_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

AYUDA A LA MOVILIDAD DE PERSONAS INVIDENTES BASADO EN VISIÓN ARTIFICIAL

Miguel Saura Herreros

Tutor: Antonio José Albiol Colomer

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2016-17

Valencia, 12 de septiembre de 2017

Resumen

Esta investigación se basa en el uso de la visión artificial mediante una aplicación desarrollada en C++ con las librerías para tratamiento de imagen CImg y OpenCV, además del motor Tesseract OCR para el reconocimiento de caracteres en una imagen.

Partimos de un problema de movilidad, el cual se intenta solucionar utilizando las herramientas aprendidas durante el Grado, que nos serán de utilidad para desarrollar la aplicación. Se plantearon varias técnicas de reconocimiento de formas para identificar las puertas de metro, como, por ejemplo, la detección de las esquinas o de los márgenes negros de las puertas.

Tras analizar las imágenes tomadas de los diferentes tipos de metros de las líneas de metro de Valencia, se llegó a la conclusión de que las formas de las puertas variaban dependiendo en la forma de los cristales de las puertas.

Sin embargo, siempre se encuentra el mismo letrero encima de cada puerta. De esta manera, la aplicación se basa en localizar el letrero y después, detectar si la puerta está abierta o cerrada mediante el reconocimiento de formas.

Resum

Aquesta investigació es basa en l'ús de la visió artificial utilitzant una aplicació desenvolupada amb C++ amb les llibreries per a tractament d'imatge CImg i OpenCV, a més del motor Tesseract OCR per al reconeixement de caràcters en una imatge.

Partim d'un problema de mobilitat que s'intenta solucionar utilitzant les ferramentes apreses durant el Grau, que ens seran d'utilitat per desenvolupar l'aplicació. Es plantejaren diverses tècniques de reconeixement de formes per identificar les portes del metro, com per exemple, la detecció dels cantons o dels marges negres de les portes.

En analitzar les imatges captades dels diferents tipus de metros de les línies de metro de Valencia, s'arribà a la conclusió de que les formes de les portes variaven depenent de la forma dels cristalls de les portes.

No obstant això, sempre es pot trobar el mateix cartell dalt de cada porta. D'aquesta manera, l'aplicació es basa en localitzar el cartell y després, detecta si la porta es troba oberta o tancada mitjançant el reconeixement de formes.

Abstract

This research is based on the use of computer vision using an application developed in C++ with CImg and OpenCV image treatment libraries, apart from Tesseract OCR engine for character recognition in images.

Starting from a mobility problem, we will try to solve it using the tools that we learned during the degree, which will be useful in the development of the application. Several shape recognition techniques to identify the doors of the wagons were proposed, for example, the detection of the corners or the black margins of the doors.

When analyzing the images taken from the different types of wagons in the lines of Metrovalencia, we arrived at the conclusion that there are many different door shapes according to the shape of the door windows.

However, we can always find a sign, which is always the same, above every door. This way, the application is based on localizing the sign and then, detect if the door is open or closed by means of shape recognition.

Índice

1. Introducción

A raíz de estos años utilizando el metro para ir a la universidad, tuve tiempo para observar que no hay muchas personas invidentes que lo utilicen. Esto me llevó a pensar que tal vez hay alguna barrera que les dificulte su uso de manera independiente. Por ejemplo, en el autobús, si no hay nadie más en la parada, el conductor puede indicar a la persona invidente por dónde entrar y el destino de ese autobús. Sin embargo, esta situación en el metro es más complicada porque el conductor se encuentra aislado en la cabina y, si quiere indicar algo a la persona, tiene que salir de esta.

Entonces me pregunté qué podía hacer para ayudarles, no solo a moverse por el metro sino a tener una vida más independiente. Buscando algo de información acerca de nuevas tecnologías creadas para este fin, encontré una charla TED con el título *How new technology helps blind people explore the world*, por la Dra. Chieko Asakawa. En esta charla se mostraba un asistente virtual que indicaba lo que la persona tenía a su alrededor, si estaba cerca de una puerta cerrada, qué alimentos estaba cogiendo en la cafetería e incluso indicarle que se acercaba un conocido suyo. Todo esto gracias a una cámara, a un teléfono inteligente y a una aplicación con visión artificial.

Esto me motivó a querer hacer algo parecido, ya que en la asignatura Tratamiento de Imágenes aprendimos varias cosas sobre visión artificial. Al hablarlo con el profesor de esta asignatura y mi tutor del trabajo de fin de grado, me recomendó que empezara por algo más concreto, por lo que decidimos empezar con la detección de las puertas del metro.

Antes de empezar el trabajo, busqué información para saber si las personas invidentes tienen alguna manera de ubicarse en el metro y localizar las puertas. Encontré un video de un telediario en el que se mostraba cómo se guían las personas invidentes. Mediante las líneas de advertencia que hay por el suelo de la estación, ubican las escaleras para bajar al metro. De igual manera, se guían con la línea de distancia de seguridad cuando esperan a que llegue el metro.

Sin embargo, no tienen una manera muy fiable de ubicar las puertas, ya que el método que utilizan es buscar con el bastón un hueco en el metro, lo que puede hacer que se confundan con el hueco que hay entre vagones. Esto le pasó a una mujer en el metro de Barcelona, la cual se confundió y cayó a las vías por el hueco de los vagones.

La aplicación que se ha empezado a desarrollar pretende evitar este tipo de accidentes y facilitar la movilidad de las personas invidentes. Para ello, se ha trabajado en lenguaje C++ con las librerías de visión artificial CImg y OpenCV.

Antes de empezar a desarrollar la aplicación, creé una galería de imágenes de puertas de metro Valencia tanto abiertas como cerradas, a partir de fotografías que he ido tomando, reuniendo un total de 61 imágenes.

Cuando tuve un número suficiente de imágenes, comencé con la detección de los letreros, «Deixeu eixir/Dejen salir» que se encuentran encima de las puertas del metro. Se ha utilizado el motor de reconocimiento de caracteres, Tesseract OCR.

Una vez conseguidos los suficientes resultados positivos dimos por concluido el desarrollo de la aplicación.

2. Metodología

En este apartado, se explicará el funcionamiento de la aplicación de manera más general. También se describirán los conceptos necesarios para comprender su desarrollo y, por último, se entrará en más detalle en el funcionamiento y se proporcionarán algunos ejemplos.

La aplicación, es capaz de detectar el letrero «Deixeu eixir/Dejen salir» ubicado en la parte superior de la puerta del metro. Una vez se ha detectado el letrero, se procede a analizar en qué lugar se ubica la línea negra que forma parte de la junta central de la puerta. Dependiendo de la ubicación en la que se encuentre la línea respecto al letrero, se establece si la puerta está abierta o cerrada.

El esquema que desarrolla la aplicación se puede simplificar en tres pasos.

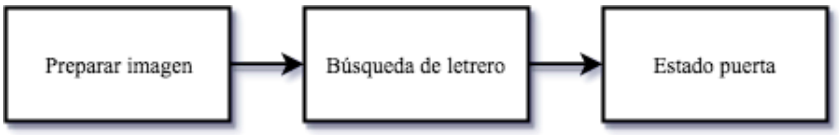

Figura 1. Diagrama básico aplicación.

Para que la aplicación pueda procesar la imagen correctamente, es necesario tratar la imagen. Primero, se debe comprobar si la imagen es en escala de grises o en RGB, si la imagen es en RGB se convertirá a escala de grises.

Una vez tengamos la imagen en escala de grises, escalaremos la imagen a una altura de 720 píxeles, para establecer un tamaño reducido para que cueste menos de procesar, pero manteniendo detalles.

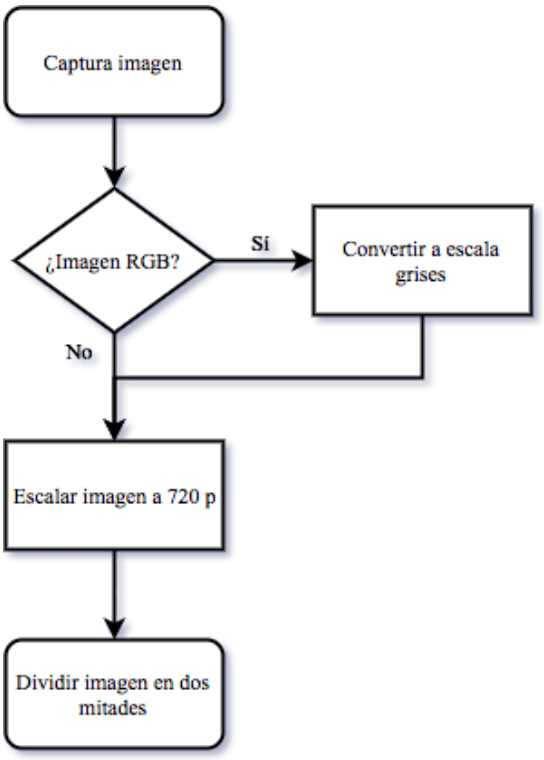

Figura 2. Diagrama preparación imagen.

