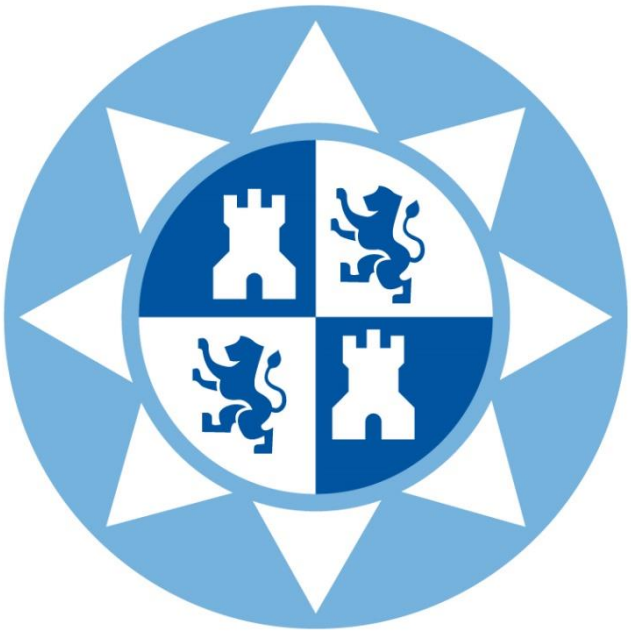
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA



Proyecto Fin de Carrera

Sistema de reconocimiento de matrículas y gestión de acceso en una plataforma embebida

AUTOR: Valentín Tomás Rubio
DIRECTOR: Esteban Egea López
DIRECTOR: Rafael Verdú Monedero
Junio / 2015

Índice de contenido

Índice de imágenes

Prólogo

Hay personas: compañeros, ingenieros profesionalmente consolidados y seguramente hasta ciertos docentes, que consideran la realización del proyecto final de carrera como un trámite casi inservible más allá de la obtención definitiva de la titulación. Yo, personalmente, creo que hay un objetivo mucho más constructivo y profundo en ello. Creo que supone una oportunidad única de demostrar, principalmente a ti mismo, pero también a la universidad, familiares, amigos, compañeros y, por qué no, empresas de lo que eres capaz de hacer como ingeniero nobel a partir de una idea o de un encargo concreto.

No deberíamos pasar por alto que se trata del último y, a la vez, primer escalón de un camino que ha centrado los últimos cuatro, cinco e incluso ocho y diez años de nuestra vida y que marcará el resto de la misma. Acabará resultando casi anecdótico el tiempo y esfuerzo dedicado en esto, pero creo que simbólicamente es un punto clave en nuestra historia.

Mi historia hasta llegar aquí la protagonizan las personas que de una forma u otra me han traído a éste sitio. Mis padres, Valentín y Dolores, que tras haberme hecho posible a mí, han hecho todo lo posible y, a veces, lo imposible para que fuera capaz de aspirar a todo lo que hoy tengo un poco más cerca. Inculcándome valores como el esfuerzo, la dedicación, el respeto y dejándome siempre claro que, con esos valores, yo sería el único capaz de decir hasta donde puedo llegar.

Si tener una madre como la que yo tengo es ya un privilegio, además, tengo la gran fortuna de tener una segunda tan buena como la primera, se trata de mi abuela Isabel. Y unos hermanos, Irene y Guillermo, que no cambio por ningunos otros.

Podría faltarme espacio y tiempo para hablar de los familiares, amigos y compañeros que siempre han estado ahí o que estuvieron y, por cualquier motivo, ya no están. Pero no puedo dejar de mencionar a Juan Pedro y Alfredo, mis hermanos de distinta madre, así como José Antonio Villanueva y la Kinta de la residencia Alberto Colao.

Un especial agradecimiento merecen profesores como Rafael Verdú, Esteban Egea (ambos directores del proyecto), Jorge Larrey, Ricardo González o Javier Toledo, que luchan en primera línea por formar ingenieros que puedan aspirar a ser, algún día, tan buenos profesionales como lo son ellos.

