



**Universidad
Rey Juan Carlos**

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Curso Académico 2023/2024

Práctica 1

DETECCIÓN DE SEÑALES VIALES

Cristian Fernando Calva Troya

Luis Ovejero Martín

Jaime Rueda Carpintero

Índice de figuras

1.	Corrección de angulo de la imagen	4
2.	Imagen tras aplicar filtro azul	5
3.	Imagen tras aplicar sobel	5
4.	Imagen tras detectar lineas y ángulos de desfase	6
5.	Corrección de angulo de imagen	6
6.	Imagen tras aplicar MSER y dibujar las regiones detectadas	7
7.	Imagen tras dibujar un rectángulo en cada región detectada	8
8.	Imagen tras filtrar los rectángulos del paso anterior.	10
9.	Imagen tras ampliar los rectángulos del paso anterior.	11
10.	Subpaneles detectados en la imagen 6.	12
11.	Panel ideal	12
12.	Panel ideal redimensionado y filtrado por color	13
13.	Panel detectado redimensionado y filtrado por color	13

Índice

1.	Introducción	4
2.	Objetivos y metodología	4
2.1.	Descripción del problema	4
2.2.	Objetivos	4
2.3.	Tecnologías	4
3.	Creación de la aplicación	4
3.1.	Normalización	4
3.1.1.	Corrección de color	4
3.1.2.	Corrección de ángulo	4
3.2.	Detección de regiones de alto contraste	6
3.2.1.	Aplicar el filtro de azul	6
3.2.2.	MSER	6
3.2.3.	Extraer los píxeles de la región en un rectángulo	8
3.2.4.	Filtrar rectángulos por relación de aspecto	8
3.2.5.	Expandir regiones filtradas	9
3.2.6.	Detección de subpaneles dentro de la región del cartel	11
3.2.7.	Coordenadas de los subpaneles en la imagen original	12
3.3.	Correlación por máscaras	12
3.3.1.	Mascara ideal	12
3.3.2.	Filtrado por correlación ideal/detectado	13
3.4.	Eliminación de regiones repetidas	14
3.5.	Validación	14
3.6.	14
4.	Conclusiones y trabajos futuros	14
5.	Referencias	14

1. Introducción

2. Objetivos y metodología

2.1. Descripción del problema

2.2. Objetivos

3 pag.

2.3. Tecnologías

3. Creación de la aplicación

3.1. Normalización

3.1.1. Corrección de color

3.1.2. Corrección de ángulo



Figura 1: Corrección de angulo de la imagen

Otro de los problemas presentes a la hora de procesar las imágenes, para su detección, es la posición de los objetos a detectar. En ocasiones, ya sea por el ángulo de la cámara o la misma posición del cartel respecto a nosotros, puede ocurrir un cierto desfase en el angulo deseado. En la **Fig. 1** podemos observar un pequeño ejemplo del problema inicial y el resultado deseado, que nos dará una mejor detección de las secciones. Para poder abarcar el problema hemos optado por realizar una corrección a la imagen completa.

Los pasos a seguir para conseguir el angulo de desfase son los siguientes:

1. Aplicamos una detección de color azul ya que respecto a las diferentes pruebas, ha dado mejores resultados para eliminar las zonas no deseadas y poder centrarnos en las secciones más azules (los paneles deseados.) En la **Fig. x** podemos ver un ejemplo de los posibles resultados.



Figura 2: Imagen tras aplicar filtro azul

2. Realizamos una detección de bordes horizontales. Ya que haremos la corrección en función de como estén estos orientados. En la **Fig. x** se ve que conseguimos las secciones deseadas.