Por último, se divide la imagen en dos mitades con un corte horizontal. Esta división se realiza para que la detección del letrero sea más rápida. Primero, se procesará la mitad superior, ya que, de normal, la posición en la que se encuentra el letrero en la imagen ocupa la mitad superior. Y en caso de que no encontrara el letrero en la mitad superior, se procesará la mitad inferior. Se decide realizar el proceso de esta manera, porque al analizar la imagen con Tesseract OCR, la aplicación se ralentiza, así podemos reducir hasta la mitad el tiempo que tarda.

Para la búsqueda del letrero, se obtienen los bordes de todos los objetos que se encuentran en la imagen, se alargan estos bordes horizontalmente para unirlos, de esta manera, se busca que los bordes de las letras del letrero formen un único objeto.

Después, se realiza una búsqueda de los contornos que hay en la imagen. Para cada uno de estos contornos se creará un rectángulo que los contenga. En caso de que el ancho del rectángulo sea mayor que el doble de su altura, el rectángulo pasará a ser analizado para comprobar si contiene algún texto. Si la cadena de caracteres que se encuentra en el texto es menor a 10 caracteres, el rectángulo queda descartado y se vuelve a realizar el proceso con el siguiente contorno.

Para comprobar que la imagen encontrada en el texto corresponde al letrero que estamos buscando, se calcula la distancia de edición entre la cadena obtenida y la cadena de referencia «Deixeu eixir/Dejen salir». Pueden darse tres casos, que la distancia de edición sea muy grande y se descarte el contorno, que la distancia de edición sea muy pequeña y el contorno pertenezca al letrero, o, por último, que la distancia de edición sea próxima pero no la suficiente para asegurar que es el letrero. En este último caso, se realizan varios tratamientos a la imagen para mejorar su calidad y se vuelve a analizar si contiene algún letrero. Estos tratamientos se detallarán más adelante.

Una vez localizado el letrero, se recorta la imagen inicial para obtener una imagen con el ancho del letrero. Después, se procede a realizar varias operaciones morfológicas para encontrar y quedarnos solamente con las líneas negras que se hayan entre las puertas. Para conseguirlo, se ha estimado que, como mínimo, el ancho de la línea debe ser de un grosor de 3 pixeles y el alto de 31 pixeles. En la imagen binaria que quede como resultado de este proceso, se buscarán los contornos y se seleccionará el contorno más alto. Obtendremos la coordenada ‘x’ de este contorno, perteneciente a las líneas negras del borde de la puerta, para saber en qué posición horizontal se encuentra.

El siguiente paso sería, a partir de la longitud del letrero, ubicar si el contorno obtenido se encuentra en el centro o en uno de los extremos del letrero. Para esto se dividirá el letrero en 5 partes. Si el contorno se encuentra en la parte central, la puerta está cerrada. Si el contorno se encuentra en una de las partes de los extremos, la puerta se encontrará abierta. Y si el contorno se encuentra en una de las partes intermedias, la puerta se encuentra en estado indeterminado, ya que, esa ubicación es propia del momento en que las puertas están abriéndose o cerrándose.

Por último, hay que tener en cuenta que para que la aplicación pueda trabajar con las imágenes correctamente, estas deben mostrar la puerta de manera correcta. Se recomienda que haya como mínimo una distancia de separación de poco más de un metro respecto al metro, para que la cámara del móvil sea capaz de capturar el letrero y la puerta.

2.1. Conceptos

Aspectos geométricos

La distancia mínima para que la imagen tomada por la cámara capte toda la puerta, se calcula con el ángulo de visión horizontal que tenga la cámara del teléfono móvil, este ángulo suele estar entre unos 70º o 50º. También hay que tener en cuenta el tamaño de la puerta y el letrero, que suele ser de una longitud de 110 cm. En el peor de los casos, con un ángulo de 50º necesitaremos estar a una distancia de 1,17 metros.

$$\boldsymbol{distancia\ (m)} = \frac{\boldsymbol{0{,}55\ metros}}{\boldsymbol{\tan(25°)}} \qquad \textbf{(1)}$$

Tipos de imagen

En nuestro caso, la aplicación no necesita procesar información sobre el color de los objetos, por lo que se trabajará con imágenes en escala de grises, de esta manera nos quedaremos con un menor número de canales que con una imagen RGB.

Cuando la aplicación se ejecute, la imagen en escala de grises se convertirá a una imagen binaria. En esta imagen, los valores de menor intensidad se asignarán a las regiones que no interesan y los valores de mayor intensidad a los detalles que queramos conservar.

Binarización

La imagen binaria nos resulta muy útil en el tratamiento de imágenes, ya que, permite segmentar la imagen para guardar solamente la información necesaria y hace que su procesamiento sea más rápido. Sin embargo, no siempre se obtienen estas imágenes en un primer momento, para ello necesitaremos herramientas que nos permitan convertir imágenes en escala de grises en imágenes binarias.

Los umbrales nos permiten binarizar la imagen seleccionando un valor a partir del cual aceptaremos el píxel o lo descartaremos. La selección del valor de umbral determinará el tipo de umbral que utilizaremos, el cual podrá ser un valor fijo, un valor determinado por el histograma (método de Otsu) o un valor diferente dependiendo de la zona de la imagen en la que nos encontremos (umbral adaptativo).

La aplicación hace uso de umbrales fijos invertidos. A este umbral se le asigna un valor a partir de cual, aceptaremos el pixel si su intensidad es inferior a la del valor del umbral, o lo descartaremos si la intensidad del píxel es superior al valor del umbral.

$$\boldsymbol{imagen\ resultante}(x,y) = \begin{cases} \boldsymbol{0}, & \boldsymbol{imagen\ original}(x,y) > \boldsymbol{umbral} \\ \boldsymbol{255}, & \boldsymbol{otros\ casos} \end{cases} \textbf{(2)}$$

El umbral fijo invertido es de utilidad en los casos en que el objeto que nos interesa es oscuro sobre fondo claro.

Figura 3. Binarización de imagen con umbral fijo invertido.

En este caso se ha utilizado un umbral de valor bajo para quedarnos solo con las letras negras.

Operaciones morfológicas

La erosión y la dilatación son dos operaciones básicas en el procesamiento morfológico. Estas operaciones utilizan una máscara que se denomina «Elemento Estructurante». El Elemento Estructurante (EE) es una matriz binaria con valores de 1 para la región en que el EE trabajará y 0 para los lugares en que no. La forma de este puede variar pudiendo ser un cuadrado, una circunferencia, una cruz, una línea horizontal o una línea vertical.

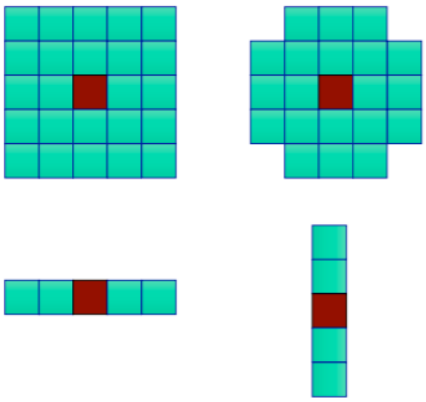

Figura 4. Tipos de Elemento Estructurante.

El centro de la matriz del EE, recorrerá cada píxel de la imagen y modificará el valor de este. Esta modificación viene dada por los valores de los píxeles que se encuentren dentro del EE, dependiendo de qué tipo de operación morfológica estemos utilizando.

En el caso de la Erosión, el valor del píxel del centro será modificado por el valor más pequeño que se encuentre dentro del EE. De esta manera, la Erosión nos sirve para eliminar las zonas o detalles claros de la imagen y aumentar el tamaño de los detalles oscuros.

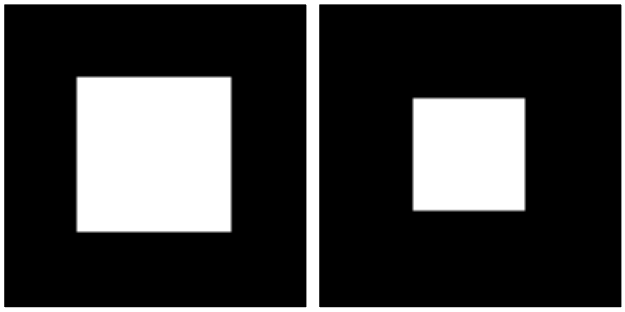

Figura 5. Ejemplo de Erosión con EE cuadrado.

Por otro lado, la Dilatación modifica el valor del píxel que se encuentra en el centro del EE por el valor más alto que se encuentre dentro del EE. Eliminando detalles o zonas oscuras y aumentando los detalles claros de la imagen.

