que detecta la umbralización adaptativa (extraer todo contorno cerrado) que implementa la biblioteca OpenCV [14]. Por tanto, la idea es calcular la tasa de acierto del algoritmo tras filtrar cada dígito o letra de entre todos los contornos, mediante el clasificador biclase de texto a entrenar.

#### 4.1.1. Construcción de la base de datos

Respecto al clasificador de texto (caracter / no caracter) que pretende desechar gran parte de los contornos extraídos que no son dígitos o letras, cabe entrenar la base de datos adecuada para realizar los experimentos que establecen el rendimiento del clasificador. A la hora de generar la colección de imágenes, el objetivo es recuperar desde la base de datos *Street View Text* del Apartado 2.2.4, tanto los caracteres de cada escena, como los falsos positivos que genera la segmentación de componentes conexas. La Figura 4.1 presenta el aspecto de las regiones extraídas para entrenar el clasificador.



Figura 4.1: Diferentes contornos de interés para entrenar el clasificador de texto.

Esta base de datos incluye una colección de hasta 5660 imágenes seleccionadas sobre diferentes condiciones de variabilidad (2830 imágenes por cada clase), con el fin de eliminar de manera eficaz el efecto de la variabilidad no deseada relativa a los caracteres naturales (fuente tipográfica, dimensión del contorno, etc).

### 4.1.2. Experimentos de clasificación

La biblioteca OpenCV proporciona un método sencillo para convertir la imagen natural de niveles de gris o color a una imagen binaria, tras aplicar el método *adaptiveThreshold* basado en la umbralización adaptativa del Apartado 3.1.1. Así, para que funcione como es debido se deben configurar los parámetros adecuados: **frame** recibirá la imagen de entrada; **bw\_frame** la imagen binaria procesada; **max\_value** la intensidad asociada a los píxeles que cumplan la condición del umbral; **adaptive\_method** la técnica que establece a partir de los vecinos (media o filtro gaussiano) la etiqueta de cada píxel; **threshold\_type** asigna el valor establecido al píxel en función de si supera o no cierto umbral; **kernel\_value** la dimensión impar de la máscara (3, 5, 7, ...) que evalúa la intensidad de los píxeles vecinos y **c\_value** establece el contraste aplicable entre los caracteres a segmentar y el fondo.

```
adaptiveThreshold(frame, bw_frame, max_value, adaptive_method,
                  threshold_type, kernel_value, c_value);
```

A continuación, el objetivo será especificar el valor a asignar a cada variable para entrar a evaluar el rendimiento de la umbralización adaptativa, a la hora de recuperar con éxito el texto de las imágenes de test del Street View. De esta manera, para establecer el umbral  $U$  de cada píxel, la idea será determinar la media de los vecinos (**ADAPTIVE\_THRESH\_MEAN\_C**) y etiquetar a blanco el píxel (**max\_value = 255**) cuando supera el umbral  $U$  respectivo (**THRESH\_BINARY**). No obstante, la variabilidad asociada a los caracteres así como la escasa resolución de las imágenes, dificultan la detección de las componentes conexas de interés.

La Figura 4.2 detalla el grado de dificultad de segmentar correctamente las fronteras de los caracteres que aparecen en la escena. Dependiendo de las características que presenta el texto de la imagen natural a procesar (separación entre caracteres, contraste con el fondo, resolución de la imagen, etc) el valor ideal para **kernel\_value** y **c\_value** será distinto. Por consiguiente, para intentar extraer cualquier dígito o letra de cada imagen, será importante repetir el algoritmo de umbralización adaptativa, cambiando el valor de ciertos parámetros (**kernel** y **C**) para evitar perder texto:

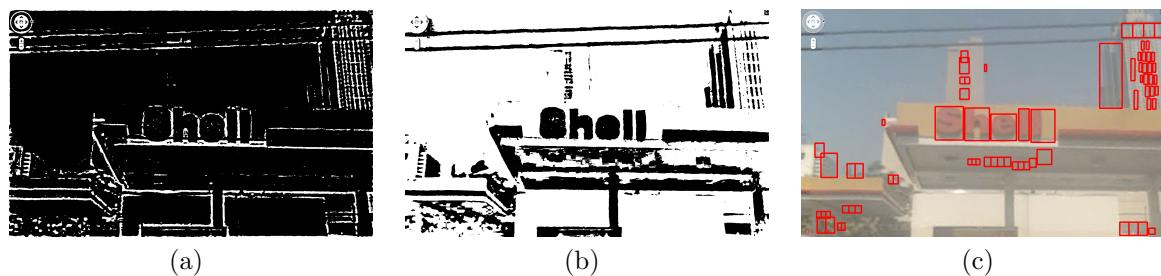


Figura 4.2: Determinar el *kernel* y *C* adecuado para recuperar el texto de la imagen.

- Para localizar caracteres de color uniforme que contrastan claramente con el fondo de la escena, conviene reducir la constante *C* que aumenta el contraste exigido entre texto y fondo (**c\_value = -5**). Respecto a la dimensión de la máscara local a evaluar (**kernel\_value = 35**), un valor inferior mejora la precisión al segmentar. De hecho, cuando la resolución sea notable, es posible extraer con éxito los caracteres próximos entre sí por separado.

- Para determinar la presencia de caracteres de baja resolución en la escena, conviene aumentar la dimensión de la máscara (`kernel_value = 75`) al detectar los bordes de cada región conexa, a costa de perder precisión al segmentar. Después, la propuesta es incrementar el valor de la constante C (`c_value = 4`) para encontrar los contornos de escaso contraste.

Como resultado, al aplicar varias veces la técnica de umbralización local con distintos valores para establecer el umbral, la cantidad de componentes conexas a extraer, supera en media los 2000 contornos para imágenes de 1280 x 800 píxeles. Sin embargo, a la hora de recuperar cada palabra como sucesión de contornos en línea, solo interesa mantener las componentes conexas con caracteres. Por tanto, la idea es rechazar parte de los contornos que no sean dígitos o letras, en función de su dimensión y su aspecto, mediante el filtrado geométrico y el clasificador de texto respectivo.

El filtro geométrico a implementar permite reducir en torno al 90 % de las componentes conexas (200 contornos restantes). Primeramente, al establecer un tamaño mínimo para cualquier letra o dígito a reconocer de 14 x 20 píxeles, es factible descartar todo contorno con un ancho o alto menor. Asimismo, para rechazar regiones demasiado grandes, la idea es limitar el área máxima del rectángulo ajustado al contorno en cuestión (bounding box). Para terminar, conviene eliminar contornos con una relación de aspecto desproporcionada, inferior a 0.2 o superior a 6 (siendo 1.42 la proporción ideal). Tras soportar la presencia de caracteres ligados, será importante separar en partes iguales cada contorno, en función de la relación de aspecto ideal, para proceder a clasificar cada región por separado.

Llegados a este punto, conviene reducir aún más la cantidad de contornos que genera el proceso de umbralización adaptativa, para facilitar la tarea del algoritmo RANSAC que permita agrupar los caracteres similares alineados, formando el vocabulario de interés. La propuesta es entrenar el clasificador de texto (caracter / no caracter) para poder conservar en torno a un 15 % de las componentes conexas de inicio (30 contornos por imagen) donde resistan los caracteres visibles desde la escena.

Tras construir una base de datos con los caracteres y falsos positivos de entrenamiento que presenta el Apartado 4.1.1, llega el momento de comenzar con los experimentos desde el entorno de desarrollo MATLAB [16], al generar el clasificador biclase óptimo. Respecto al vector de características de entrada, el clasificador pretende reconocer la imagen de cada contorno en escala de grises, normalizada a 14 x 20 píxeles de tamaño (280 dimensiones). Así, para garantizar que los resultados obtenidos sean independientes a cierta colección de entrenamiento elegida, la propuesta será hacer uso de la técnica conocida como validación cruzada, que consiste en repetir el experimento, para calcular una media aritmética de las medidas de evaluación. De esta manera, con 5 experimentos (*5-fold*), la validación emplea 4528 imágenes para entrenar y 1132 imágenes para clasificar, eligiendo en cada iteración una secuencia de imágenes distinta.

Asimismo, dada la alta dimensionalidad del problema, el objetivo ahora es reducir esta dimensión inicial mediante PCA, para después aplicar la técnica LDA sobre el conjunto de vectores de características reducido y maximizar el índice de separabilidad de ambas clases en el espacio transformado.

PCA permite construir una transformación lineal  $A_{N \times 280}$  que ayude a escoger un nuevo sistema de coordenadas donde la varianza de mayor tamaño se captura en el eje principal, la segunda varianza más grande en el segundo eje, y así sucesivamente formando estas  $N$  componentes principales. Por otra parte, mediante LDA será posible obtener la matriz de proyección  $A_{1 \times N}$  que reduce el problema a una única dimensión ( $C - 1$ ). De hecho, en este nuevo espacio generado  $Z_{1 \times 5660}$  que presenta la Ecuación 4.1, las imágenes de los distintos contornos deben quedar separadas, cuando se cumplen las hipótesis del LDA.

$$Z_{1 \times 5660} = LDA_{1 \times N} \cdot PCA_{N \times 280} \cdot X_{280 \times 5660} \quad (4.1)$$

Con el espacio de características reducido, es el momento de seleccionar el clasificador que permita recuperar el texto de interés. Primero, suponiendo un completo conocimiento a priori de la estructura estadística de ambas clases (caracter / no caracter), la propuesta será hacer uso del clasificador bayesiano estudiado en el Apartado 3.2.2. Sin embargo, el rendimiento que ofrece el clasificador, tras recibir como vector de características la región de cada contorno en escala de grises, incluso al calcular la dimensión de PCA óptima ( $N$ ), no será suficiente como detalla este Cuadro 4.1. Como alternativa, para evitar la pérdida de caracteres de la escena, conviene aumentar la probabilidad a priori  $P(\alpha_i)$  de reconocer cualquier contorno como texto, con la idea de disminuir la tasa media de falsos negativos (mejorar el recall del clasificador).

$P(\alpha_i)$	$N$	Precisión	Recall	$F_2$ -Measure
[0.5, 0.5]	1	0.582	0.699	0.672
[0.4, 0.6]	1	0.537	0.936	0.815

Cuadro 4.1: Rendimiento del clasificador bayesiano para escala de grises.

Para mejorar el rendimiento, el Cuadro 4.2 detalla la calidad del clasificador bayesiano al recibir, como vector de características, la imagen binaria de cada contorno a reconocer, producto de la umbralización adaptativa previa. De esta forma, debe ser posible reducir la variabilidad no deseada relativa a los caracteres naturales, para mejorar así el rendimiento del clasificador biclase que permite desechar los contornos restantes que no sean dígitos o letras de interés.