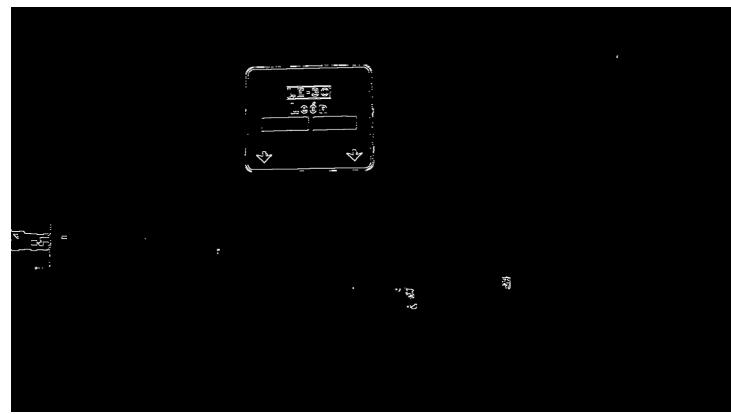


Figura 3: Imagen tras aplicar sobel

3. Finalmente con el algoritmo de Hough extraemos líneas presentes en la imagen, dando como resultado las rectas horizontales. En la **Fig. x** se visualizan las rectas detectadas, y se calcula el angulo de cada uno de ellas haciendo uso de las

coordenadas (x, y) y $(x2, y2)$, de estas nos quedaremos con un desfase intermedio. Finalmente con el ángulo de cada imagen, corregimos el giro sobre la imagen original, tal y como se muestra en la **Fig. x**.

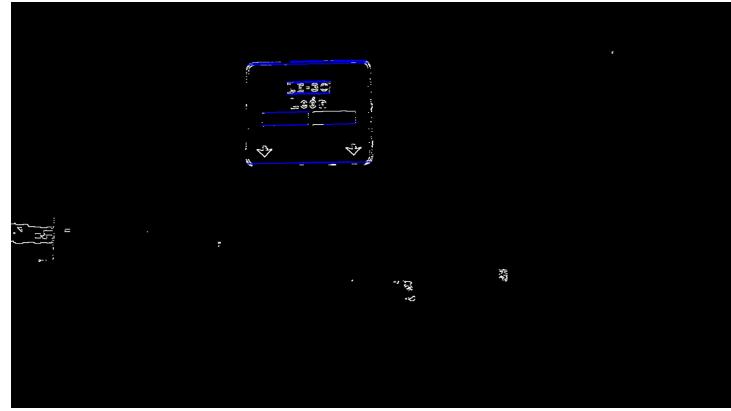


Figura 4: Imagen tras detectar lineas y ángulos de desfase

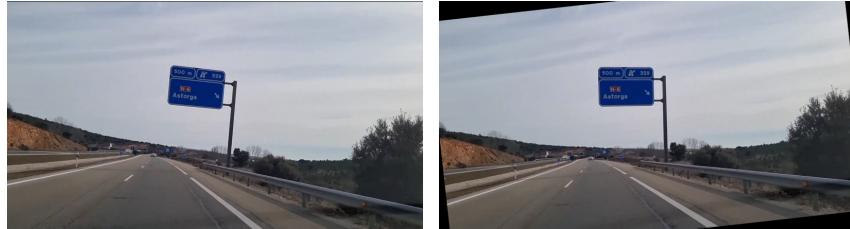


Figura 5: Corrección de angulo de imagen

3.2. Detección de regiones de alto contraste

3.2.1. Aplicar el filtro de azul

Antes de empezar a detectar regiones, vamos a aplicar el filtro de azul sobre las imágenes normalizadas. Esto lo hacemos con la idea de mejorar la detección de los carteles, ya que al aplicar el filtro de azul podemos determinar de forma más eficaz la zona de imagen que puede pertenecer a un cartel.

3.2.2. MSER

Una vez hemos aplicado este filtro a las imágenes ya podemos empezar a detectar regiones sobre las máscaras que hemos generado con el paso anterior.

Para detectar las regiones de alto contraste definimos la función *mser*. Esta recibirá la lista de imágenes normalizadas y la lista de máscaras que hemos creado al aplicar el filtro de azul.

Crearemos dos listas que nos servirán para guardar aquellas imágenes sobre las que vamos a dibujar las regiones que detectemos y la otra para guardar las regiones detectadas.