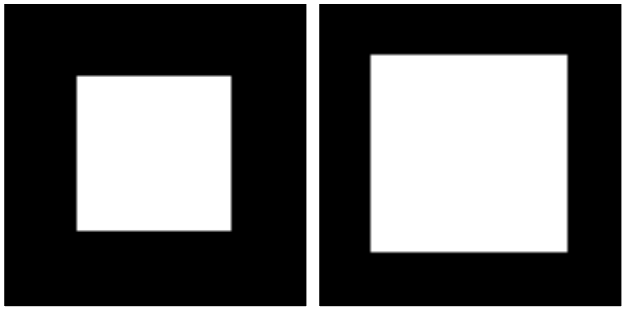

Figura 6. Ejemplo Dilatación con EE cuadrado.

Para utilizar estas operaciones correctamente, debemos tener en cuenta la forma y el tamaño de los detalles que deseamos conservar o eliminar de la imagen, para poder seleccionar el tamaño y la forma del EE que más convenga.

Mostraremos la importancia de esta selección con un ejemplo en que veremos cómo quedarnos con las formas deseadas.

Figura 7. Imagen inicial.

En esta imagen, podemos ver dos líneas horizontales de tamaño 20 x 5 px, una línea horizontal de 20 x 10 px, una línea vertical de 10 x 20 px, dos líneas verticales de 5 x 20 px y una cruz formada por una línea horizontal de 20 x 5 px y una línea vertical de 5 x 20 px. Utilizaremos dos tipos de EE con forma de línea, uno de ellos horizontal de 5 filas y 20 columnas, y el otro vertical de 20 filas y 5 columnas.

Primero, vamos a ver cómo podríamos unir dos figuras utilizando una dilatación con EE horizontal. Veremos que las dos líneas verticales paralelas se unirán en una sola figura, también veremos como las demás figuras estarán más alargadas en horizontal.

Figura 8. Dilatación con EE horizontal.

Si en lugar de usar la dilatación con el EE vertical, usamos el EE vertical, las figuras que se unirán en este caso serán las dos líneas horizontales paralelas y, además, las demás figuras se alargarán en vertical.

Figura 9. Dilatación con EE vertical.

Otro uso que nos ofrecen las operaciones morfológicas, es la posibilidad de eliminar figuras que no queremos y quedarnos con las que nos sean de utilidad. Para ello, necesitamos que el EE quede completamente dentro de la figura que deseamos mantener.

Por esto, si queremos conservar las figuras horizontales, utilizaremos un EE horizontal del mismo tamaño que las figuras o más pequeño, pero teniendo en cuenta que tiene que ser más grande que el ancho de las figuras verticales.

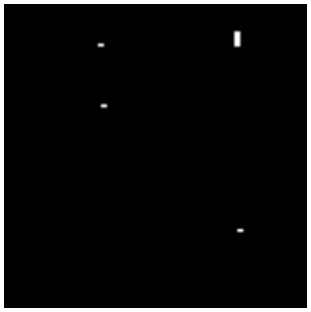

Figura 10. Erosión con EE horizontal.

En las figuras resultantes, vemos unos rectángulos más pequeños que las figuras de la imagen original, estos rectángulos se sitúan en los centros de las figuras horizontales, ya que el EE tenía el mismo tamaño horizontal que las figuras.

Para recuperar la forma inicial de las figuras horizontales, tendremos que aplicar una dilatación a la imagen resultante de la erosión con el EE horizontal utilizado anteriormente.

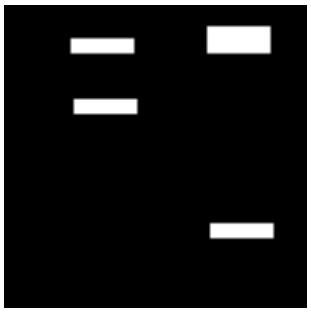

Figura 11. Dilatación con EE horizontal.

De esta manera, podemos ver cómo hemos recuperado las figuras horizontales, entre ellas también la línea horizontal de la cruz, ya que tiene el mismo tamaño que las demás figuras. Esta serie de operaciones se denomina apertura y sirve para eliminar o conservar elementos claros de la imagen sin alterar los elementos oscuros.

$$apertura(imagen\,, elemento\ estructurante) = dilatación(erosión(imagen\,, elemento\ estructurante)) \quad \textbf{(3)}$$

Sin embargo, si primero dilatamos la imagen y después la erosionamos, el proceso se denomina cierre y sirve para eliminar o conservar los detalles oscuros y mantener los detalles claros.

$$cierre(imagen\,, elemento\ estructurante) = erosión(dilatación(imagen\,, elemento\ estructurante)) \quad \textbf{(4)}$$

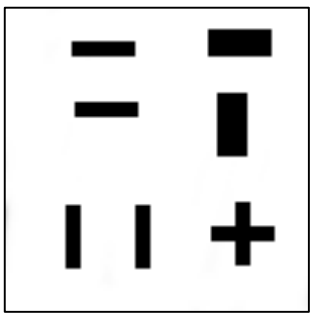

Figura 12. Imagen inicial con colores invertidos.

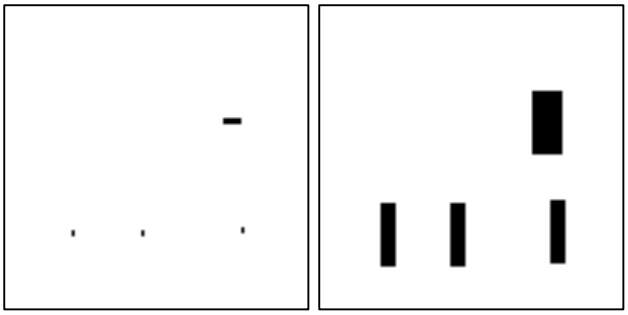

Figura 13. Cierre con EE vertical.

En esta ocasión, al realizar el cierre con el EE vertical, hemos mantenido las líneas verticales y eliminado las líneas horizontales.

Otra de las operaciones morfológicas que utilizaremos será el blackhat, este es el residuo del cierre y de la imagen original. Devuelve una imagen en negro con los detalles negros en blanco. Será útil para encontrar las figuras negras de una imagen, en nuestro caso, los detalles de las puertas como los bordes y marcos.

$$blackhat(imagen\,, elemento\ estructurante) = cierre(imagen\,, elemento\ estructurante) - imagen \quad \textbf{(5)}$$

Figura 14. Borde de la puerta y blackhat.

Detección de bordes

En el tratamiento de imagen, los bordes nos indican cambios de intensidad abruptos en la imagen. Estos bordes pueden indicar la forma de una figura que se encuentre en la imagen. La librería OpenCV, tiene varias funciones que resultan útiles para la detección de bordes, pero en esta ocasión usaremos la función *Canny*.

Esta función, utiliza el algoritmo de Canny, desarrollado por John F. Canny en 1986. Este algoritmo es capaz de detectar una amplia variedad de bordes en la imagen. Este algoritmo busca cumplir tres criterios:

- Detectar los bordes con el mínimo error, esto es, la detección debe captar con precisión tantos bordes como haya en la imagen.
- La distancia entre los pixeles detectados y los pixeles que realmente pertenecen a los bordes debe ser mínima.
- Un borde solo debe ser marcado una vez, intentando que el ruido de la imagen no cree falsos bordes.

Para cumplir estos requisitos, la función lleva a cabo los siguientes pasos:

1. Filtrar el ruido de la imagen, utilizando un filtro Gaussiano. Esto nos creará una imagen más borrosa. El filtro se implementa convolucionando la imagen con una máscara. Cuanto mayor sea la máscara peor será la sensibilidad al ruido, por lo que la máscara debe ser mucho menor que la imagen. Se utiliza una máscara de tamaño 5x5 con una desviación típica (σ) de 1,4.

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Figura 15. Filtro gaussiano de 5x5.

2. Obtener el gradiente de la imagen. Para la obtención del gradiente utiliza el operador Sobel, el cual convoluciona dos máscaras de tamaño 3x3, una para estimar el valor de gradiente en el eje 'x' y otra para el eje 'y'.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Figura 16. Máscaras para gradientes eje 'x' y eje 'y'.

Para calcular la magnitud y la dirección del gradiente utiliza las siguientes fórmulas.

$$\boldsymbol{G = \sqrt{G_x^2 + G_y^2}} \quad \textbf{(6)}$$

$$\boldsymbol{\theta = \arctan\left(\frac{Gy}{Gx}\right)} \quad \textbf{(7)}$$

La dirección se aproxima a una de los cuatro ángulos posibles (0º, 45º, 90º o 135º).

3. Supresión no máxima para eliminar pixeles que no se consideren parte del borde. Esta es una técnica para conseguir los bordes más delgados. Se comprueba si el valor de la magnitud de gradiente del píxel es el máximo local de sus vecinos más próximos y si es el caso, se le asigna el valor 0, si no lo es, se le asigna el valor que tiene la magnitud del gradiente.

4. Histéresis del umbral. Se realizan dos umbrales, uno superior y otro inferior. Si el gradiente de un píxel es mayor que el umbral superior, ese píxel se definirá como borde. Si el gradiente es inferior al umbral inferior, el píxel será rechazado. Si el gradiente está entre los dos umbrales, se aceptará si está conectado a un píxel que pertenezca a un borde.