$P(\alpha_i)$	$N$	Precisión	Recall	$F_2$ -Measure
[0.5, 0.5]	61	0.682	0.819	0.787
[0.4, 0.6]	70	0.648	0.892	0.829

Cuadro 4.2: Rendimiento del clasificador bayesiano para blanco y negro.

Por tanto, la Figura 4.3 especifica el rendimiento del clasificador estadístico bayesiano, con la dimensión de PCA óptima ( $N = 70$ ), tras aumentar la probabilidad a priori  $P(\alpha_i)$  de reconocer cualquier contorno extraído como texto y reducir la dimensión del problema, de acuerdo a la transformación lineal  $LDA_{1 \times 70} \cdot PCA_{70 \times 280}$  que minimiza la probabilidad de error al clasificar.

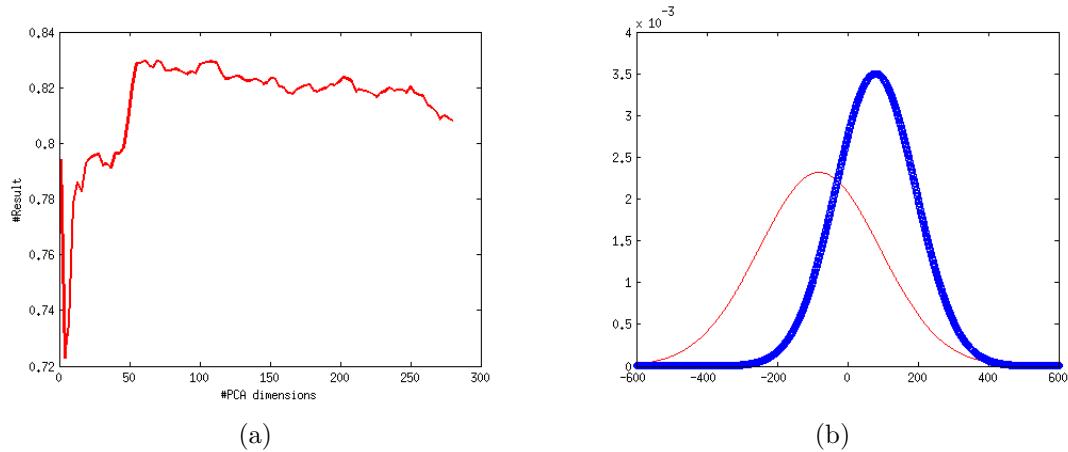


Figura 4.3: Hipótesis de distribución gaussiana que minimiza la probabilidad de error.

Como es evidente, resolver este problema de clasificación modelando estadísticamente la distribución de las muestras de ambas clases, no basta para separar con éxito de manera lineal los datos de entrenamiento. Como alternativa, conviene comparar el rendimiento del clasificador bayesiano, con respecto al clasificador asociado a los  $K$  vecinos más cercanos. Con este método no paramétrico, será suficiente suponer que los  $K$  vecinos más próximos al vector de características  $x$ , aportan la información relevante para estimar la función de densidad de probabilidad  $p(\alpha_i, x)$ .

Respecto a la elección del  $K$ , conviene seleccionar el valor óptimo durante el proceso de validación cruzada, para evitar la presencia de datos atípicos o características irrelevantes al clasificar. Así, el rendimiento que ofrece el clasificador KNN, tras reducir igualmente la dimensión del problema con PCA y maximizar el índice de separabilidad de ambas clases con LDA, es ligeramente inferior al que proporciona la distribución gaussiana.

Como resultado, parece lógico descartar esa idea de separar linealmente las muestras de entrenamiento relativas al reconocimiento de texto natural. De hecho, el modelo estadístico bayesiano solo es válido para clases de distribución unimodal con escasa variabilidad. Con la propuesta de A. Martínez et al. [12], cabe la posibilidad de evitar el conflicto entre las clases (caracter / no caracter), tras dividir estas clases en distintas subclases mediante el algoritmo SDA (*Análisis Discriminante de Subclases*) facilitando la separación óptima de las muestras al hacer la proyección. No obstante, el rendimiento que presenta el Cuadro 4.3 sigue siendo insuficiente para desechar los contornos sin pérdida de caracteres de interés.

Técnica	$K$	Precisión	Recall	$F_2$ -Measure
PCA+LDA	13	0.676	0.775	0.759
SDA	3	0.699	0.812	0.785

Cuadro 4.3: Rendimiento del clasificador KNN sobre imágenes binarias.

Por tanto, para la resolución de problemas de carácter no lineal, la alternativa es usar el clasificador SVM del Apartado 3.1.3 con el fin de generar el hiperplano que proporciona un margen geométrico máximo entre los elementos de las 2 categorías a diferenciar.

A la hora de construir el clasificador no lineal deseado, el objetivo es transformar cada vector de entrada  $x$ , mediante una función kernel  $K(x, z) = \langle \phi(x), \phi(z) \rangle$ , a un espacio de mayor dimensión, siendo  $\phi$  el mapeo a un espacio de características donde las clases sí son linealmente separables. De entre las distintas funciones kernel, la propuesta será hacer uso del kernel RBF (*Radial Basis Function*), para aproximar la predicción  $f(x)$  por medio de una combinación lineal de gaussianas.

Respecto a la elección, tanto de la constante  $C$  que especifica la holgura del margen del clasificador (manejar cierto error en la estimación, al reducir el sobreajuste a los datos de entrenamiento), como del parámetro  $\sigma$  asociado al kernel RBF, la idea será seleccionar, mediante el proceso de validación cruzada, cada valor para generar el clasificador de texto óptimo. La Figura 4.4 detalla la elección de ese parámetro  $\sigma$  que determina el rendimiento del clasificador SVM.

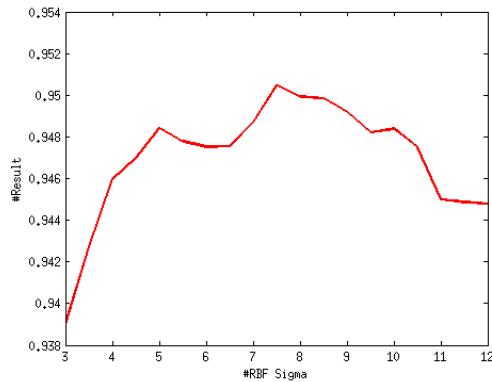


Figura 4.4: Determinar el  $\sigma$  óptimo asociado al kernel RBF.

Sin embargo, desde el punto de vista computacional, el número de vectores soporte  $z_i$  (muestras cercanas al hiperplano que forman la superficie de decisión) limita el tiempo de ejecución del sistema al reconocer cada contorno (caracter / no caracter). Por consiguiente, el objetivo es intentar reducir la cantidad de vectores soporte, sin acusar excesiva pérdida de rendimiento. Así pues, el análisis del Cuadro 4.3 presenta el clasificador SVM óptimo con  $\sigma = 7.5$  y  $C = 16$  (tasa del 95 % de acierto). No obstante, al aumentar  $\sigma$  y la holgura del margen, es posible disminuir el sobreajuste para reducir la cantidad de vectores soporte hasta un 35 % con una pérdida mínima de rendimiento del 1 %.

Vectores soporte	$\sigma$	C	Precisión	Recall	$F_2$ -Measure
3895	3	6	0.819	0.975	0.939
2525	5	46	0.907	0.959	0.948
1640	7.5	16	0.952	0.950	0.950
1057	10.5	200	0.948	0.944	0.945

Cuadro 4.4: Rendimiento del clasificador SVM sobre escala de grises.

## 4.2. Reconocimiento de palabras

El sistema debe estimar, a partir de la colección de caracteres extraída, cada secuencia de contornos similares alineados que forman el vocabulario visible desde la escena natural. Por lo tanto, el objetivo es implementar mediante la biblioteca OpenCV [14], el algoritmo que agrupa los caracteres por palabras (RANSAC) y proceder a clasificar el contenido de cada carácter de manera individual, para calcular la tasa de acierto del OCR.

### 4.2.1. Construcción de la base de datos

Respecto al clasificador de caracteres (0-9, A-Z, a-z) que permite reconocer cada dígito o letra posible, cabe entrenar la base de datos adecuada para realizar los experimentos que establecen el rendimiento de ese clasificador multiclase. A la hora de generar las imágenes, el objetivo es reconocer el alfabeto latino sobre caracteres de aspecto similar a la colección *Street View Text* del Apartado 2.2.4. Por ende, para clasificar con éxito el aspecto de cada contorno (vector de características), es necesaria una base de datos donde abarcar las 62 clases del problema: 10 dígitos, 26 mayúsculas y 26 minúsculas.

Asimismo, para afrontar con éxito la clasificación de caracteres naturales, la propuesta es entrenar el clasificador multiclase con la idea de soportar el estilo y la apariencia que proporciona cada fuente tipográfica (*True Type Font*) al carácter. La biblioteca FreeType [15] escrita en C++, implementa un motor para el manejo de letras que posibilita convertir la imagen en formato vectorial de cada carácter, a mapa de bits (proceso de rasterización). Así, para construir la base de datos de manera sintética, la propuesta es extraer mediante FreeType los caracteres de interés de cada tipo de letra a identificar (hasta 15 tipografías distintas): Andale Mono, Arial, Arial black, Comic Sans MS, Courier New bold, Georgia, Georgia bold italic, Impact, OCR-B, Times New Roman, Times New Roman bold italic, Trebuchet MS, Trebuchet MS bold italic, Verdana y Verdana bold italic.

Llegado el momento de generar esa colección de caracteres para cada clase, cabe aplicar el conjunto de transformaciones a detallar, que permiten asemejar los caracteres sintéticos con los extraídos por el algoritmo de cada escena natural. La Figura 4.5 presenta el proceso que modifica cada carácter, para dificultar el reconocimiento de los caracteres.



Figura 4.5: Procesar los caracteres sintéticos desde FreeType.