La función va a iterar sobre la lista de imágenes y creará una copia de la imagen normalizada para dibujas las regiones y una copia de la máscara sobre la que aplicará *MSER*. Antes de aplicar el algoritmo dilatamos la mascara y restamos la máscara original a la máscara dilatada para obtener una imagen de bordes. Tras esto creamos un *MSER* y ajustamos los parámetros para filtrar un poco las regiones que pueda detectar, por ejemplo, haciendo que el área mínima de las regiones que detecte sea 900 y que la máxima sea 5000.

Las regiones detectadas en una imagen se guardaran en la variable llamada *polygons*. Sobre esa lista iteraremos a continuación y para cada elemento de la lista aplicaremos la función *convexHull* de *openCV*, con la finalidad de hallar el casco convexo de un conjunto de puntos. Esto quiere decir que la usaremos para determinar el perímetro de la región que hemos detectado. Guardaremos estos perímetros en la lista *hulls*, que usaremos más adelante para poder dibujar las regiones en la imagen. También guardamos la lista de regiones detectadas en *detected_regions* y finalmente dibujaremos las regiones sobre la copia de la imagen y la guardaremos en la lista que hemos creado al principio para ello. Por último mostramos la primera y última imagen con las regiones dibujadas y devolvemos las listas de las imágenes con las regiones dibujadas y la lista de las regiones detectadas.

Tras ejecutar la función deberían aparecer las imágenes que podemos ver en la **Fig. x**

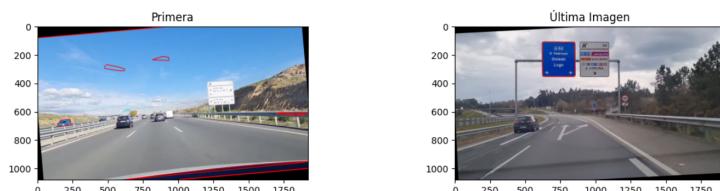


Figura 6: Imagen tras aplicar MSER y dibujar las regiones detectadas

En la siguiente celda del *notebook* vemos un bloque de código con un bucle *for* que está comentado, lo podemos descomentar y ejecutar para ver el resultado de nuestra función sobre el resto de imágenes de test. Para pasar las fotos simplemente vamos pulsando cualquier tecla. En caso de querer parar tenemos que pulsar el cuadrado de arriba a la izquierda y cerrar la ventana para terminar la ejecución.

3.2.3. Extraer los píxeles de la región en un rectángulo

Para este paso vamos a definir la función `rectangle_of_regions` que tiene como objetivo dibujar, usando `cv2.boundingRect`, un rectángulo alrededor de las regiones detectadas en el paso anterior. La función recibirá la lista de imágenes normalizadas y la lista de las regiones detectadas en el paso anterior. Se creará una lista en la que se añadirán las imágenes con los rectángulos dibujados. Iteraremos sobre las imágenes de test y crearemos una copia de cada imagen. Ahora iteraremos sobre las regiones de la imagen sobre la que estamos trabajando y usaremos `cv2.boundingRect`, para obtener el punto superior izquierdo de la región y el ancho y alto de la misma y con ello dibujaremos el rectángulo con `cv2.rectangle`.

Tras dibujar las regiones añadimos la imagen a la lista, imprimimos la primera y la última de estas imágenes y devolvemos la lista donde las hemos guardado.

Al igual que en la región anterior, tenemos una celda de código comentado que podremos usar para poder ver el resultado en el resto de imágenes.

Un ejemplo de los resultados que podemos ver en estas imágenes resultado lo vemos en la Fig.

x



Figura 7: Imagen tras dibujar un rectángulo en cada región detectada

3.2.4. Filtrar rectángulos por relación de aspecto

En este apartado se pide que filtremos los rectángulos de las regiones, que hemos dibujado en el paso anterior, por su relación de aspecto con la finalidad de eliminar aquellos que disten mucho de la relación de aspecto de los carteles que se pretenden detectar.