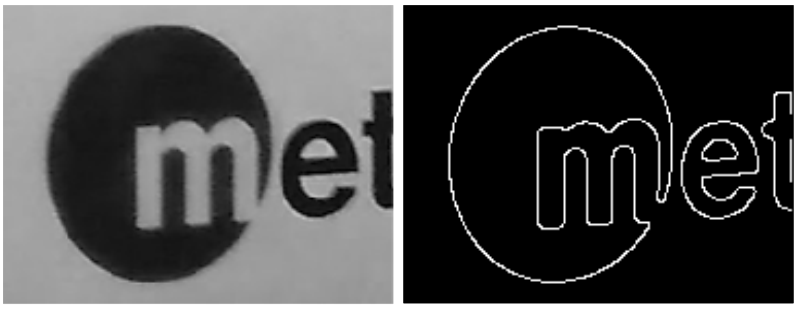

Figura 17. Imagen original y detección de bordes.

La obtención de los bordes es el paso previo a reconocer si ese borde corresponde al contorno de un objeto o a detalles de la imagen que no nos sirvan para el análisis de la imagen.

Búsqueda de contornos

Una vez procesados los bordes de la imagen, debemos identificar si estos bordes corresponden a la forma de algún objeto de la imagen. Cada objeto que encontremos en la imagen, estará definido por uno o varios contornos. A diferencia de los bordes, un contorno es una curva de puntos sin huecos ni saltos.

La librería OpenCV, ofrece una función capaz de realizar esta búsqueda, *findContours*. Para poder realizar esta búsqueda de manera más precisa, se deben utilizar imágenes binarias, en que las figuras deben ser blancas y el fondo negro.

Los contornos pueden clasificarse en dos tipos, exteriores e interiores. Un objeto puede tener un contorno exterior y, además, uno o más contornos interiores, dependiendo de si el objeto tiene huecos o no. Cuando un contorno contiene dentro otros contornos, se establece una jerarquía de dos niveles. Al contorno que contiene a los demás contornos, se le denomina contorno padre y a los contornos contenidos, contornos hijo.

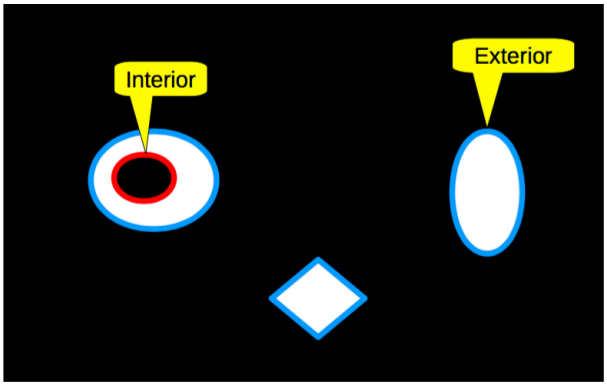

Figura 18. Ejemplo de contornos exterior e interior.

En esta figura se puede ver que el objeto que está a la izquierda, tiene un contorno exterior y otro interior, el exterior es el contorno padre y el interior el contorno hijo.

La función *findContours*, nos devolverá un vector de vectores, en los que se almacenan las coordenadas que componen cada contorno. También, puede devolver un vector con detalles de la jerarquía de cada contorno, si tiene padre, hijos o cuál es el siguiente contorno que se encuentra al mismo nivel jerárquico.

Hay distintos modos de almacenar los contornos. La aplicación utilizará el modo CV_RETR_CCOMP, este modo almacena todos los contornos y los organiza en dos categorías, exteriores e interiores, sin almacenar información sobre si pertenece a algún padre o si tiene hijos.

Una vez buscados los contornos, se debe analizar cada uno para encontrar el contorno deseado. La aplicación basa su análisis en el tamaño y la forma del contorno, y dado que la forma del letrero se adapta bien a un rectángulo alargado, se utiliza una función que crea un rectángulo que encierra el contorno.

Esta función se llama *boundingRect*, esta recibe el contorno y devuelve el rectángulo perteneciente al contorno. Crea este rectángulo a partir de dos puntos, uno de estos puntos se consigue con los valores más bajos de la fila y la columna del contorno, el otro punto se consigue con los valores más altos de la fila y la columna. Esta función no tiene en cuenta la rotación del contorno, por lo que puede que el rectángulo no sea el más ajustado al contorno.

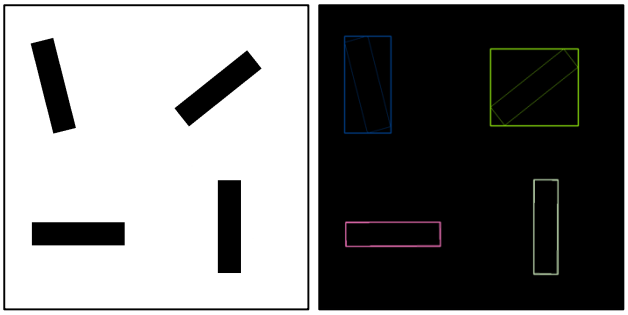

Figura 19. Ejemplo rectángulos de contornos.

Tesseract OCR

Tesseract OCR, es un motor óptico de reconocimiento de caracteres desarrollado por Hewlett-Packard entre 1984 y 1994, y que, posteriormente, fue lanzado como código abierto. Desde ese momento, Google se encarga de su desarrollo, bajo la dirección de Ray Smith. Se puede descargar desde GitHub. Y soporta alrededor de 100 idiomas, entre ellos inglés y español.

El uso de Tesseract se puede definir en estos pasos:

1. Crear el objeto de tipo tesseract.
2. Inicializar el objeto y definir el idioma.
3. Definir el nombre de la imagen, el número de columnas, el número de filas, el número de canales y en qué pixel debe saltar a la siguiente fila de píxeles.
4. Almacenar el texto extraído de la imagen. Nos devuelve la cadena con codificación unicode (UTF-8).
5. Eliminar el objeto para liberar memoria.

En nuestro caso, el idioma que definiremos será el que viene por defecto, el inglés, ya que la cadena que buscamos no incluye ningún caracter exclusivo del castellano o el valenciano. Antes de devolver la cadena obtenida, comprobaremos si tiene espacios al inicio y al final de la cadena y los eliminaremos.

Distancia de edición

La distancia de edición o distancia de Levehnstein, indica el número mínimo de modificaciones que necesitamos en una cadena de caracteres para convertirla en otra. Las modificaciones pueden ser: eliminar, añadir o cambiar de lugar caracteres de la cadena. Este algoritmo se suele utilizar en correctores ortográficos. A continuación, se muestran varios ejemplos:

- Rojo → Roto: número de operaciones 1, sustitución de la j por la t.
- Roto → Robot: número de operaciones 2, inserción de la b y cambio de lugar de la t.
- Robot → Robo: número de operaciones 1, eliminación de la t.

El algoritmo que calcula esta distancia no necesita que las cadenas sean del mismo tamaño, como sí pasa si se utiliza la distancia de Hamming. Este parámetro, nos será útil a la hora de comprobar si el texto que detectamos en la imagen es igual o muy similar a la cadena de referencia que estemos buscando.

En nuestro caso descargamos la librería Edlib de GitHub para implementar este algoritmo. Para el uso de la función que implementa el algoritmo, se necesitan 4 argumentos, estos son, las cadenas que queremos calcular y la longitud de cada una.

2.2. Detección letrero

La detección del letrero parte de una imagen en escala de grises dividida, en primer lugar con la mitad superior y en caso de no encontrarse letrero, se repite el proceso con la mitad inferior. Posteriormente, cuando se analice el estado de la puerta, se trabajará con la imagen completa en vertical, ya que, dividiendo la imagen en dos mitades horizontales, podriamos partir los contornos de las líneas.