- *Rotación de cada carácter tipográfico.* La biblioteca FreeType ofrece la posibilidad de transformar la imagen vectorial de cada carácter tipográfico a mapa de bits, sin pérdida de información. Así, para simular una ligera rotación previa en el carácter, la idea es aplicar para cada clase, la matriz de rotación  $R(\theta)$  que detalla la Ecuación 4.2, donde  $\theta$  representa el ángulo de rotación (radianes) en sentido antihorario. Al aplicar ligeras rotaciones al carácter desde  $\theta \in [-5, 5]$  (grados), la producción aumenta hasta 11 imágenes por carácter.

$$R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (4.2)$$

- *Transformación afín aleatoria.* Para cada imagen de mapa de bits generada, la idea es modificar el aspecto ideal del carácter sometiéndole a trasformaciones geométricas que producen el efecto de pixelado de interés. Así, la biblioteca OpenCV proporciona el método *warpAffine* para proceder a escalar y trasladar de manera aleatoria el carácter de cada imagen (aumentar la variabilidad de la base de datos). Respecto a los parámetros adecuados a evaluar: **src** recibirá la imagen de entrada; **dst** la imagen transformada; **warp\_mat** la matriz  $T(\alpha, dx, dy)$  que detalla la Ecuación 4.3, donde  $\alpha$  representa el factor de escala y  $[dx, dy]$  la translación aleatoria del carácter; **dst\_size** la dimensión de la imagen de salida y **flags** el método de interpolación.

$$T(\alpha, dx, dy) = \begin{bmatrix} \alpha & 0 & dx \\ 0 & \alpha & dy \end{bmatrix} \quad (4.3)$$

```
warpAffine(src, dst, warp_mat, dst_size, flags);
```

- *Suavizado gaussiano.* La escasa resolución al procesar cada escena natural, dificulta el reconocimiento de los caracteres. Por tanto, para producir el efecto de desenfoque y reducir la nitidez de cada dígito o letra (reducir los detalles), la biblioteca OpenCV proporciona el método *GaussianBlur* a estudiar. La distribución gaussiana controla el alcance del suavizado con una matriz donde el peso disminuye al alejarse del píxel central. A la hora de configurar los parámetros adecuados: **src** recibirá la imagen de entrada; **dst** la imagen suavizada; **kernel\_size** la dimensión de la máscara que difumina los detalles de la imagen y  $[\sigma_x, \sigma_y]$  la desviación estándar.

```
GaussianBlur(src, dst, kernel_size, sigma_x, sigma_y);
```

- *Operación morfológica.* La teoría de conjuntos da origen a la morfología matemática que trata el problema de modificar el aspecto original de cada carácter. La biblioteca OpenCV proporciona el método *dilate* para ensanchar de manera aleatoria el trazo de cada carácter (distinto grosor con la misma tipografía). Respecto a los parámetros a evaluar: **src** recibirá la imagen de entrada; **dst** la imagen resultado; **kernel** el elemento estructurante de la dilatación; **anchor** la posición del kernel respecto al eje de ordenadas y **iterations** el número de repeticiones del algoritmo.

```
dilate(src, dst, kernel, anchor, iterations);
```

- *Reducción del contraste.* Para intentar facilitar la labor del clasificador, cabe destacar el contraste de los caracteres sobre el fondo (negro sobre blanco respectivamente). De hecho, para simular el mismo contraste desde la colección sintética, el objetivo es asignar aleatoriamente a cada píxel del carácter un valor cercano al negro  $\in [0, 55]$ , mientras que para cada píxel relativo al fondo el blanco  $\in [102, 255]$ . Como resultado, la idea es umbralizar la escena para generar la imagen binaria (ampliar los intervalos de blanco y negro genera el ruido de la imagen).

Por consiguiente, la base de datos a generar debe incluir una colección de hasta 10230 imágenes (62 clases · 15 tipografías · 11 orientaciones) de aspecto similar a la secuencia de la Figura 4.6. No obstante, como la cantidad de imágenes por clase parece insuficiente, la propuesta es repetir 5 veces el proceso de construcción aleatoria por cada carácter, para alcanzar la cifra de 51150 imágenes binarias (825 imágenes distintas por clase).

Asimismo, el clasificador a elegir debe recibir como vector de características de entrada, la imagen binaria del carácter normalizada a 14 x 20 píxeles de tamaño (280 dimensiones).



Figura 4.6: Dígitos y letras sintéticos para entrenar el clasificador de caracteres.

#### 4.2.2. Experimentos de clasificación

Análogamente, la biblioteca OpenCV permite implementar el algoritmo RANSAC que presenta el Apartado 3.2.1, a la hora de relacionar cada sucesión de caracteres alineados como palabras de interés, para realizar la lectura adecuada del texto de la escena natural, y proceder a reconocer el contenido de cada contorno de manera individual.

El problema inicial del algoritmo RANSAC es establecer de antemano el número  $K$  de iteraciones a repetir, con el fin de obtener la mejor estimación del modelo  $l_{best}$ . Por tanto, la idea es seleccionar al azar dos contornos de los disponibles para formar las respectivas líneas candidatas a evaluar. De esta manera, es posible detectar el texto tras recuperar las secuencias de contornos alineados más importantes. Respecto a la cantidad de iteraciones (**RANSAC\_ITERATIONS**) a calcular, debe ser suficiente con elegir dos contornos de la misma línea candidata óptima. Desde el punto de vista computacional, la alternativa es establecer el número  $K$  de iteraciones a partir de la cantidad de contornos restantes  $R$  (combinaciones sin repetición  $C_{R,2}$ ) que presenta la Ecuación 4.4.

$$K = \binom{R}{2} = \frac{R!}{2! \cdot (R - 2)!} \quad (4.4)$$

Respecto a cada sucesión de contornos alineados  $l_k$ , la propuesta trata de corroborar que la relación de aspecto de los caracteres asociados a la línea candidata sea similar. Así, para evitar hacer uso de los falsos positivos que han resistido el clasificador biclase de texto (caracter / no carácter) hay que cuidar determinadas características de los contornos que forman el vocabulario a recuperar. De hecho, conviene limitar la altura de cada contorno para eliminar todo carácter que supere la altura promedio de la línea  $l_k$ , tras considerar la presencia de mayúsculas de altura hasta un 60 % superior (**MAX\_HEIGHT\_RELATIONSHIP**).

A priori, a la hora de calcular el número de inliers que encajan con el modelo, sólo son de interés los contornos situados a una distancia euclídea mínima de la línea candidata  $l_k$  (**MAX\_DISTANCE\_TO\_LINE**). Además, conviene acotar la inclinación del texto  $\theta \in [-20, 20]$  para recuperar solo las palabras orientadas de manera horizontal (**MAX\_HORIZONTAL\_SLOPE**). La Figura 4.7 presenta el proceso de detección de texto, al estimar la aparición de líneas candidatas como secuencias de caracteres de altura similar alineados correctamente.

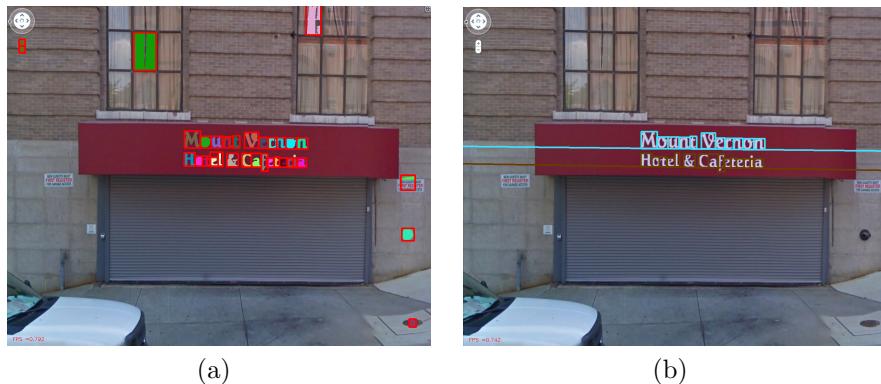


Figura 4.7: Sucesión de caracteres de altura similar en línea horizontal.

Por otra parte, para reducir el riesgo de formar líneas candidatas a partir de los falsos positivos de la escena, cabe limitar el número de caracteres por palabra a cuatro inliers mínimo por línea candidata (`MIN_CONTOURS_INLINE`) y verificar la proximidad relativa de los inliers entre sí. No obstante, la posible pérdida de dígitos o letras durante la etapa de detección, dificulta la opción de separar las distintas palabras, pues el algoritmo pretende soportar la pérdida ocasional de hasta cuatro caracteres consecutivos, de ancho similar al del resto de inliers (`MAX_CHARACTERS_IGNORED`).

Finalmente, conviene evaluar el rendimiento del algoritmo RANSAC de esta propuesta, para estimar la presencia de palabras (texto natural) mediante los caracteres a localizar de la colección Street View. Ahora, el objetivo es ejecutar el sistema actual para recuperar el posible texto de las 250 imágenes a estudiar. Por tanto, cabe identificar hasta 753 palabras de interés en total (en torno a 3 palabras por imagen de media). Así, el Cuadro 4.5 detalla la tasa de acierto de la aplicación al detectar palabras de la escena natural, donde destaca la elevada precisión, que reduce el riesgo de detectar líneas candidatas de manera errónea.

		Real	
		$\bar{T}$	T
Predict	T	$\times$	356
	T	59	397

Base de datos	Precisión	Recall	$F_2$ -Measure
Street View Text	0.87	0.52	0.57

Cuadro 4.5: Detección de palabras sobre imágenes naturales.

Respecto a la escasa sensibilidad del sistema, destaca el elevado riesgo a rechazar texto de interés, producto de la pérdida de caracteres durante la segmentación de contornos con la técnica de umbralización adaptativa. De hecho, el análisis del Apartado 2.2.4 refleja la dificultad de reconocer el vocabulario de las imágenes naturales. Análogamente, desde la propuesta de K. Wang et al. [1] mediante *Random Ferns*, no mejora el rendimiento actual, incluso tras hacer uso del léxico  $\mathcal{L}$  de 50 palabras por imagen.

Llegados a este momento, el objetivo es presentar al usuario el contenido de cada línea candidata  $l_{best}$  extraída de la escena natural. Así, la alternativa es entrenar el clasificador de caracteres (0-9, A-Z, a-z) para reconocer cada inlier de manera individual. Tras generar la base de datos con los caracteres de entrenamiento que detalla el Apartado 4.2.1, la idea es comenzar con los experimentos desde el entorno MATLAB [16], con el fin de construir el clasificador multiclasé óptimo. Como vector de características de entrada, la técnica busca reconocer la imagen binaria de cada contorno, normalizada a 14 x 20 píxeles de dimensión.

Para poder garantizar que los resultados obtenidos sean independientes de la colección de entrenamiento elegida, la propuesta es hacer uso nuevamente de la técnica de validación cruzada, que consiste en repetir cada experimento para calcular la media aritmética de las medidas de evaluación. De esta manera, con 5 experimentos diferentes (*5-fold*), es factible emplear 40920 imágenes para entrenar y 10230 imágenes para clasificar, seleccionando en cada iteración una secuencia de imágenes distinta.