Cabe ahora hacer mención a todo el que en algún momento pensó que no sería capaz de conseguirlo, porque sin esa motivación seguro que no lo hubiera hecho.

Y, para terminar, levantar la vista al cielo para dedicarle un día como hoy a mi abuelo Fulgencio que, allá donde esté, me estará viendo con orgullo. Gracias.

1. Introducción

El presente proyecto se presenta como una respuesta a la ya extendida y demandada automatización de todos los procesos que engloba la actividad industrial y comercial. Centrando el objeto del mismo en la necesidad real que supone un control de acceso vehicular eficiente y eficaz a un complejo cerrado, tanto desde el punto de vista de seguridad, como el del desarrollo de la actividad misma.

La tendencia, basada en dicha necesidad, apuesta por las soluciones de ingeniería computacional que, además de cumplir las premisas de eficiencia y eficacia, permitan la reducción del recurso humano en éstas tareas, pudiendo así enfocar los esfuerzos y recursos en otros fines con mayor repercusión en la actividad económica generada.

La principal problemática asociada a éste tipo de soluciones se encuentra en lo elevado de su coste inicial que, desde el planteamiento hasta su puesta en marcha y consiguiente implantación, puede alcanzar precios prohibitivos para muchas *pymes* que, con una presencia mayoritaria, forman el tejido empresarial de este país.

A pesar de la opacidad existente a la hora de acceder a los presupuestos asociados a los sistemas tradicionales de control de acceso vehicular, es fácil llegar a la conclusión de lo costoso de los mismos sólo por el despliegue que suponen. Esta premisa es aplicable tanto si se trata de: sistemas basados en "*tags*" activados por radiofrecuencia, sistemas de visión por computador tradicionales con procesado local o por parte de un servidor, los basados en tarjetas magnéticas de acceso, etc...

De los sistemas tradicionales expuestos cabe resaltar otros factores en contra como pueden ser: la degradación de los elementos de identificación, la posible pérdida o sustracción de dichos elementos (con el fin de llevar a cabo una suplantación de identidad y la consiguiente violación de la seguridad) o, más comúnmente, un tráfico congestionado en los accesos por la necesidad de la actuación humana en los procedimientos de identificación.

Partiendo de las premisas expuestas y de la problemática derivada de las tecnologías utilizadas hasta ahora, se deduce la necesidad de una solución que mantenga un compromiso entre el cumplimiento de las especificaciones y con un coste que, a su vez, resulte asumible para las empresas que decidan implementar este tipo de sistemas. Ese es, por tanto, el principal objetivo del proyecto que se presenta en este documento.

Profundizando algo más en este objetivo general (tan técnicamente correcto y conocido, como ambiguo), se establecen las siguientes condiciones como objetivos a priori para considerar que la solución cumple el compromiso pactado:

- Un **alto porcentaje de eficacia** a la hora de reconocer el elemento de identificación elegido. Es decir, que el sistema sea capaz de reconocer correctamente dicho elemento con el menor número de intentos posible.

- Un **alto nivel de eficiencia** en la tarea que se le encomienda. Es decir, que éste sea capaz de realizar dicha tarea en el menor tiempo y con el menor volumen de recursos posible.

- Un **bajo coste** para el desarrollo, despliegue, puesta en funcionamiento y mantenimiento del sistema.

En base a tales objetivos se propone como solución la implementación de un sistema de reconocimiento de matrículas mediante la utilización de la librería OpenCV con procesamiento local embebido sobre un dispositivo Raspberry-Pi (R-Pi) para la gestión del control de acceso a un recinto cerrado.

Los motivos que llevan a elegir dicha solución se deducen directamente de los objetivos marcados y de los inconvenientes encontrados en las tecnologías tradicionales utilizadas para este fin.