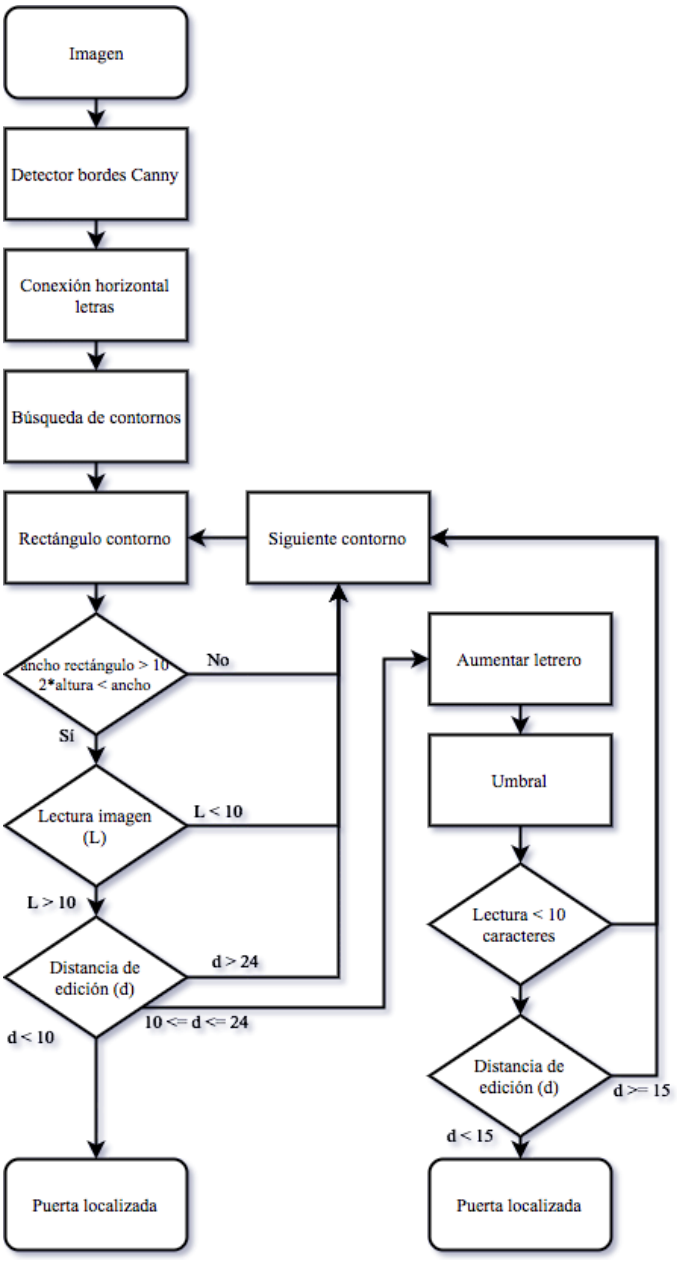

Figura 20. Diagrama búsqueda letrero.

A partir del diagrama (Figura 20), se puede ver cómo se desarrolla la aplicación. Primero, recibe la imagen en escala de grises y aplica la detección de bordes de Canny, con el valor del umbral inferior de 100 y el valor del umbral superior de 200. No se espera que la imagen tenga una intensidad muy oscura, pero tampoco demasiado clara, ya que la estación del metro suele estar iluminada. Al usar la detección de bordes, se obtiene una imagen binaria con los bordes obtenidos.

Después, a la imagen binarizada se le realiza un cierre con un EE con forma de línea horizontal y tamaño de 15 píxeles. Con esto, se busca unir los contornos de los bordes de las letras, para conseguir que todas las letras del letrero formen un solo objeto alargado.

Se realiza una búsqueda de los contornos exteriores y se crea el rectángulo que encierra cada contorno. Para hacer una selección de los contornos previa al análisis con Tesseract OCR, se seleccionan los rectángulos con una anchura superior a 10 píxeles, en los que el ancho sea mayor al doble de la altura. De esta manera, descartaremos los contornos que no sean alargados horizontalmente y los que sean pequeños.

Los contornos que cumplen estas condiciones son analizados con Tesseract OCR, en el caso de que un contorno devuelva una cadena de tamaño inferior a 10 caracteres, este se descarta. Si el contorno no ha sido descartado se calcula la distancia de edición.

Para el cálculo de la distancia de edición, utilizaremos la cadena «Deixeu eixir/Dejen salir» como cadena de referencia. Esta cadena tiene un total de 24 caracteres. Si la distancia de edición es superior a 24, el contorno queda descartado ya que, no tiene nada en común con el letrero. Si la distancia de edición es inferior a 10, el contorno se selecciona como letrero. En caso de que la distancia de edición no cumpla las condiciones anteriores, se procederá a tratar la imagen que queda dentro del rectángulo del contorno correspondiente. Al poder darse el caso de que la aplicación devuelva una cadena mayor a 24 caracteres por posibles objetos cercanos al letrero, se deja un límite de distancia de edición del mismo tamaño que la cadena, ya que, posteriormente será procesada la imagen para volver a analizarse con Tesseract OCR.

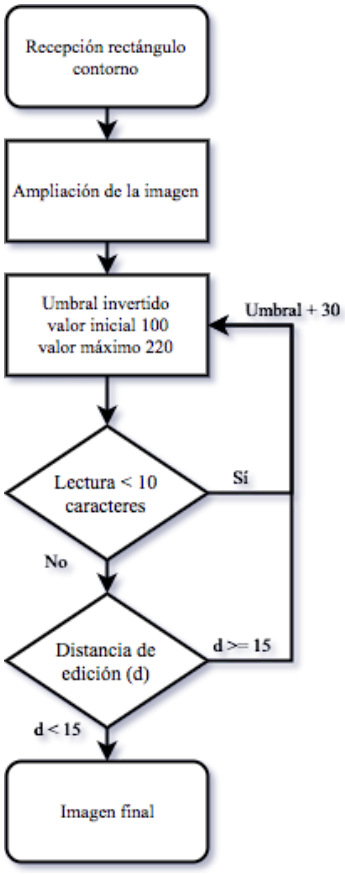

Figura 21. Diagrama para mejora de lectura.

Para mejorar la imagen, primero se amplía al doble de su tamaño. Se aplica un umbral invertido, ya que las letras del letrero son en negro, con un valor inicial de 100 y, cada vez que se tenga que volver a realizar el umbral, se sumará 30 a su valor hasta que llegue al valor máximo de 220.

Estos umbrales se aplican para mejorar el contraste de las letras y el fondo, para mejorar la lectura. Se realiza el análisis con Tesseract OCR y, en caso de que la lectura sea inferior a 10 caracteres, se vuelve a realizar el umbral.

Si no se ha descartado la imagen, se calcula la distancia de edición. En caso de que esta sea inferior a 15, se considera que el contorno es el letrero. En caso contrario, se realiza otro umbral. Sin embargo, si se han realizado todos los umbrales y no se ha encontrado el letrero, el contorno se descarta.

Los límites de la distancia de edición que se deben cumplir en cada caso, han sido establecidos y ajustados tras comprobar el funcionamiento de la aplicación con la librería de imágenes.

A continuación, veremos cómo funciona la aplicación a través de varios ejemplos, cubriendo distintas situaciones con las que nos podremos encontrar.

Ejemplo 1

La primera imagen que utilizamos fue detectada sin ningún problema.

Figura 22. Puerta cerrada de metro en RGB.

Primero, se convierte la imagen a escala de grises y se modifica el tamaño a una altura de 720 píxeles sin alterar la relación de aspecto de la imagen.

Figura 23. Puerta de metro cerrada en escala de grises.

Se recorta la imagen en dos mitades y primero se trabaja con la mitad superior.

Figura 24. Parte superior de la imagen.

Lo siguiente, es realizar la detección de contornos utilizando el algoritmo Canny. Quedando una imagen binaria, en la que se ha eliminado mucha información innecesaria. Entre estos bordes, se puede distinguir el letrero.

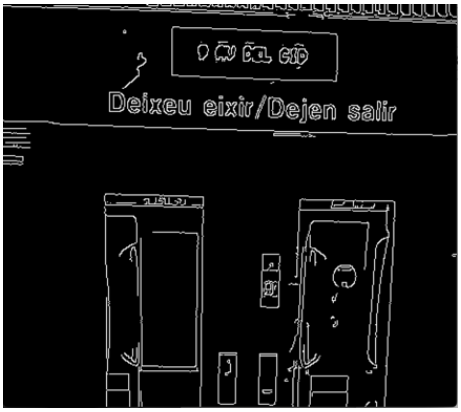

Figura 25. Detección de contornos con algoritmo Canny.

Después, se procede a unir el letrero para crear un solo contorno, así más adelante, no se tendrán que buscar los 24 caracteres del letrero por separado.

Figura 26. Conexión horizontal de contornos.

Se buscan los contornos y se crea el rectángulo que contiene cada contorno. En el análisis de los rectángulos, el siguiente rectángulo cumple los requisitos del tamaño mínimo y la relación entre altura y anchura, y se analiza con Tesseract OCR.

Figura 27. Letrero.

A continuación, se calcula la distancia de edición, lo que da como resultado una distancia de 0, esto es, que el letrero se ha leído sin ningún fallo.

Figura 28. Letrero localizado en la parte superior de la puerta.

Al encontrar el letrero, ya podemos ubicar la puerta del metro para posteriormente, analizar si la puerta se encuentra abierta o cerrada.

Ejemplo 2

Hay otras situaciones en las que sí se detecta un contorno que puede contener un letrero, pero la distancia de edición da un valor superior a 10 e inferior o igual a 24. En este ejemplo se muestra como se trata la imagen para mejorar la lectura.

Figura 29. Puerta metro en escala de grises.

En esta imagen, el letrero se encuentra un poco inclinado y tiene una luz de advertencia en el lado derecho.

Figura 30. Detección contorno.