Dada la alta dimensionalidad inicial del problema (280 dimensiones), como ocurría con el clasificador de texto, conviene reducir la dimensión mediante PCA, para después aplicar LDA sobre el espacio de características reducido. Por consiguiente, la transformación lineal  $A_{N \times 280}$  asociada a PCA pretende recoger el porcentaje de variabilidad adecuado, mientras LDA facilita la matriz de proyección  $A_{61 \times N}$  que trata de diferenciar las clases del problema. La Ecuación 4.5 presenta el nuevo espacio generado  $Z_{61 \times 51150}$  que separa los 62 caracteres a reconocer, al cumplir las hipótesis del LDA.

$$Z_{61 \times 51150} = LDA_{61 \times N} \cdot PCA_{N \times 280} \cdot X_{280 \times 51150} \quad (4.5)$$

Con el espacio de características reducido, será el momento de seleccionar el clasificador que permita reconocer cada carácter de interés. Así, suponiendo un completo conocimiento a priori de la estructura estadística de las 62 clases, la propuesta es hacer uso nuevamente del clasificador bayesiano, con la misma probabilidad  $P(\alpha_i)$  para cada carácter. Respecto al rendimiento que ofrece el clasificador gaussiano, oscila en torno al 95 % de acierto para la dimensión de PCA óptima ( $N = 70$ ) con 62 clases (a). La Figura 4.8 detalla el proceso de selección del parámetro  $N$ , al entrenar el clasificador bayesiano para intentar reconocer las distintas clases del problema.

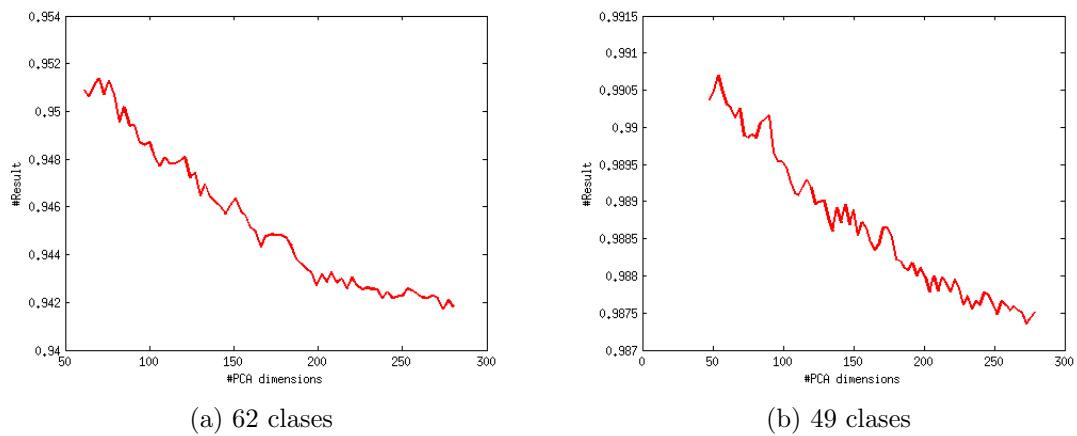


Figura 4.8: Rendimiento del clasificador bayesiano al reconocer los caracteres sintéticos.

A la hora de determinar la presencia de caracteres de apariencia similar que delimitan esa tasa de acierto del clasificador (possible confusión entre clases parecidas), la Figura 4.9 presenta la matriz de distancias asociada a los 62 caracteres de interés, para poder explicar la causa de las escasas predicciones erróneas. Basta con calcular la distancia euclídea entre las medias de cada clase  $\mu_i$  para evaluar el parecido de los caracteres entre sí.

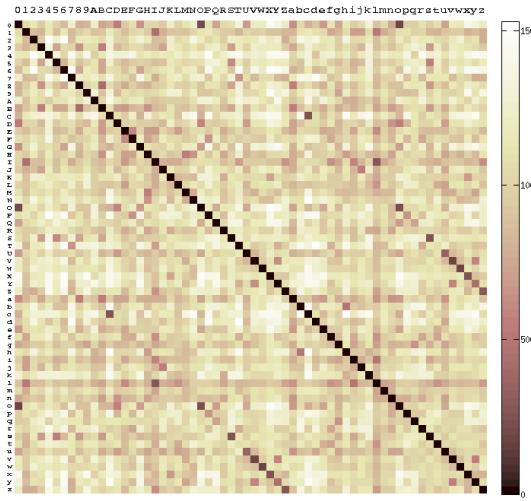


Figura 4.9: Matriz de distancias asociada al clasificador de caracteres.

Como era evidente, la distancia euclídea respectiva entre caracteres de aspecto similar es mínima [0, 250]. De hecho, para evitar aumentar la tasa de error al confundir las clases, la propuesta es renunciar a determinados caracteres: 0, c, i, j, l, o, s, u, v, w, x, y, z. Por consiguiente, tras eliminar las minúsculas que ofrecen un aspecto parecido al de la misma letra en mayúscula (véase z y Z respectivamente), será posible disminuir la dimensión del espacio generado  $Z_{48 \times 40425}$  que separa las 49 clases distintas del problema, para aumentar el rendimiento del clasificador al 99 % de acierto con la dimensión de PCA óptima ( $N = 54$ ).

Como resultado, el clasificador multiclase calcula para cada carácter natural extraído, la probabilidad a posteriori  $p(x|\alpha_i)$  relativa a cada dígito o letra que forma el alfabeto de interés (49 clases distintas), para proceder a asignar la clase más probable. La Figura 4.10 presenta al usuario de manera ordenada el texto de la escena en la parte superior derecha de cada imagen. Sin ir más lejos, al intentar reconocer la cadena **STARBUCKS COFFEE** de la escena, soportar la pérdida del carácter ‘K’ limita la opción de separar por palabras cada línea candidata con distinto léxico próximo entre sí. Análogamente, la amplia variabilidad de los caracteres y la escasa resolución de cada escena de exterior, reduce el rendimiento del clasificador bayesiano que confunde clases similares como son ‘C’ y ‘e’ (línea candidata **STARBUeSCOFFEE**).

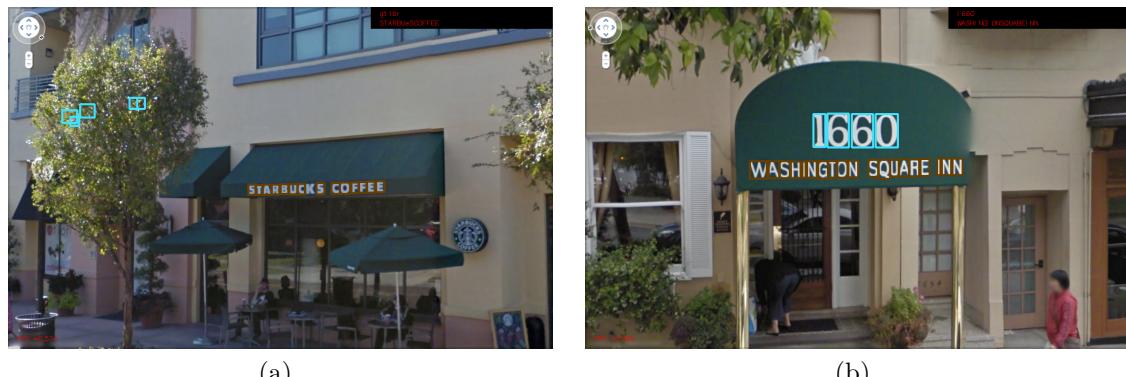


Figura 4.10: Reconocimiento de texto natural aplicado a la sucesión de caracteres.



# Capítulo 5

## Descripción informática

La descripción informática hace referencia al análisis detallado de la etapas del ciclo de desarrollo software a elegir (captura de requisitos, fase de análisis, diseño e implementación del sistema), que permitirá una evolución coherente y progresiva del problema. Por tanto, el objetivo será construir la aplicación en C++, donde aplicar las distintas técnicas a evaluar durante el Capítulo 4, ofreciendo las prestaciones adecuadas a la hora de recuperar el texto natural de la escena, para proceder al reconocimiento del contenido por palabras.

### 5.1. Metodología y plan de trabajo

El primer apartado detalla las pautas a seguir para desarrollar con éxito la aplicación de interés, tras especificar tanto la metodología a emplear para alcanzar el objetivo inicial, como el orden establecido en el plan de trabajo, para la realización del proyecto. Además, cabe destacar la estimación del tiempo necesario para realizar las tareas que propone cada etapa de desarrollo.

Así, como punto fundamental, para que el desarrollo del modelo trascurra de manera satisfactoria, es necesaria la elección de un ciclo de vida adecuado al principio del proyecto. A priori, un desarrollo en cascada resulta difícil de aplicar sobre un proyecto de carácter iterativo como es este, pues no será capaz de analizar los resultados de las primeras etapas hasta la implementación del sistema. Por consiguiente, conviene considerar hacer uso de algún ciclo de vida del software diferente que permita generar resultados intermedios para verificar el rendimiento de la aplicación durante el desarrollo, evitando rediseñar al final el sistema al completo.

El desarrollo en espiral es el modelo de ciclo de vida definido por Barry Boehm [13]. Las tareas de este modelo conforman una espiral, en la que cada bucle o iteración representa el conjunto de hasta cuatro actividades que genera cada etapa del proyecto. De esta manera, para cada iteración el objetivo debe ser: determinar los objetivos (fijar requisitos), análisis de los posibles riesgos (amenazas no deseadas), desarrollar y validar el algoritmo (pruebas) y planificación (próxima iteración). Por tanto, una vez elegido el modelo en espiral como herramienta para desarrollar el producto software, conviene detallar cuáles son las tareas del plan de trabajo.

- *Documentación previa (1 mes)*. Introducción al reconocimiento automático de texto desde escenas naturales. La idea es realizar una lectura analítica de los artículos más significativos relacionados con el tema.

Por otra parte, es la hora de construir un sistema de control de versiones o repositorio (**Subversion**) donde manejar el estado del proyecto a lo largo del tiempo. Así pues, el repositorio ayuda a almacenar todo el directorio de la aplicación con la información de los ficheros y los sucesivos cambios durante el proyecto. Respecto al entorno de desarrollo para C/C++, la idea es hacer uso del framework **Eclipse CDT Indigo** y la herramienta **CMake** para controlar el proceso de compilación del software con ficheros de configuración sencillos e independientes de la plataforma.