Además de esto hemos hecho que la función encargada de realizar esto también elimine los rectángulos contenidos dentro de otros. Esto lo hacemos porque queremos solo detectar la región

del panel azul, no los subpaneles. La idea es encontrar los subpaneles más adelante, pero solo sobre la imagen del cartel en lugar de hacerlo en la imagen completa, así podemos eliminar detecciones basura que pueda llegar a hacer en la imagen completa, pero de esto ya hablaremos próximamente.

Crearemos dos listas, una para las imágenes con las regiones filtradas dibujadas y otra para las regiones filtradas.

Luego iteraremos sobre las imágenes, para cada una de estas vamos a crear una lista para guardar las regiones filtradas de esta imagen y copiaremos la imagen para poder dibujar las regiones en la copia así como hemos anteriormente.

Ahora iteraremos sobre la lista de regiones que pertenecen a esta imagen. Para cada región que hay en la lista vamos a usar *cv2.boundingRect* para obtener el ancho y el alto, con esto podremos calcular la relación de aspecto de las regiones. Nos vamos a quedar con las regiones que tengan una relación de aspecto (ancho/alto) entre 0.6 y 6.5.

Con las regiones que cumplan esto vamos a hacer la comprobación de si están dentro de otra, en caso de que se cumpla y esté dentro de otra, no nos vamos a quedar con ella, solo queremos quedarnos con las regiones más exteriores con la idea de quedarnos con la región del cartel completo, ya que como hemos dicho antes los subpaneles los buscaremos luego.

Iremos agregando las regiones que cumplen ambas condiciones a la lista *filtered_regions* y dibujando las mismas en sus imágenes correspondientes, estas las añadiremos también a *images_filtered_rectangles*.

Finalmente mostraremos la primera ya la última imagen ya que en estas vemos como se eliminan algunas regiones que no cumplen con las exigencias que queremos. Por último vamos a devolver la lista de imágenes y de regiones filtradas.

Al igual que en los pasos anteriores tenemos una celda comentada para poder ver el resultado en el resto de imágenes.

Podemos ver el resultado sobre la **Fig. x** anterior y la **Fig. x** que tenemos a continuación.

3.2.5. Expandir regiones filtradas

Al igual que se explica en el enunciado, es posible que haya regiones, sobre las que hemos dibujado un rectángulo, en las que nos haya quedado parte del cartel fuera del rectángulo y para arreglar este pequeño problema, que puede empeorar los resultados en pasos posteriores, tenemos que aumentar el tamaño del rectángulo.



Figura 8: Imagen tras filtrar los rectángulos del paso anterior.

Para solucionar esto tenemos la función *expand_regions*. Al igual que las funciones anteriores recibirá dos listas. Una de las imágenes normalizadas y la otra de las regiones filtradas en el paso anterior.

La función irá iterando sobre las imágenes de la lista que recibe como primer argumento. Para cada imagen iterará sobre sus regiones filtradas y hallará el punto superior izquierdo (valores *x* e *y*), el ancho y el alto de la región. Una vez tiene estos valores en el caso de los valores de *x* e *y* restaremos un valor para que se muevan más hacia arriba a la izquierda. En el caso de la altura y anchura sumaremos un valor porque queremos que sean más grandes.

Dibujaremos el rectángulo de nuevas dimensiones sobre la copia de la imagen normalizada que hemos hecho previamente. Después de eso vamos a crear una pequeña imagen de la región, llamada *pixels_region*. También vamos a guardar en una lista de posibles paneles (*possible_panels_img*) este recorte de la imagen normalizada en la que podría haber un panel. Además guardaremos en forma de tupla las coordenadas de cada región ampliada para usarlas más adelante.

Al igual que hemos hecho en los pasos anteriores vamos a mostrar el resultado de la primera y última imagen y se devolverá la lista de imágenes con las regiones ampliadas dibujadas, otra lista para las coordenadas nuevas de las regiones y la lista de posibles subpaneles.