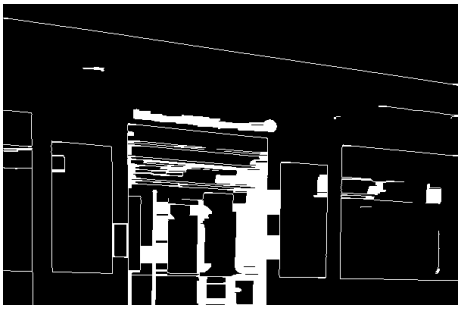

Figura 31. Conexión horizontal.

En esta ocasión, el contorno de la bombilla se encuentra cerca de las letras, por lo que los contornos se juntarán creando un solo objeto alargado. Esto no supondrá un problema, ya que se seguirá detectando un rectángulo alargado horizontalmente.

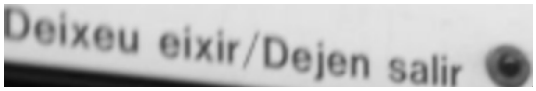

Figura 32. Letrero.

El letrero ha sido detectado, pero la lectura devuelta por Tesseract tiene una distancia de edición superior a 10. Para asegurarse de que no es una lectura errónea, se amplía el letrero y se realizan varias umbralizaciones con distintos valores de umbral.

Figura 33. Umbral con valor 100.

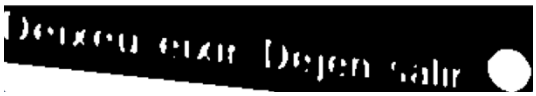

Figura 34. Umbral con valor 130.

Figura 35. Umbral con valor 160.

Figura 36. Umbral con valor 190.

Figura 37. Umbral con valor 220.

De todos estos valores de umbral, se selecciona el que proporciona una distancia de edición menor, quedándonos en esta ocasión con un umbral de 160. De esta manera, hemos mejorado la lectura y se ha pasado de una distancia de edición de 13 a 9. Ubicando correctamente el letrero sobre las puertas del metro.

Figura 38. Letrero localizado.

2.3. Detección puertas abiertas o cerradas

A partir de la ubicación del letrero, la aplicación procederá a identificar en qué estado se encuentra la puerta del metro, cerrada, abierta o sin determinar.

El funcionamiento es el siguiente.

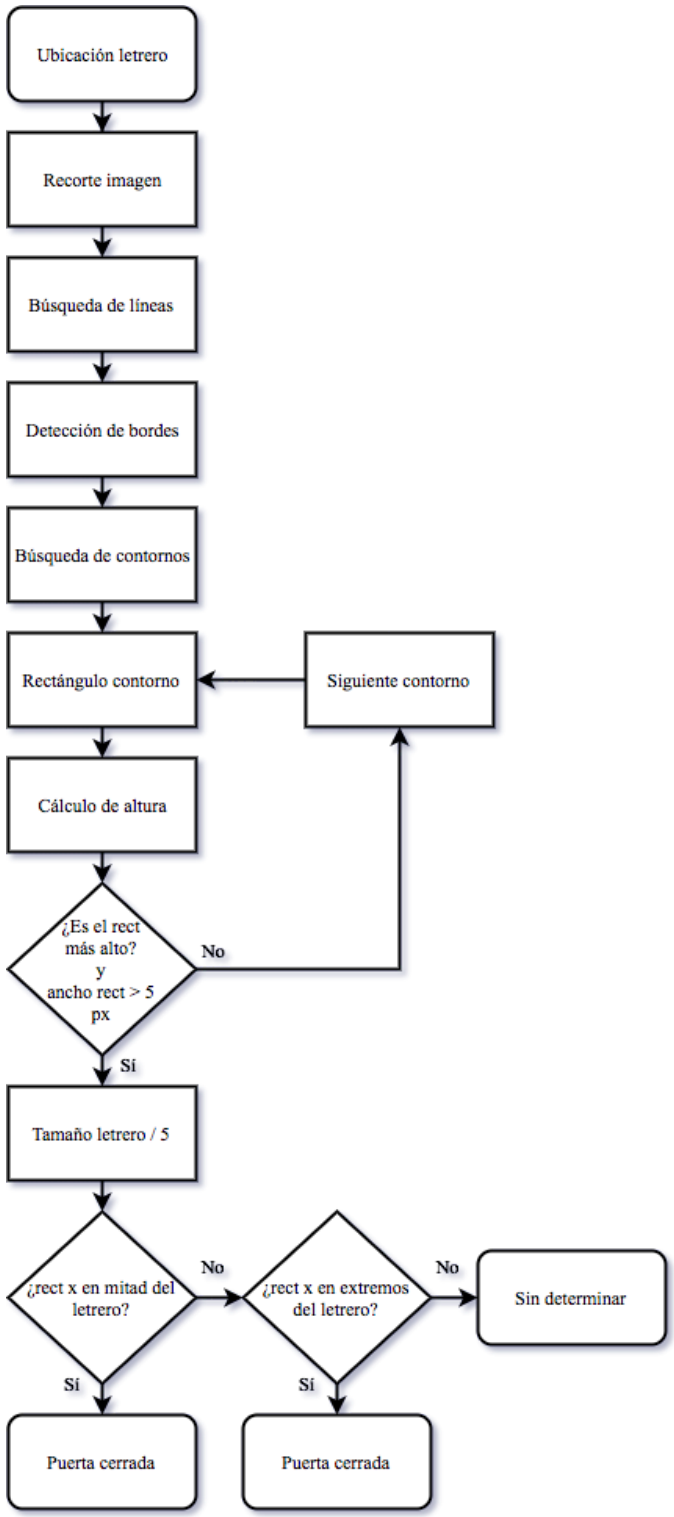

Figura 39. Diagrama detección del estado de las puertas.

Primero, se recortará la imagen con los límites establecidos por el ancho del letrero, ya que el borde de la puerta se encontrará entre estos límites y, de esta manera, quitaremos objetos que no interesen o que puedan perjudicar el análisis.

Una vez hecho esto, se procede a la búsqueda de las líneas que haya en la imagen. Para esto, se ha creado un constructor que se encarga de realizar los procesos necesarios para identificar las líneas. Dentro de este constructor los procesos son los siguientes.

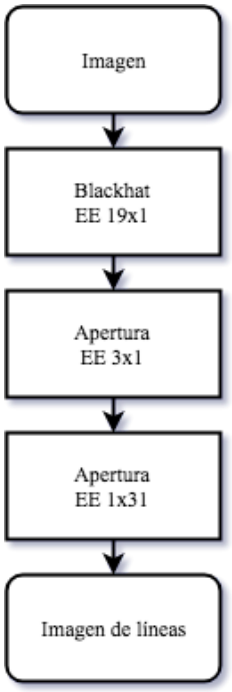

Figura 40. Diagrama buscador de líneas.

La primera operación es un *blackhat* con EE con forma de línea de tamaño 19x1, para conseguir resaltar en blanco las líneas negras de las puertas del metro.

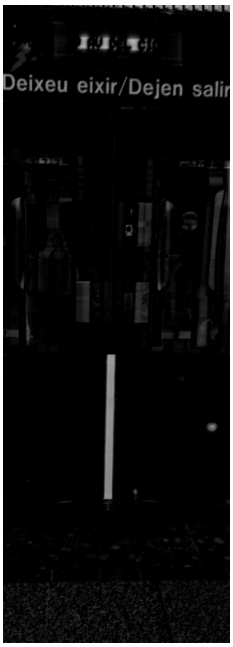

Figura 41. Blackhat EE 19x1.

Después, se realiza una apertura con EE de tamaño 3x1, para eliminar líneas delgadas que se hayan podido quedar en la imagen.

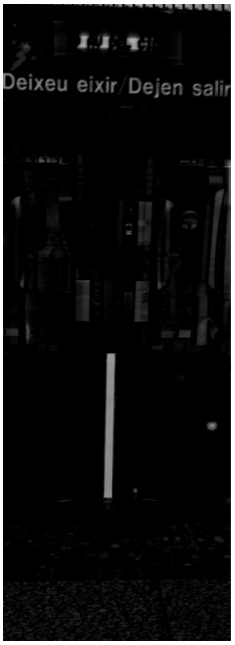

Figura 42. Apertura EE 3x1.

Por último, para eliminar las líneas que no tengan una altura mínima, se realiza otra apertura con un EE de tamaño 1x31. Este tamaño, se ha estimado como el tamaño mínimo de las posibles puertas de metro que se encuentran en MetroValencia, a una distancia que pueda ser captada sin problemas por la aplicación.

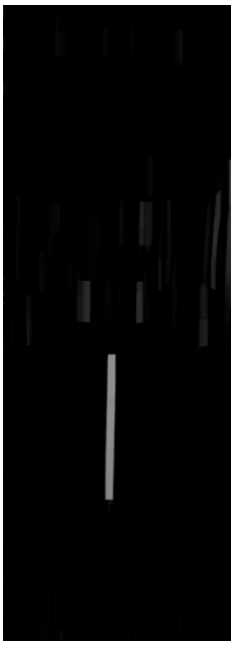

Figura 43. Apertura EE 1x31.