- *Detección del texto que aparece en la escena natural (2 meses)*. Localizar con éxito los caracteres a partir de la frontera de cada componente conexa de la imagen, para recuperar contornos de color uniforme y distinto del fondo (incluye dígitos y letras de interés). De hecho, para evitar perder regiones estables al segmentar la escena al completo, la propuesta es hacer uso de la umbralización local o adaptativa.
- *Clasificador binario carácter / no carácter (2 meses)*. Construir el clasificador para desechar la mayor parte de los contornos extraídos que no sean dígitos o letras de la escena. A la hora de generar la base de datos de entrenamiento, la idea es recuperar los distintos contornos que detecta la umbralización local, para proceder a separar las clases con éxito. Asimismo, el clasificador SVM permite separar de manera lineal las muestras (caracteres y falsos positivos), al transformar los datos a un espacio de mayor dimensión.
- *Procesamiento de cada sucesión de caracteres alineados (2 meses)*. Intentar estimar de manera robusta el texto que aparece en la escena natural, al agrupar los contornos por palabras. El algoritmo RANSAC establece aleatoriamente cada línea candidata a partir de los caracteres restantes, recuperando la presencia del vocabulario con la mayor cantidad de inliers posible.
- *Clasificador multiclasificación de caracteres (1 mes)*. Entrenar el clasificador para proceder con el reconocimiento de texto natural extraído de la escena (0-9, A-Z, a-z). Por ende, conviene simular la variabilidad asociada a los caracteres de exterior (tipografías del carácter, escasa resolución, posición del contorno, ...) para generar la base de datos sintética. Por último, al aplicar el clasificador bayesiano óptimo, destaca la intención de reducir a priori la dimensión del problema con PCA, para después aplicar LDA y maximizar el índice de separabilidad de las clases.
- *Interfaz para presentar la lectura del texto al usuario (1 mes)*. La aplicación C++ a desarrollar debe de gestionar los parámetros de entrada y salida relativos al proyecto. Así, la última etapa trata de recibir la imagen o vídeo, para procesar el contenido de cada frame en busca de texto a reconocer (por defecto emplear la posible webcam). Respecto a la salida, la aplicación presentará de manera ordenada el texto al usuario por pantalla (salida estándar) o por el fichero indicado.

## 5.2. Captura de requisitos

Para considerar adecuado el funcionamiento del sistema a desarrollar, el objetivo debe ser cumplir la lista de condiciones de la propuesta. Ahora, la idea es enumerar las reglas que establecen qué tipo de acciones debe realizar el producto deseado así como los requisitos técnicos asociados a las herramientas que implementan el software.

### 5.2.1. Requisitos funcionales

Los requisitos funcionales indican qué debe hacer el sistema para diferentes situaciones, es decir, los servicios que debe proporcionar la aplicación al usuario.

RF1. El sistema ofrece la opción `help` para aportar información al usuario acerca de los parámetros que recibe la aplicación capaz de detectar y reconocer el texto de la escena natural.

RF2. El argumento `input-file` debe permitir al usuario cargar una imagen o vídeo de entrada (dirección absoluta o relativa de un archivo) para procesar el contenido de cada frame en busca de texto a identificar. Como opción por defecto al no especificar nada, cabe recurrir a la webcam asociada al ordenador.

RF3. La aplicación debe ordenar y agrupar por palabras el vocabulario extraído para proceder a reconocer el texto, clasificando cada carácter por separado.

RF4. La salida del sistema debe presentar la lectura del vocabulario de la escena por línea de comandos (salida estándar). Con la opción `output-file` el usuario además puede almacenar el texto extraído dentro del fichero de salida a detallar (por defecto `ocr-output.txt`).

RF5. El sistema debe facilitar al usuario una opción (`show-results`) para visualizar por pantalla la ubicación de las distintas palabras a estimar, así como la predicción de los caracteres.

RF6. El usuario selecciona cuando desea mostrar los contornos. De hecho, la idea es ocultar los contornos que resistan al clasificador de texto y solo presentar los inliers de cada línea candidata establecida.

RF7. La aplicación ofrece la posibilidad de almacenar en el disco la imagen resultado con la ruta que establece el argumento `results-output-image-file`.

RF8. Al iniciar la ejecución del programa, la aplicación debe cargar automáticamente cada clasificador de interés (texto natural y caracteres sintéticos).

RF9. El usuario tiene que permanecer informado en todo momento del estado actual de la aplicación. En caso de error, será necesario precisar con claridad el origen o la causa del problema.

### 5.2.2. Requisitos no funcionales

Los requisitos no funcionales establecen la funcionalidad mínima exigida por el sistema para su correcto funcionamiento, es decir, las exigencias de cualidades impuestas a la hora de diseñar e implementar el producto software.

RNF1. La aplicación emplea la biblioteca FreeType [15] para construir la base de datos de caracteres sintéticos, transformando las imágenes vectoriales de las tipografías en imágenes de mapas de bits.

RNF2. Extender las capacidades del lenguaje de programación C++ con los recursos disponibles desde la biblioteca Boost (estructuras de datos, álgebra lineal, generación de números pseudoaleatorios, ... ).

RNF3. Distintas técnicas de la propuesta hacen uso de la biblioteca de visión artificial OpenCV 2.4.1 [14]. Por consiguiente, para ejecutar el software con éxito es necesario haber instalado OpenCV en el equipo.

RNF4. Desarrollo del sistema asociado al entorno de desarrollo **Eclipse CDT Indigo** bajo el sistema operativo **Ubuntu 12.10** (distribución Linux). No obstante, **CMake** ayuda a controlar el proceso de compilación del software con ficheros de configuración sencillos que hacen al sistema independiente de la plataforma.

## 5.3. Análisis y diseño

Realizar el desarrollo del sistema propuesto sin una planificación previa adecuada suele conducir a grandes frustraciones al percibir que finalmente el software no ha cumplido las expectativas planteadas en un principio. El etapa de análisis y diseño trata de aplicar un conjunto de técnicas y principios básicos, con la idea de definir el sistema con el suficiente nivel de detalle para proceder a su realización física. Por tanto, el objetivo será construir un software de calidad, para aplicar todos los requisitos establecidos y proporcionar una visión completa de la aplicación mediante el diagrama de clases general.

A continuación, la propuesta del trabajo es modelar los diagramas estáticos de clases de UML asociados a la aplicación. De hecho, no suele ser conveniente representar todas las clases en un mismo modelo de diseño, pues en teoría los diagramas pretenden ser simples para ayudar a describir el problema. La Figura 5.1 presenta el diagrama estático de clases centrado en la idea de cargar automáticamente cada clasificador de interés (texto natural y caracteres sintéticos) al iniciar la ejecución del programa C++. Así pues, la función *main()* asociada a la aplicación **TextRecognition** representa el punto de inicio de la ejecución del programa. Con respecto a la implementación de los clasificadores, la interfaz **Classifier** declara los distintos métodos del subsistema, sin conocer el comportamiento concreto del algoritmo (principio de abstracción).

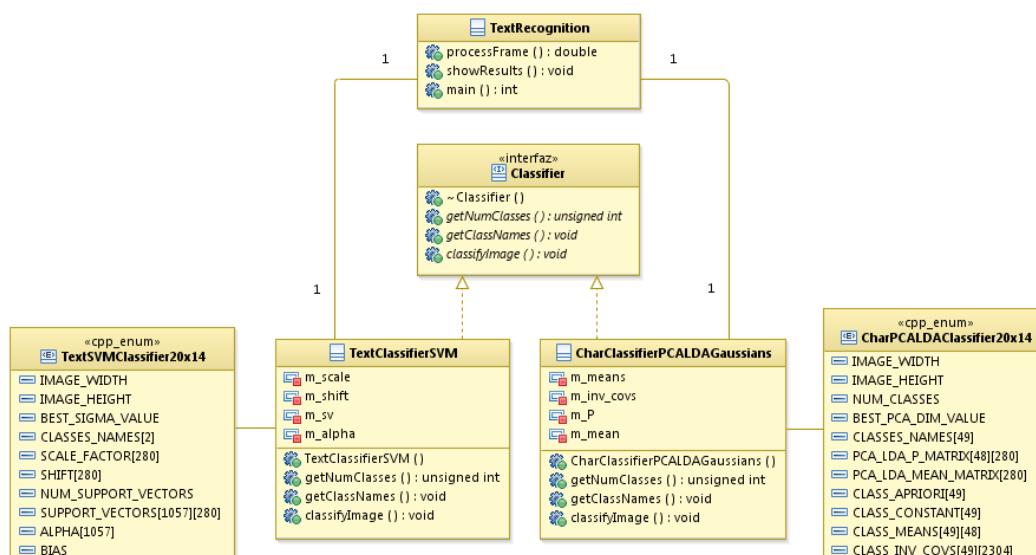


Figura 5.1: Diagrama de clases de diseño que permite construir cada clasificador.

La clase **TextClassifierSVM** especifica el comportamiento deseado para el clasificador de texto (caracter / no caracter) que permite reducir el número de contornos que genera la segmentación de la escena y conservar solamente dígitos y letras de interés. No obstante, el coste computacional asociado a generar el clasificador óptimo, limita el rendimiento de la aplicación para detectar cada carácter. De hecho, conviene evitar entrenar el clasificador en tiempo de ejecución. Como resultado, cabe almacenar desde el entorno MATLAB [16], las constantes oportunas en **TextSVMClassifier20x14**.

Análogamente, para implementar el clasificador multiclase (0-9, A-Z, a-z) que reconoce los caracteres mediante el alfabeto de interés, la clase **CharClassifierPCALDAGaussians** detalla el comportamiento del clasificador bayesiano con separación óptima entre clases. Por consiguiente, para estimar la función de densidad de probabilidad de cada clase, basta con hacer uso de la información asociada a **CharPCALDAClassifier20x14**.

Por otra parte, desde la Figura 5.2 se presenta el diagrama de clases de diseño centrado en la perspectiva de recuperar todo el texto natural que aparece e identificar su contenido. Así, la clase **InterestRegions** implementa el algoritmo de la propuesta para localizar la posición exacta de los caracteres de la escena natural (encontrar las componentes conexas de color uniforme que contrastan con el fondo), desechar los contornos que no sean letras o dígitos, y proceder a agrupar los contornos restantes por palabras. Respecto a la detección de caracteres que genera la función *detectText()*, la idea será almacenar en una estructura **ContourData** la información relativa a cada contorno extraído.