Con la idea de eliminar la interacción humana en el proceso, se apuesta por el reconocimiento de un elemento difícilmente sustraible y controlado por un organismo público, es decir, la Dirección General de Tráfico, como son las placas identificativas del vehículo. Además, dicho organismo prevé mediante el código de circulación la obligación de que cualquier vehículo se encuentre correctamente matriculado, con una codificación única y que ésta se encuentre en un estado en el que sea perfectamente legible.

Otra consecuencia directa de la elección de la matrícula del vehículo como elemento de identificación es que no resulta necesaria la adquisición y renovación, por parte de la empresa, de dichos elementos y empieza suponiendo un ahorro en coste respecto a soluciones como las basadas en “*tags*” RF o tarjetas magnéticas.

La opción concreta de la librería OpenCV, así como del dispositivo R-Pi, deriva directamente del objetivo económico dado que ambas poseen licencias del tipo *Berkeley Software Distribution* (BSD), la cual está pensada para aplicaciones de dominio público y gratuito. En el caso concreto de R-Pi se encuentra la mayor mejoría en coste respecto a los sistemas tradicionales, ya que además de tener soporte, e incluso distribuciones propias para Linux (gratuito y de código abierto), posee la capacidad de procesado necesaria en este tipo de aplicaciones y su coste es muy bajo.

Finalmente, los motivos expuestos llevan a decantarse por el lenguaje de programación C/C++ para el desarrollo de la misma ya que, además de ser uno de los utilizados por la librería elegida (junto con Python), permite un nivel relativamente bajo de

abstracción y, de esa forma, una mejor optimización e implantación del sistema software en el dispositivo elegido.

Fuera de la parte encargada puramente del procesado, como cualquier aplicación que se precie, se hace fundamental una localización para la gestión y registro (y consultas) de los datos obtenidos (y esperados) y, para ello, una base de datos que se encargue de dicha tarea.

Continuando en la línea de elecciones seguida en las tecnologías de procesado, ésta se decanta por una solución mediante un servidor MySQL, que aloja una base de datos y mediante una comunicación HTTP a través de *scripts* PHP (usando funciones de la librería cURL), permite la transferencia de información entre el dispositivo local y el servidor de forma eficiente y segura.

En los siguiente capítulos y secciones de la presente memoria se irán desgranando las distintas fases de las que consta el proyecto, desde la investigación de los detalles la solución introducida hasta la comprobación del cumplimiento de los objetivos fijados a priori, pasando por el análisis, diseño e implementación de la misma. Se pretende así, dar al lector una visión detallada y con un paralelismo temporal a lo que ha supuesto la realización del proyecto, como si de un diario se tratase.

2. Estado del Arte

Este apartado recoge la investigación realizada en los distintos campos que engloba la solución seleccionada y pretende dar una visión concreta de la misma, tanto de las herramientas como de los procedimientos con los que se pretende llevarla a cabo.

2.1. Visión por computador

Se puede definir la “Visión por computador” (o Visión artificial) como un campo de la “Inteligencia Artificial” que, mediante la utilización de las técnicas adecuadas, permite la obtención, procesamiento y análisis de cualquier tipo de información especial obtenida a través de imágenes digitales. [1]

En un sistema de Visión por Computador (CV, por sus iniciales en inglés), deseamos obtener a partir de los estímulos (los elementos de la imagen digital) una representación de la realidad externa que refleje ciertas propiedades de interés: típicamente el reconocimiento de objetos, o clases de objetos, o algún modelo de la 'estructura espacial' del espacio circundante. [2]

Para llevar a cabo el objetivo, se hace uso de las técnicas que engloba el “Procesamiento de Imágenes Digitales” y se detallan en profundidad en [3]. Se realizará un acercamiento del lector a las técnicas y definiciones necesarias para la correcta comprensión de este trabajo.