Tras obtener la imagen con las líneas negras destacadas, procedemos a la detección de bordes para conseguir una imagen binaria con los bordes de las líneas. Se buscan los contornos de esta imagen binaria y se selecciona el contorno de más altura y con un ancho superior a 5 píxeles.

Por último, se obtiene la coordenada 'x' del contorno seleccionado, de esta manera, ubicaremos en qué posición se encuentra la puerta, utilizando el letrero como referencia.

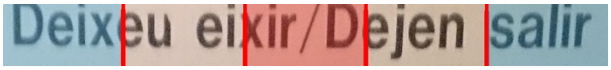

Figura 44. Ejemplo letrero de referencia.

Se divide el letrero en 5 partes y, según en qué zona se encuentre el contorno, se establecerá el estado en el que se encuentra la puerta. Si el contorno se encuentra en la zona central, la puerta está cerrada. Si el contorno se encuentra en uno de los extremos, la puerta está abierta. En los casos restantes, el estado de la puerta no se puede determinar, ya que se es una posición intermedia entre abierta o cerrada y haría falta imágenes posteriores para saber cuál de las dos acciones estaba realizando.

Ejemplo 1

Vamos a continuar con el ejemplo 1, ahora ya está localizado el letrero, pero falta establecer si la puerta está abierta o cerrada.

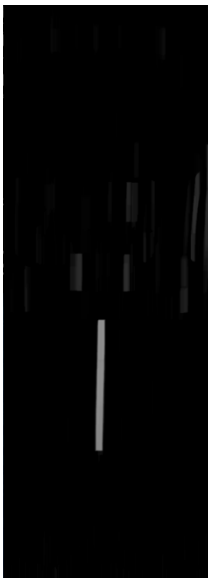

Figura 45. Imagen líneas en blanco.

Primero, se recortará la imagen con las coordenadas del letrero obtenidas anteriormente. La imagen se procesará para que solo queden las líneas negras.

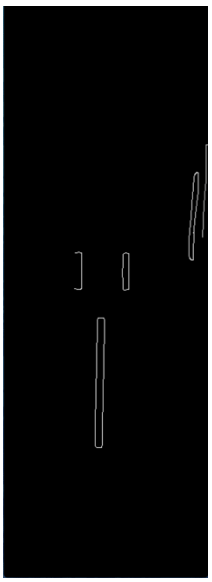

Figura 46. Detección de bordes.

Después, se utiliza el algoritmo Canny para la detección de los bordes. En la imagen, se puede observar que solo han quedado tres bordes cerrados, en los que se puede distinguir que el contorno de más altura corresponderá a la línea negra de la puerta del metro y que se encuentra en el centro de la imagen.

Figura 47. Línea de la puerta de metro detectada.

Tras realizar la búsqueda de contornos, se ha detectado sin problemas la línea negra que se encuentra entre las puertas de metro. Solo se ha detectado la mitad de la línea, porque al ser negra la parte superior de la puerta, la aplicación no es capaz de identificarla por completo.

Por último, se comprueba en qué ubicación se encuentra el contorno. En este caso, se encuentra en la zona central y se ha identificado sin problemas que la puerta se encuentra cerrada.

Ejemplo 2

A partir del ejemplo 2 del apartado anterior, vamos a comprobar cómo funciona la detección cuando la puerta está abierta.

Figura 48. Recorte imagen.

Primero, se recorta la imagen ajustándola al ancho del letrero obtenido anteriormente.

Figura 49. Imagen tras buscador de líneas.

Esta vez, al procesar la imagen con el buscador de líneas, nos aparecen más líneas que en el ejemplo anterior. Además, hay algunas con tamaños parecidos, puesto que la puerta estaba abierta y se podían ver las líneas de la puerta del otro lado del vagón.

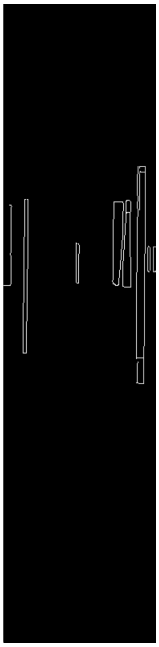

Figura 50. Detección de bordes.

Cuando se utiliza el detector de bordes, las figuras se pueden ver más definidas y ya podemos distinguir una figura más alargada.

Figura 51. Puerta abierta detectada.

Por último, tras la búsqueda de los contornos, la aplicación ha dibujado el contorno que ha sido seleccionado, el cual corresponde al borde de la puerta. Además, como el borde se encuentra a un extremo del letrero, se ha establecido que la puerta está abierta.

3. Resultados

A partir de la galería de fotos que he conseguido crear, comprobaremos cómo se comporta la aplicación y qué casos han resultado más difíciles de determinar.

A continuación, se muestran los resultados que devuelve la aplicación.

```
metro_ab01.jpg Letrero leído con distancia = 0
Puerta abierta.

metro_ab02.jpg Letrero leído con distancia = 0
Puerta abierta.

metro_ab03.jpg Letrero leído con distancia = 6
Puerta cerrada.

metro_ab04.jpg Letrero leído con distancia = 2
Puerta abierta.

metro_ab05.jpg Letrero leído con distancia = 4
Puerta abierta.

metro_ab06.jpg Letrero leído con distancia = 1
Puerta abierta.

metro_ab07.jpg Letrero leído con distancia = 0
Estado indeterminado.

metro_ab08.jpg No se encontró letrero.

metro_ab09.jpg Letrero leído con distancia = 12
Puerta abierta.

metro_ab10.jpg No se encontró letrero.

metro_ab11.jpg Letrero leído con distancia = 9
Puerta abierta.

metro_ab12.jpg No se encontró letrero.
```

metro_ab13.jpg No se encontró letrero.

metro_ab14.jpg No se encontró letrero.

metro_ab15.jpg No se encontró letrero.

metro_ab16.jpg Letrero leído con distancia = 2
Puerta abierta.

metro_ab17.jpg Letrero leído con distancia = 0
Puerta abierta.

metro_ab18.jpg Letrero leído con distancia = 7
Puerta abierta.

metro_ab19.jpg Letrero leído con distancia = 6
Puerta abierta.

metro_ab20.jpg Letrero leído con distancia = 6
Puerta abierta.

metro_ab21.jpg Letrero leído con distancia = 3
Puerta abierta.

metro_ab22.jpg Letrero leído con distancia = 7
Estado indeterminado.

metro_ab23.jpg Letrero leído con distancia = 3
Puerta abierta.

metro_ab24.jpg No se encontró letrero.

metro_ab25.jpg No se encontró letrero.

metro_ab26.jpg Letrero leído con distancia = 5
Puerta abierta.

metro_ab27.jpg Letrero leído con distancia = 2
Puerta abierta.

metro_cer01.jpg No se encontró letrero.

metro_cer02.jpg Letrero leído con distancia = 12
Puerta cerrada.

metro_cer03.jpg Letrero leído con distancia = 14
Puerta cerrada.

metro_cer04.jpg Letrero leído con distancia = 0
Puerta cerrada.

metro_cer05.jpg Letrero leído con distancia = 6
Estado indeterminado.

metro_cer06.jpg Letrero leído con distancia = 2
Puerta cerrada.

metro_cer07.jpg Letrero leído con distancia = 5
Puerta cerrada.

metro_cer08.jpg Letrero leído con distancia = 13
Puerta cerrada.

metro_cer09.jpg Letrero leído con distancia = 4
Estado indeterminado.

metro_cer10.jpg Letrero leído con distancia = 6
Puerta cerrada.

metro_cer11.jpg Letrero leído con distancia = 6
Puerta cerrada.

metro_cer12.jpg Letrero leído con distancia = 4
Puerta cerrada.

metro_cer13.jpg Letrero leído con distancia = 13
Puerta cerrada.

metro_cer14.jpg Letrero leído con distancia = 2
Puerta abierta.

metro_cer15.jpg No se encontró letrero.

metro_cer16.jpg No se encontró letrero.

metro_cer17.jpg Letrero leído con distancia = 0
Puerta cerrada.

metro_cer18.jpg Letrero leído con distancia = 9
Puerta cerrada.

metro_cer19.jpg Letrero leído con distancia = 14
Puerta cerrada.

metro_cer20.jpg Letrero leído con distancia = 11
Puerta cerrada.

metro_cer21.jpg No se encontró letrero.

metro_cer22.jpg Letrero leído con distancia = 7
Puerta abierta.

metro_cer23.jpg Letrero leído con distancia = 4
Puerta cerrada.

metro_cer24.jpg Letrero leído con distancia = 9
Puerta cerrada.

metro_cer25.jpg Letrero leído con distancia = 11
Puerta abierta.

metro_cer26.jpg Letrero leído con distancia = 4
Puerta cerrada.

metro_cer27.jpg No se encontró letrero.

metro_cer28.jpg No se encontró letrero.

metro_cer29.jpg Letrero leído con distancia = 5
Puerta abierta.

metro_cer30.jpg Letrero leído con distancia = 10
Estado indeterminado.

metro_cer31.jpg Letrero leído con distancia = 3
Puerta cerrada.

metro_cer32.jpg Letrero leído con distancia = 6
Estado indeterminado.

metro_cer34.jpg Letrero leído con distancia = 5
Puerta cerrada.

metro_cer35.jpg Letrero leído con distancia = 10
Estado indeterminado.

metro_cer36.jpg Letrero leído con distancia = 6
Puerta cerrada.