Al igual que en los apartados anteriores podemos ver el resultado del resto de imágenes al ejecutar la celda de código siguiente.

Un ejemplo sería el que podemos ver en la **Fig. x**



Figura 9: Imagen tras ampliar los rectángulos del paso anterior.

3.2.6. Detección de subpaneles dentro de la región del cartel

El proceso de filtrado de las regiones detectadas tenía como objetivo llegar a este punto. Según el enunciado de la práctica, esta parte no se pedía. Nuestra idea era trabajar solo con las regiones que podrían ser carteles a la hora de buscar los subpaneles, ya que pensamos que al haber menos elementos en la imagen seríamos capaces de localizar los paneles de manera más efectiva.

Dentro de este apartado vamos a volver a ejecutar acciones parecidas a las que hemos usado ya, pero en este caso lo haremos sobre los recortes de las imágenes que hemos determinado que pueden ser carteles.

En la función *detect_subpanels* vamos a recibir las imágenes normalizadas y la lista de los recortes de cada imagen. Lo que haremos será, imagen por imagen, volver a usar *mser* para detectar las regiones de los subpaneles. Estas imágenes las vamos a redimensionar con *cv2.resize* para hacerlas más grandes. Les aplicaremos un filtro de azul para binarizarlas y usaremos *cv2.Canny* para hallar la imagen de bordes. La dilataremos para no perder detalles en los bordes y aplicaremos *mser* para detectar los subpaneles. Posteriormente a detectar las regiones vamos a ampliarlas y guardar sus coordenadas.

Finalmente la función va a devolver una lista de recortes de cada región detectada anteriormente, es decir las subregiones dentro de las regiones previas. Estos recortes son posibles subpaneles de las imágenes (*possible_subpanels*). También vamos a devolver las coordenadas de estas subregiones en la lista *subpanel_coords*.

En la **Fig. x** podemos ver las subregiones detectadas en este paso en la imagen 6.



Figura 10: Subpaneles detectados en la imagen 6.

3.2.7. Coordenadas de los subpaneles en la imagen original

Este es el último paso antes de pasar a la correlación de máscaras. En este paso vamos a corregir las coordenadas que acabamos de obtener de los subpaneles. Esto lo hacemos porque al aumentar el tamaño de la región sobre la que hemos detectado las subregiones, las coordenadas de la subregión también se ven afectadas por el cambio. Para ello tenemos que corregir las coordenadas en el sentido inverso en el que habíamos aumentado la imagen. En el paso anterior hemos multiplicado por tres el ancho y el alto de la imagen y ahora tendremos que dividir por tres.

Para esto tenemos la función *fix_coords* de manera similar a como lo hemos ido haciendo anteriormente obtendremos las coordenadas de cada posible subpanel detectado en cada una de las imágenes. Una vez tenemos las coordenadas las ajustamos y las volvemos a guardar en una lista que vamos a devolver.

3.3. Correlación por máscaras

3.3.1. Mascara ideal



Figura 11: Panel ideal

Para la correcta comprobación de que si una imagen ya filtrada por color es un panel deseado, necesitamos un panel de referencia, para poder indicar como debería ser un panel detectado de forma aproximada.

En la **Fig. 1** podemos ver la imagen de la que partimos para realizar esa mascara ideal que utilizaremos posteriormente. Esta imagen inicial se extrae de la Norma 8.1-IC de señalización

vertical de carreteras en España [1].

Para poder crear la matriz de 1's (para azules) y 0's (para no azules) hacemos uso de un filtro de color. Antes de poder hacer uso del filtro de color necesitamos cambiar el formato de la imagen de **BGR** a **HSV**. Este cambio es fundamental porque en el espacio de color HSV, el componente Hue permite una separación clara del color azul, independientemente de las variaciones de iluminación y sombra. Esta conversión y filtrado mejoran significativamente el procesamiento y permiten realizar comparaciones directas con un panel de referencia, facilitando la verificación.