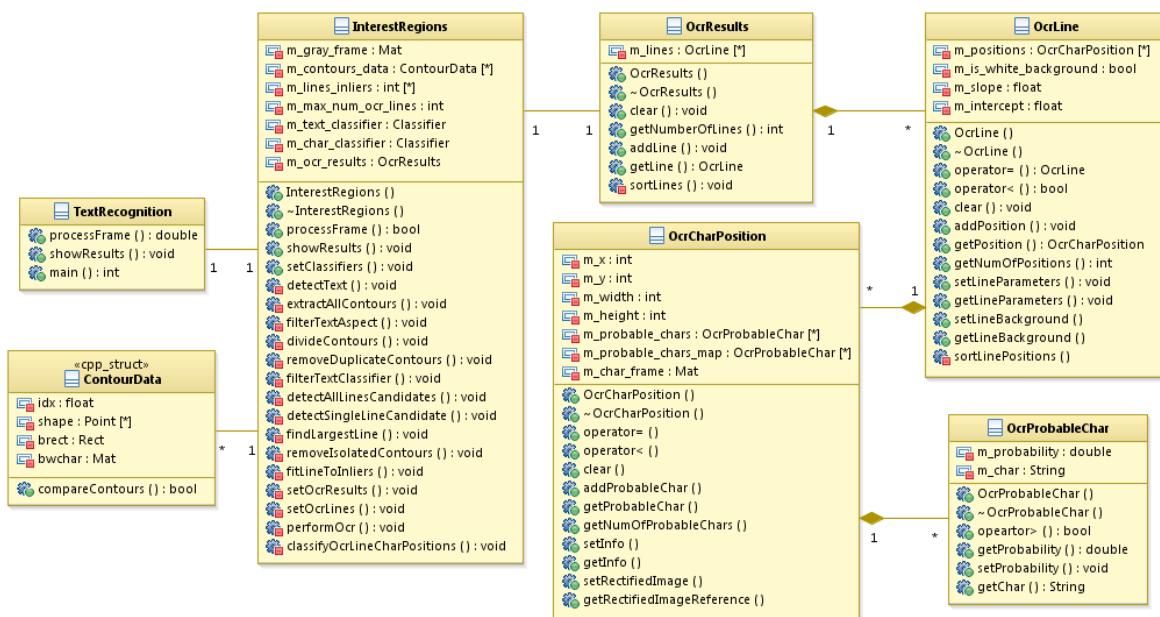


Figura 5.2: Perspectiva UML asociada a la detección y reconocimiento de texto.

La aplicación debe recuperar el posible vocabulario de la escena natural con el objetivo de reconocer y almacenar cada palabra en la clase **OcrResults**. No obstante, a la hora de estimar cada sucesión de caracteres alineados de interés (líneas candidatas) generada por la función *detectAllLinesCandidates()*, la opción es retener cada posible recta dentro de la clase **OcrLine**. Asimismo, cada línea candidata mantiene una relación de composición con la clase **OcrCharPosition**, que especifica la posición de cada inlier dentro de esa línea.

Por último, para proceder a reconocer cada carácter que forma parte de la línea candidata, el método *classifyOcrLineCharPositions()* hace uso de la clase **OcrProbableChar** para asignar al inlier el carácter del alfabeto más probable.

Para mostrar los resultados, la Figura 5.3 implementa el diagrama de diseño centrado sobre la interfaz de la aplicación, asociada a la clase **Viewer**. A fin de cuentas, el software debe permitir al usuario visualizar por pantalla tanto la ubicación exacta de los caracteres extraídos de la escena, como el vocabulario reconocido de manera ordenada. A la hora de ubicar cada contorno, basta con dibujar el rectángulo ajustado sobre la región segmentada. Con respecto al léxico a presentar al usuario, la interfaz detallará la predicción en la parte superior derecha de la imagen.

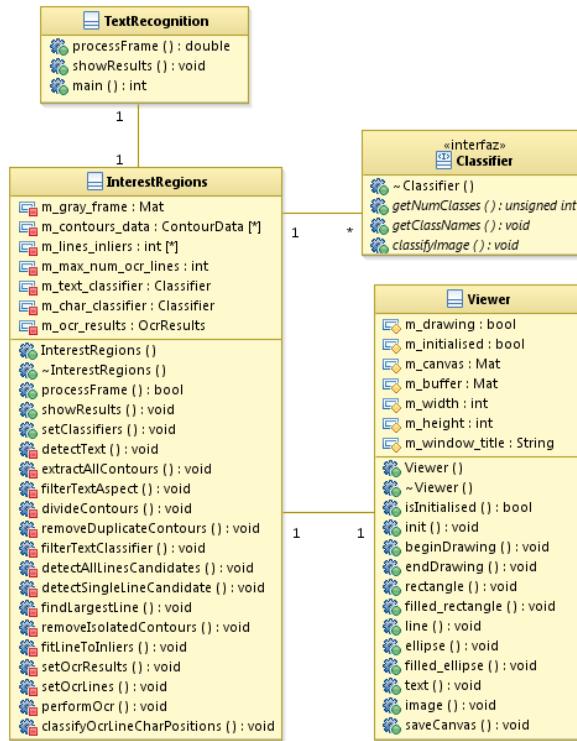


Figura 5.3: Diagrama de clases de diseño que presenta la interfaz al usuario.

Una vez conocido el diseño del sistema, conviene exponer su funcionamiento y detallar la relación entre clases en tiempo de ejecución. Así, los diagramas de secuencia a estudiar modelan la interacción entre objetos a través del tiempo en lenguaje UML. Cada diagrama de secuencia muestra, las instancias que intervienen en un escenario de interés con líneas discontinuas verticales, y los mensajes pasados entre objetos como flechas horizontales que representan las invocaciones asociadas a los métodos de cierta clase que reciba el mensaje, en orden cronológico.

La Figura 5.4 presenta el diagrama de secuencia que recoge la interacción, producto de cargar automáticamente, una única vez al inicio de la ejecución, cada clasificador a usar, desde el módulo **TextRecognition**. El método *setClassifiers()* asigna cada clasificador declarado, al atributo miembro respectivo de la clase **InterestRegions**. De esta manera, basta con implementar el método heredado *classifyImage()* para facilitar el uso correcto del clasificador.

Acto seguido, será el momento de invocar a la función *processFrame()* para recuperar el vocabulario que muestra la escena natural y identificar el contenido del texto extraído. De hecho, cuando el algoritmo sea capaz de detectar al menos una línea candidata sobre la escena (sucesión de caracteres alineados), el sistema permite visualizar por pantalla la ubicación exacta de cada inlier en el frame procesado, mediante el método *showResults()*. Además, cabe ilustrar la frecuencia a la que procesa cada fotograma la aplicación (frames por segundo) para estimar el tiempo de ejecución del algoritmo. La clase *Viewer* presenta al final con el método *text()* el valor calculado, en la parte inferior izquierda de la imagen.

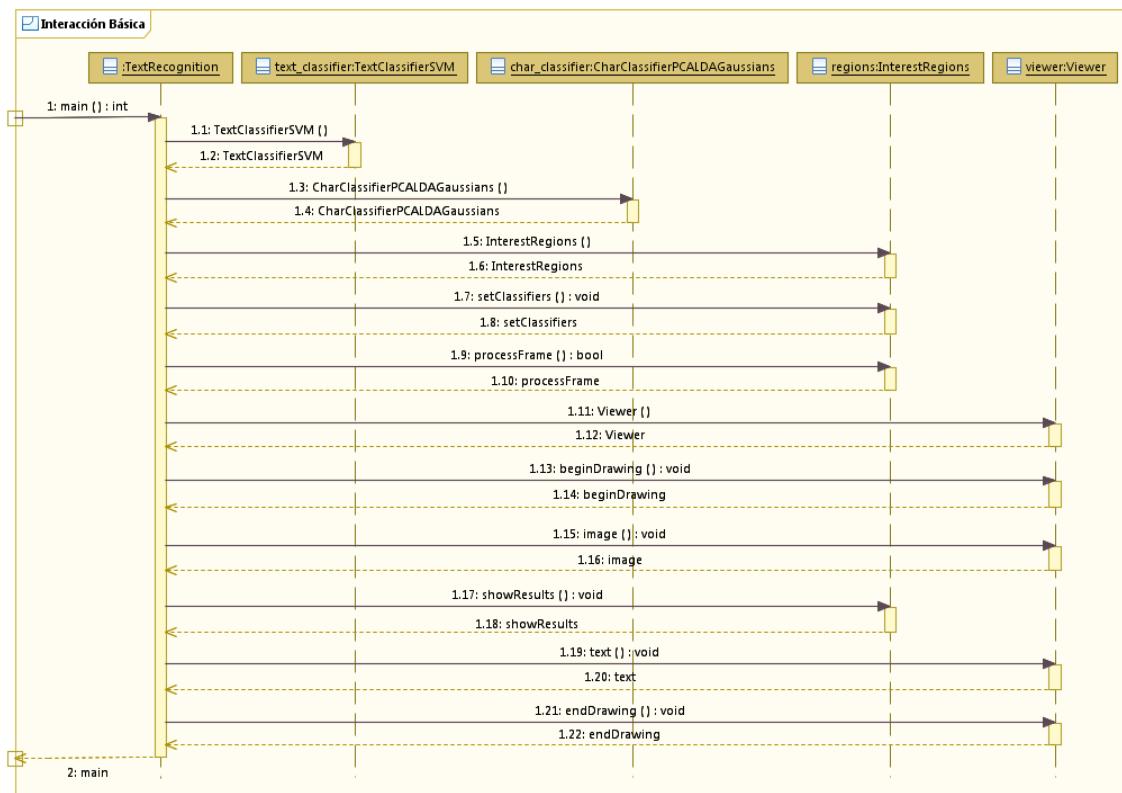


Figura 5.4: Diagrama de secuencia UML que detalla la ejecución básica del programa.

Por otra parte, el diagrama de secuencia de la Figura 5.5 presenta el aspecto dinámico o la evolución temporal, al reconocer el léxico extraído desde el módulo *InterestRegions*. Por cada sucesión estimada de caracteres candidatos similares alineados, la idea es evaluar la predicción que genera el clasificador multiclas de caracteres para cada inlier y guardar el resultado final en la clase *OcrResults*.