48/62

La aplicación ha conseguido detectar 48 puertas de 62 imágenes. De estas 48 puertas, ha acertado 38 correctamente, 7 de los 10 errores de estado han sido establecidos como indeterminados. Ahora vamos a ver los casos más destacables.

Figura 52. Imagen inclinada.

En la imagen, la aplicación ha sido capaz de detectar la puerta, y ha detectado el letrero con una distancia de edición de 12. Esto puede ser por la distancia a la que se encuentra de la imagen. Aun estando inclinada la imagen, ha sido capaz de detectar el letrero.

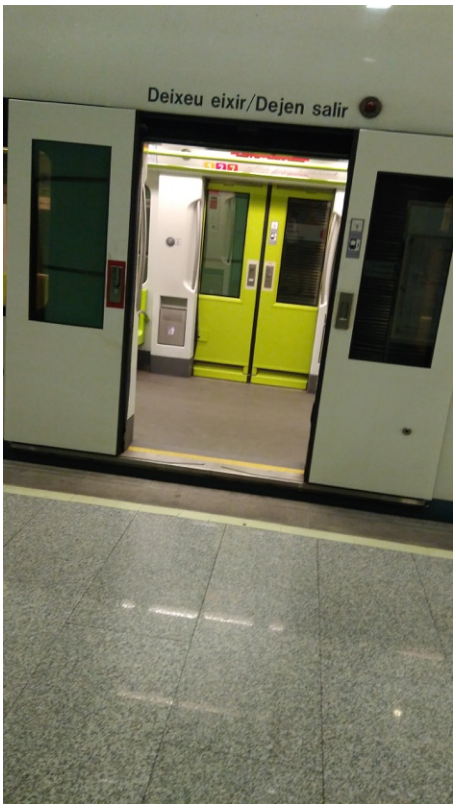

Figura 53. Puerta inclinada.

Esta imagen no ha sido detectada ya que, está demasiado inclinada y, aunque la iluminación del letrero sea correcta, el contorno se juntará con el de la puerta o con algún otro contorno más grande.

Figura 54. Imagen lejana.

Esta imagen no ha sido detectada correctamente, la cámara se encuentra bastante lejos del metro, lo que hace que las letras sean demasiado pequeñas. Además, al estar tan alejada, aparecen 2 puertas en la imagen, esto podría crear confusión si la aplicación fuera capaz de detectar ambas puertas.

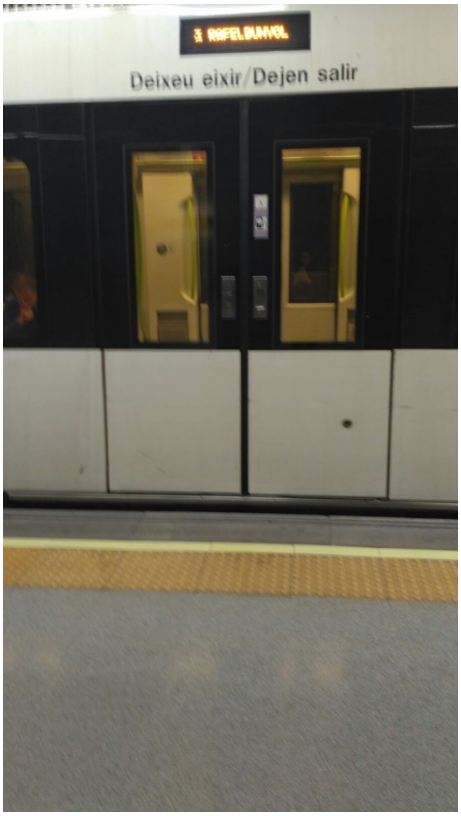

Figura 55. Imagen borrosa.

En esta imagen, la aplicación ha sido capaz de detectar la puerta con una distancia de edición de 14, pese a que las letras están borrosas. Ha determinado correctamente que la puerta se encuentra cerrada.

Figura 56. Letrero cortado.

En esta imagen, la aplicación ha localizado la puerta con una distancia de edición de 6. No obstante, como algunas letras del letrero aparecen cortadas, no ha sido capaz de localizar el ancho total de la puerta, por lo que ha podido determinar de manera errónea que la puerta está abierta.

Figura 57. Puerta de metro abierta.

En este caso, la aplicación ha sido capaz de localizar el letrero, con una distancia de edición igual a 0, pese a la mala iluminación. Sin embargo, ha determinado que la puerta se encuentra cerrada. Esto puede pasar porque se haya seleccionado el borde de la puerta contraria, el cual se encuentra en la parte central del letrero, como borde de la puerta que nos interesa.

En el ejemplo 2 (Figura 51), hemos podido ver una situación similar, en la que como las puertas estaban abiertas, se podía ver la puerta del lado contrario. Una posible solución podría ser recortar los laterales de la imagen con un margen más amplio que el ancho del letrero. Este margen debería de estar relacionado con la distancia a la que estamos del vagón, para que, en caso de captar la imagen desde lejos, no se capten líneas negras de otros objetos que puedan interferir.

4. Conclusiones

A la vista de los resultados obtenidos, se puede llegar a concluir que el funcionamiento de la aplicación es bastante positivo. El porcentaje de acierto en la localización de las puertas es del 77% y el porcentaje de acierto del estado de las puertas es del 80%.

Los resultados se pueden valorar de manera más positiva, si tenemos en cuenta que muchas de las imágenes de la galería no se han detectado porque las puertas se encontraban lejos, inclinadas o porque las letras estaban borrosas.

Estos problemas no serán tan graves en la aplicación cuando se trabaje a tiempo real, ya que, de esta manera, se estarán tomando imágenes muy seguidas, por lo que, al acercarse, las letras se irán viendo mejor y llegará un punto en que la aplicación detecte la puerta.

Al implementarse utilizando el letrero como referencia y utilizando librerías de reconocimiento óptico de caracteres, se ha evitado tener que realizar un estudio morfológico con el que determinar las características de cada modelo de vagón de MetroValencia. De esta manera, solo tenemos que buscar unas características que son comunes en los vagones.

La aplicación se podría continuar desarrollando, y se podría crear una aplicación para Android o iOS, que fuera capaz de trabajar a tiempo real. Para la implementación de estas aplicaciones, sería interesante contactar con personas invidentes para estudiar la manera más cómoda de que la aplicación les avise si ha encontrado la puerta.

Una posible implementación, sería avisar mediante sonidos de la ubicación de la puerta. En el caso de que el usuario utilizara auriculares, el sonido podría ser estéreo para que, en caso de que la puerta este en un lateral, como por ejemplo el lado derecho, el sonido provenga del auricular derecho.

También tendría que avisar del estado en el que se encuentra la puerta, para ello podría añadirse un mensaje de audio que indique el estado, o se podrían utilizar diferentes tonos para el sonido de localización de la puerta dependiendo del estado.

Además, con el acelerómetro del teléfono móvil, se podría indicar a la persona si tiene el teléfono demasiado inclinado tanto lateral como verticalmente.

Personalmente, estoy satisfecho con los resultados obtenidos y con la manera en que se ha desarrollado la aplicación. He podido ampliar conocimientos relacionados con el tratamiento de imágenes y he adquirido nuevos conocimientos de programación (Tesseract OCR, el concepto de distancia de edición, el uso de constructores, etc.).

Me ha servido para conocer las barreras con las que se encuentran las personas invidentes, lo que me ha motivado, no solo a empezar a desarrollar la aplicación, sino también a seguir desarrollándola en un futuro.

5. Bibliografía

[1] Chieko Asakawa. (2015, octubre). How new technology helps blind people explore the world [Archivo de video]. Recuperado de https://www.ted.com/talks/chieko_asakawa_how_new_technology_helps_blind_people_explore_the_world?utm_campaign=tedspread--b&utm_medium=referral&utm_source=tedcomshare

[2] Bradski, Gary R & Adrian, Kaehler (2008), *Learning OpenCV*, Sebastopol, California, Estados Unidos de America: O'Reilly Media, Inc.

[3] Documentación librería OpenCV. <http://docs.opencv.org/2.4/index.html> [Consulta: enero – septiembre de 2017]

[4] Smith, R. (2007). "An Overview of the Tesseract OCR Engine" en *IEEE Computer Society,* vol. 2, p. 629-633

[5] Documentación librería Edlib. <https://github.com/Martinsos/edlib> [Consulta: abril – septiembre de 2017]

[6] Green, B. (2002). Canny Edge Detection Tutorial: <http://masters.donntu.org/2010/fknt/chudovskaja/library/article5.htm> [Consulta: septiembre de 2017]