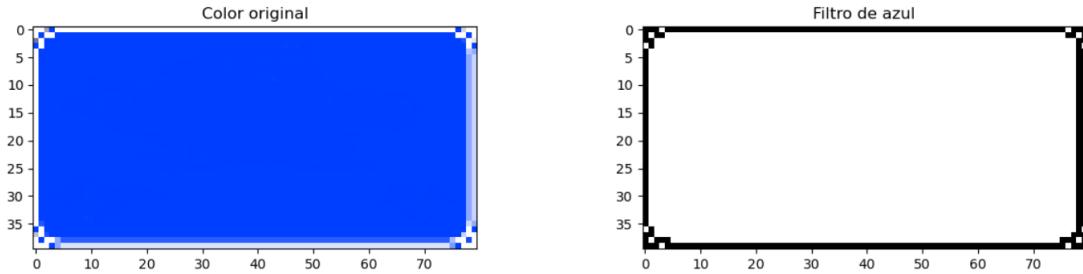


Figura 12: Panel ideal redimensionado y filtrado por color

Ya que nos encontraremos con diferentes paneles de diferentes tamaños, establecemos una tamaño estandar de (80x40) a redimensionar para todos los paneles, tanto detectados como para el ideal. Tras aplicar esta redimensión podemos ya aplicar el filtro de color a nuestro panel, en la **Fig. 2** podemos ver el resultado de este proceso con el panel ideal.

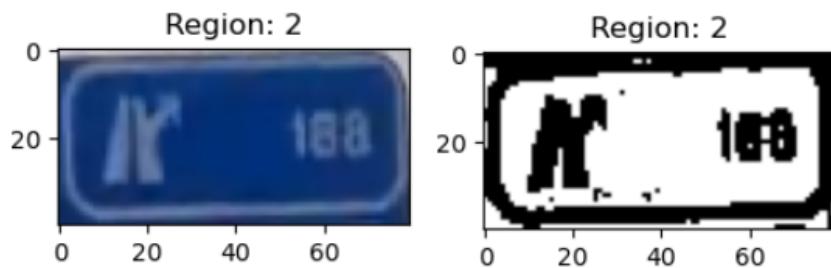


Figura 13: Panel detectado redimensionado y filtrado por color

Respecto a los paneles detectados en la **Fig. 3** podemos ver un ejemplo de su proceso de redimensionado y filtrado por color. De esta manera ya podríamos realizar una comparación y correlación entre detectado/ideal.

3.3.2. Filtrado por correlación ideal/detectado

Llegados a este punto quedaría decidir que paneles recibidos son lo suficientemente parecidos al panel ideal. Para ellos realizamos:

1. Una suma de cuantos píxeles azules tiene nuestra máscara ideal. Teniendo en cuenta que es una máscara de 0's y 1's queda bajo la siguiente operación:

$$B = \sum_{i=0}^{39} \sum_{j=0}^{79} I(i, j) \quad (1)$$

2. Calcular la cantidad de azules que tiene nuestra detección. De igual forma que en el paso anterior:

$$B' = \sum_{i=0}^{39} \sum_{j=0}^{79} I'(i, j) \quad (2)$$

3. Calcular un porcentaje de correlación normalizado entre [0,1]. Simplemente haciendo una proporción de B' sobre B , en caso de que B' supere a B descartamos esta detección poniendo su proporción a 0. Quedando:

$$C = \begin{cases} B'/B & \text{si } B' \leq B, \\ 0 & \text{si } B' > B. \end{cases} \quad (3)$$

4. Descartamos aquellos valores C que superen un umbral. En este caso se ha puesto uno mínimo de 0'4.

Posterior a estos pasos nos quedarían las secciones más fiables a la detección deseada, pero nos quedaría eliminar posibles detecciones repetidas.

3.4. Eliminación de regiones repetidas

3.5. Validación

3.6. ...

4. Conclusiones y trabajos futuros

3 pag.

5. Referencias

[1] <https://www.fomento.gob.es/az.bbmf.web/documentacion/pdf/re3723.pdf>