El método *setOcrResults()* debe generar tantas instancias para la clase *OcrLine* como líneas candidatas estima el algoritmo RANSAC, tras agrupar los contornos que resistan al clasificador biclase de texto. Al mismo tiempo, por cada línea candidata dada, la función *setOcrLines()* procede a crear tantas instancias de *OcrCharPosition* como inliers forman la recta estimada. Así, bastará con extraer el carácter más probable de cada posición, para recuperar el texto de la escena. Cuando la aplicación consiga detectar con éxito al menos una línea candidata de la escena, la idea es presentar por pantalla la posición de ese texto y la predicción del clasificador.

La clase `Viewer` establece con el método `rectangle()` la ubicación exacta de cada inlier (rectángulo ajustado al dígito o letra) sobre el frame original. Con respecto al vocabulario que genera el reconocimiento óptico de caracteres, mediante la función `filled_rectangle()` cabe delimitar una región donde presentar el texto.

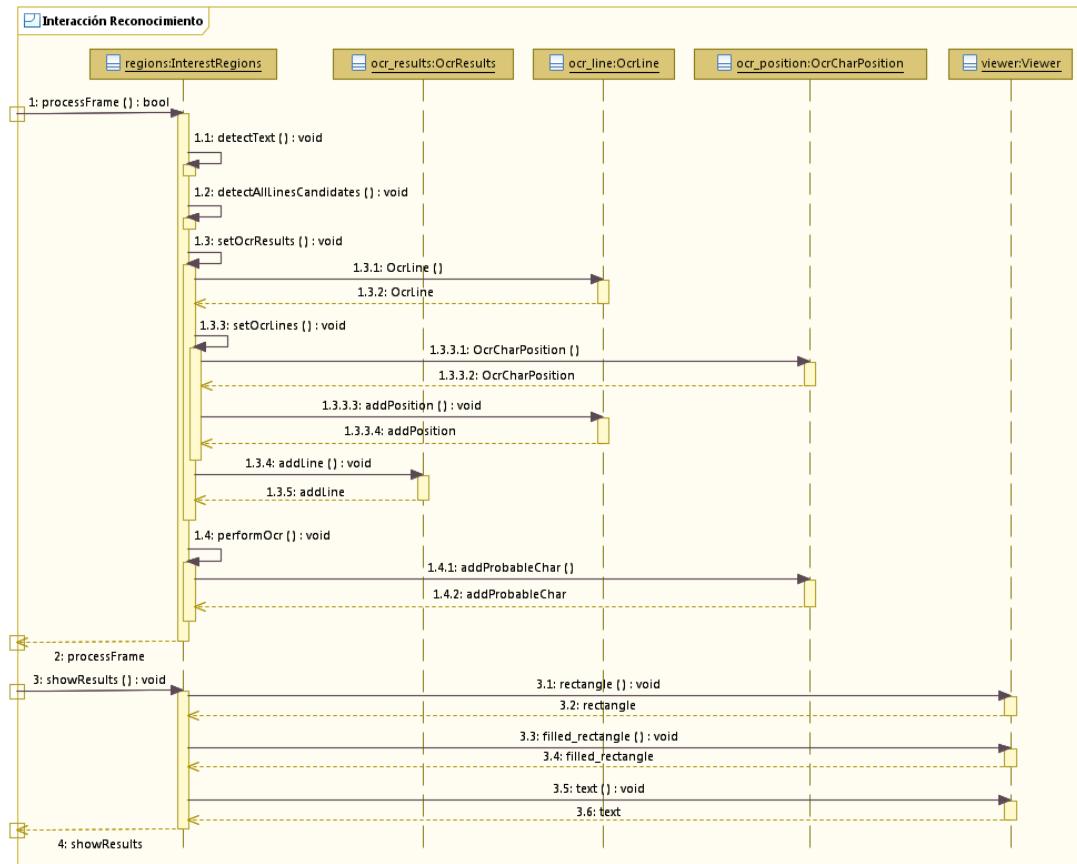


Figura 5.5: Diagrama de clases de diseño que presenta el texto en la interfaz.

A continuación, el objetivo es entrar a evaluar los distintos métodos que implementan las técnicas a estudiar durante el Capítulo 3:

- **`detectText()`.** Localizar la posición exacta de los distintos caracteres que aparecen en la escena natural.
  - **`extractAllContours()`.** Función que permitirá segmentar las fronteras de los contornos al hacer uso de la umbralización adaptativa (componentes conexas que contrastan con el fondo) tras aplicar el *kernel* y *C* adecuado, para extraer la mayor cantidad de contornos posible.
  - **`filterTextAspect()`.** Permitirá reducir la cantidad de contornos que genera la umbralización adaptativa. Por tanto, cualquier carácter a reconocer necesita superar el tamaño mínimo exigido, sin rebasar la superficie máxima establecida para el rectángulo ajustado al contorno.
  - **`divideContours()`.** Método que ofrece la opción de separar los contornos con una relación de aspecto inferior a la ideal (separar posibles caracteres pegados).
  - **`removeDuplicateContours()`.** Tras procesar repetidas veces la escena natural con los distintos umbrales, conviene eliminar los contornos de dimensión similar solapados entre sí.

- **filterTextClassifier()**. Recurrir al clasificador (caracter / no carácter) al seleccionar solamente los dígitos y letras de interés de la colección de contornos restantes.
- **detectAllLinesCandidates()**. Establecer cada sucesión de caracteres candidatos alineados posible, para generar el vocabulario de la escena.
  - **detectSingleLineCandidate()**. Comprobar la cantidad de inliers asociada a cada línea candidata extraída, para descartar líneas que no superan el mínimo necesario por palabra.
  - **findLargestLine()**. Técnica RANSAC para estimar líneas candidatas a partir de los contornos extraídos, tras escoger dos muestras de manera aleatoria para definir la recta a evaluar. El objetivo será seleccionar el modelo con más apoyo (superior cantidad de inliers) al realizar el número de iteraciones adecuado.
  - **removeIsolatedContours()**. Función que estudia la posición de los inliers en la línea candidata y comprueba la proximidad de los contornos ordenados entre sí, para evitar hacer uso de los falsos positivos que resistan al clasificador.
  - **fitLineToInliers()**. Ajustar con precisión la línea candidata estimada sobre la colección de inliers restante (no solo a las dos muestras aleatorias).
- **performOcr()**. Reconocer el contenido probable de los distintos inliers de cada línea candidata estimada, para formar el léxico de la escena.
  - **classifyOcrLineCharPositions()**. Método que usa el clasificador multiclas (0-9, A-Z, a-z) para reconocer el carácter más probable asociado a cada posición de la línea candidata (predecir el texto de la escena).

## 5.4. Implementación

Con respecto a la última etapa asociada al proceso de desarrollo software que trata la construcción del sistema (implementación del software y documentación de las decisiones), conviene detallar la línea principal de desarrollo, desde la Figura 5.6:

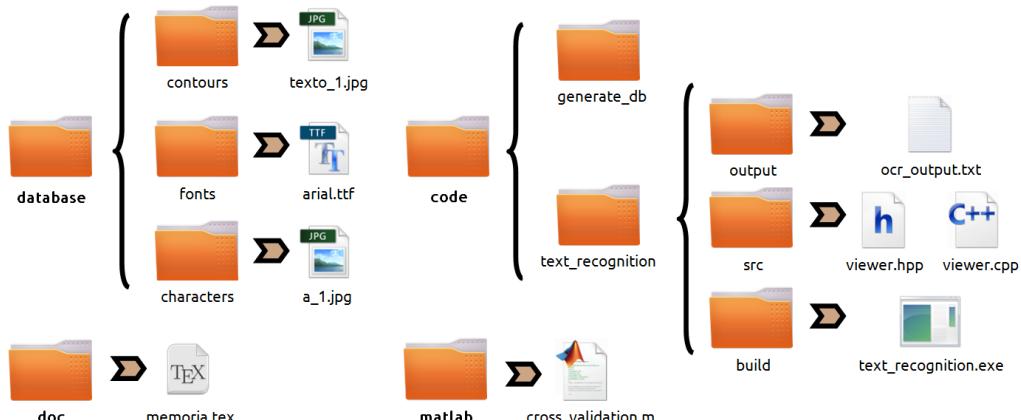


Figura 5.6: Estructura jerárquica que gestiona el contenido del producto software.

- El directorio **database** almacena las bases de datos de interés que ayudan a generar cada clasificador de la propuesta. Asimismo, la carpeta *contours* adjunta la colección de 5660 contornos extraídos con la umbralización local, para generar el clasificador biclase (caracter / no caracter) que desecha todo contorno que no sea dígito o letra. Respecto al clasificador de caracteres, la carpeta *fonts* incluye hasta 15 tipografías distintas que permiten construir la base de datos sintética que almacena el directorio *characters* (0-9, A-Z, a-z).
- La carpeta **doc** establece donde documentar el código fuente del software que detalla cada técnica de la propuesta. La memoria debe aportar suficiente información como para explicar qué hace cada módulo de la aplicación.
- El directorio **code** almacena los proyectos C++ necesarios para desarrollar el sistema. Así, la carpeta *generate\_db* hace referencia a la aplicación que genera la base de datos de caracteres sintéticos, mientras el directorio *text\_recognition* almacena el software principal que permite recuperar e identificar el texto de la escena natural.

```
./text_recognition_exe --input-file="/home/username/frame.jpg"
--results-output-image-file="/home/username/output.jpg"
```

- La carpeta **matlab** administra los distintos ficheros con extensión .m que detallan los experimentos para estimar con éxito cada clasificador desde el entorno de desarrollo MATLAB. Existen dos clases de ficheros .m a estudiar, como son las funciones y los propios scripts (sucesión de comandos a ejecutar).



# Capítulo 6

## Conclusiones

El último capítulo presenta el análisis de los resultados obtenidos durante el desarrollo del trabajo fin de máster para corroborar si el sistema cumple o no con el objetivo inicial. Asimismo, detalla la colección de problemas que conlleva hacer uso de la técnica propuesta, a la hora de detectar y reconocer el texto de cada escena natural de entrada. Para terminar, cabe destacar el posible margen de mejora de la aplicación, con la idea de progresar dentro del campo del reconocimiento óptico de caracteres.

### 6.1. Análisis de los resultados

El trabajo fin de máster estudia el problema asociado a la detección automática y el reconocimiento de texto aplicable a escenas naturales, a partir de las características de la propia imagen o vídeo de entrada. Así, a la hora de determinar la posición de cada dígito o letra de la escena por separado, la propuesta es localizar el vocabulario, en función de las características típicas de los caracteres (contornos cerrados que contrastan con el fondo). La aproximación del trabajo, basada en la segmentación de contornos mediante la técnica de umbralización adaptativa, delimita con éxito la frontera de cada componente conexa de la imagen (incluyendo los caracteres de interés). Respecto a los posibles problemas durante el desarrollo del sistema, conviene controlar el comportamiento del algoritmo para cada condición adversa específica:

- *Dimensión de la imagen.* La resolución de la cámara digital que establece el nivel de detalle en la imagen, afecta a la calidad de la segmentación (al ajustar el *kernel* y  $C$  de la umbralización adaptativa). De hecho, la detección mejora al procesar imágenes de tamaño similar al Street View (1600 x 1200 píxeles o inferior).
- *Etiquetar componentes conexas.* La segmentación basada en la extracción de regiones de color uniforme y distinto del fondo, determina la ubicación de los caracteres entre otros. Sin embargo, el algoritmo olvidará el uso de tildes (á, é, í, ó, ú), puntos (í, j), virgulilla (ñ) y diéresis (ü) colocadas sobre ciertas letras.

Para desechar regiones que no sean dígitos o letras del alfabeto latino, basta con dividir cada contorno extraído en función de la relación de aspecto ideal 20 x 14 píxeles (evitando caracteres ligados) y proceder a usar un clasificador SVM biclase (caracter / no caracter). El clasificador genera la frontera de decisión  $f(x)$  que proporciona una separación no lineal óptima entre los elementos de ambas clases, con una tasa del 95 % de acierto, reconociendo caracteres de la colección *Street View Text* del Apartado 2.2.4.

- *Complejidad temporal.* Desde el punto de vista computacional, preocupa la cantidad de regiones que extrae la umbralización, limitando el tiempo de ejecución del sistema. Así, la alternativa es reducir el número de vectores soporte que delimita la frontera de decisión del clasificador, sin pérdida de rendimiento.

La etapa de detección de texto de la escena natural por palabras, pretende agrupar los caracteres con el fin de precisar cada sucesión de contornos de dimensión similar alineados. El algoritmo RANSAC determina la formación del léxico, al estimar cada línea candidata con la máxima cantidad de inliers posible, con una tasa del 52 % de acierto, que denota la pérdida de texto producto de esa segmentación de componentes conexas sobre la colección *Street View Text*. Respecto a la implementación del algoritmo:

- *Orientación del texto.* Acotar la probable inclinación de cada secuencia de caracteres, para localizar solamente el texto horizontal de la escena.
- *Mínimo de caracteres próximos.* Cada sucesión de inliers estimada consta de mínimo 4 caracteres cercanos entre sí por línea candidata, para evitar hacer uso de los falsos positivos que resistan al clasificador de texto.
- *Sucesión de palabras alineadas.* Soportar la posible pérdida de caracteres de la misma palabra, producto de la umbralización adaptativa, impide estimar la separación entre palabras alineadas próximas entre sí.

Por último, cabe reconocer el contenido de los inliers de las líneas candidatas mediante el sistema OCR implementado con un clasificador multiclasificación de caracteres (0-9, A-Z, a-z), para presentar al usuario el léxico extraído de cada escena natural. Reducir la dimensión inicial del problema mediante la técnica PCA-LDA, permite al clasificador bayesiano (una gaussiana por clase) alcanzar una tasa del 99 % de acierto, reconociendo caracteres de la colección sintética del Apartado 4.2.1.

- *Semejanza entre clases.* La confusión entre dígitos y letras de aspecto similar, invita a descartar determinadas clases del problema, para evitar aumentar la tasa de error confundiendo clases parecidas.

Como resultado, a la hora de recuperar y clasificar las palabras de cada escena natural, destaca la dificultad en imágenes como las que presenta la Figura 6.1, donde la legibilidad de los caracteres no es la deseada (regiones conexas nítidas que contrastan con el fondo).

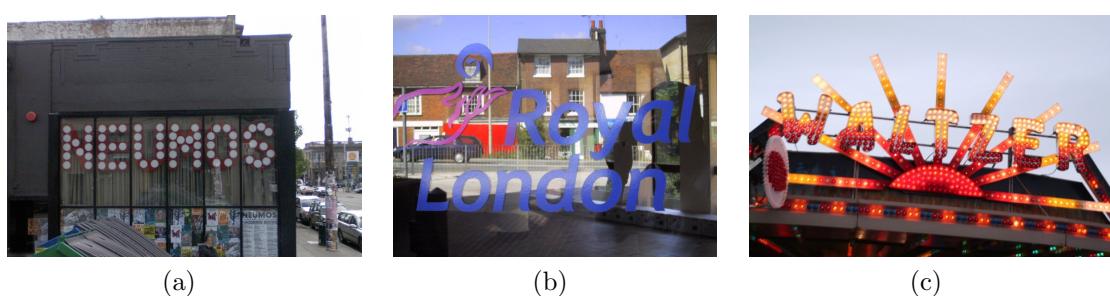


Figura 6.1: Imágenes naturales inadecuadas para evaluar la precisión del sistema.

Al comienzo del Capítulo 1 se detalla el interés que suscita el desarrollo del OCR para procesar el texto natural de cada escena, determinando la posición exacta de los caracteres y reconociendo el contenido por palabras.

Por consiguiente, cabe ilustrar el rendimiento de la aplicación C++ desarrollada, sobre las posibles aplicaciones a considerar (recuperar información, soporte a la navegación, etc). Asimismo, la Figura 6.2 especifica la elevada precisión asociada a la lectura de matrículas (1993HXC), tras acercar la cámara digital al vehículo para facilitar la segmentación óptima de los caracteres. Análogamente, será lógico recuperar con éxito el vocabulario de la señal de tráfico que consta de mínimo 4 caracteres alineados (500rn, Jaraguas, Fuenterrubles).



Figura 6.2: Distintas aplicaciones a considerar al hacer uso de la aplicación.

## 6.2. Trabajos futuros

Llama la atención el posible margen de mejora de la aplicación, con la idea de progresar dentro del campo del reconocimiento de texto para imágenes naturales. Las características del problema y las distintas vertientes que presenta, hacen que sea un proceso en continua investigación. No obstante, hay aspectos de este trabajo que conviene estudiar en el futuro para complementar el sistema:

- *Componentes conexas.* La variabilidad asociada al texto de la escena natural dificulta la segmentación al tratar encontrar el valor adecuado para `kernel_value` y `c_value`. Toma importancia repetir el algoritmo de umbralización adaptativa sin incrementar el coste temporal, cambiando el valor de estos parámetros para evitar la pérdida de caracteres de interés.

- *Vector de características.* La desmedida cantidad de contornos segmentados, obliga a construir el clasificador biclase (caracter / no caracter) para conservar los caracteres. Para reconocer la imagen de 14 x 20 píxeles de cada contorno, la opción es prescindir del espacio de 280 dimensiones y emplear algún descriptor de características visuales (invariante a una posible rotación, traslación o cambio de escala en el carácter) como HOG (*Histograms of Oriented Gradients*), momentos de Hu, etc.
- *Coste del clasificador SVM.* Tras cambiar el vector de características, encaja repasar la propuesta de A. Martínez et al. [12], para facilitar la separación de ambas clases al hacer la proyección mediante SDA. Otra alternativa, será implementar el clasificador óptimo directamente sobre la GPU para acelerar así el cálculo de los vectores soporte.
- *Lectura de palabras con sentido.* Recuperar todo el léxico con sentido de cada escena (sin pérdida de caracteres o errores al clasificar). Sin embargo, el algoritmo RANSAC de la propuesta identifica el contenido de los caracteres de manera individual, dentro de sucesiones de caracteres donde suele haber pérdida de inliers o errores al clasificar. Con el apoyo del léxico  $\mathcal{L}$  es posible estimar el vocabulario con acierto.
- *Soporte para móviles.* Hoy prevalece el impacto social de dispositivos móviles dotados de cámaras digitales de última generación. Así, crece el interés que suscita desarrollar la aplicación para **Android** o **iOS**, mediante el entorno OpenCV [14].



# Bibliografía

- [1] Kai Wang, Boris Babenko y Serge Belongie, *End-to-End Scene Text Recognition*, International Conference on Computer Vision (ICCV), pp. 1457-1464 (2011).
- [2] Boris Epshtain, Eyal Ofek y Yonatan Wexler, *Detecting Text in Natural Scenes with Stroke Width Transform*, Computer Vision and Pattern Recognition (CVPR), pp. 2963-2970 (2010).
- [3] Teófilo E. de Campos, Bodla R. Babu y Manik Varma, *Character Recognition in Natural Images*, International Conference on Computer Vision Theory and Applications (VISAPP), pp. 273-280 (2009).
- [4] Simon M. Lucas, *Text Locating Competition Results*, European Conference on Computer Vision (ECCV), pp. 591-604 (2010).
- [5] Kai Wang y Serge Belongie, *Word spotting in the wild*, International Conference on Computer Vision (ICCV), pp. 1457-1464 (2011).
- [6] Lucas Neumann y Jiri Matas, *A method for text localization and recognition in real-world images*, Asian Conference on Computer Vision (ACCV), pp. 770-783 (2010).
- [7] Mustafa Özuysal, Pascal Fua y Vincent Lepetit, *Fast keypoint recognition in ten lines of code*, Computer Vision and Pattern Recognition (CVPR), 2007.
- [8] Jiri Matas, Ondrej Chum, Martin Urban y Tomás Pajdla, *Robust wide-baseline stereo from maximally stable extremal regions*, In British Machine Vision Conference (BMVC), pp. 384-393 (2002).
- [9] Satoshi Suzuki y Keichi Abe, *Topological structural analysis of digitized binary images by border following*, Computer Vision, Graphics and Image Processing (GVGIP), pp. 32-46 (1985).

- [10] Vladimir N. Vapnik, *The nature of statistical learning theory*, Springer, New York, ISBN 0-387-94559-8 (1995).
- [11] Martin A. Fischler y Robert C. Bolles, *Random Sample Consensus: a paradigm for model fitting with applications to image analysis and automated cartography*, Communications of the ACM, pp. 381-395 (1981).
- [12] Manli Zhu y Aleix M. Martínez, *Subclass Discriminant Analysis*, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), pp. 1274-1286 (2006).
- [13] Barry Boehm, *A Spiral Model of Software Development and Enhancement*, ACM SIGSOFT Software Engineering Notes (ACM), pp. 14-24 (1986).
- [14] Gary Bradski, *The OpenCV library*, Dr. Dobb's Journal of Software Tools (2000).
- [15] David Turner, *The design of FreeType 2*, The FreeType Development Team (2010).
- [16] MATLAB version R2012a, Natick, Massachusetts, The MathWorks Inc. (2012).