2.1.1. Introducción al procesamiento digital de imágenes

El procesamiento digital de imágenes abarca el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad, hacer más evidentes en ellas ciertos detalles que se desean hacer notar o facilitar la búsqueda de información dentro de las imágenes. La imagen puede haber sido generada de muchas maneras, por ejemplo, fotográficamente, o electrónicamente, por medio de monitores de televisión. El

procesamiento de las imágenes se puede llevar a cabo por medio de métodos ópticos, o bien por medio de métodos digitales en una computadora. [4]

2.1.2. Representación y definición de una imagen digital

Una imagen digital puede ser definida como una función de dos dimensiones, *f(x, y)*, donde *x* e *y* son las coordenadas espaciales y el valor de *f* en un punto *(x, y)* es llamado **intensidad**, valor o nivel de gris de la imagen en ese punto.

La imagen digital se puede considerar como una matriz cuyos índices de filas y columnas identifican un punto de la imagen y el valor del elemento correspondiente. Los miembros de una distribución digital de este tipo se denominan elementos de la imagen o más comúnmente **píxeles** (que proviene del término inglés *"picture element"*).

Por defecto, la representación espacial de un píxel con coordenadas *(0,0)* se localiza en la esquina superior izquierda de la imagen. En nuestro caso el valor de *x* se incrementa de izquierda a derecha y el valor de *y* de arriba hacia abajo. [3]

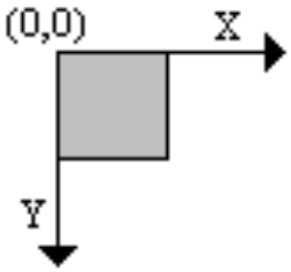


Figura 1: Representación espacial de un píxel

2.1.3. Propiedades de las imágenes digitales

Hay tres conceptos íntimamente relacionados con una imagen digital: la resolución espacial, la resolución de niveles de gris y el número de canales.

2.1.4. Resolución espacial

La resolución espacial de una imagen es el número de píxeles por fila y por columna. Una imagen de *m* filas y *n* columnas tiene un total de *n* x *m* píxeles.

La resolución espacial se relaciona con los detalles que pueden hacerse visibles en una imagen: mientras mayor sea la resolución espacial, menor será el área representada por cada píxel en una imagen digital y mayores serán los detalles que pueden ser apreciados en la misma. El píxel representa el detalle más pequeño diferenciable en una imagen. [3]

2.1.5. Resolución de niveles

La resolución de niveles de una imagen digital, también llamada profundidad del píxel, indica el número de niveles de gris que pueden verse en la imagen. La profundidad del píxel es el número de bits usado para definir la intensidad que representa. Para una cantidad de bits *n*, el píxel puede tomar 2^n valores diferentes. Por ejemplo, si *n* es igual a 8 bits, un píxel puede tomar 256 valores distintos en el rango de 0 a 255. Representa el cambio más pequeño discernible en los niveles de gris que conforman la imagen.

2.1.6. Canales de una imagen

El número de canales en una imagen es el número de matrices de píxeles que la componen. Una imagen en tonos de gris está compuesta de un solo plano (en realidad son tres planos, pero iguales), mientras una imagen de color verdadero ("*true color*") está compuesta por tres planos: el de la componente roja, la verde y la azul (en el caso de RGB) o matiz, saturación y valor (en el caso de HSV).

Imágenes en tonos de gris

Si las correspondientes intensidades de la componente roja, verde y azul de cada píxel en una imagen son iguales, se forma una imagen (monocromática) en tonos o niveles de gris.

Una imagen en tonos (o niveles) de gris es una imagen bidimensional donde cada píxel sólo representa un valor de intensidad acotado entre 0 y 2^{n-1}, donde *n* es la cantidad de bits utilizados para representar cada uno de los valores de intensidad. Por convenio los valores extremos de este rango representan el negro y el blanco respectivamente.

Una imagen en niveles de gris se obtiene promediando los valores de las tres componentes R, G y B de todos los píxeles de la imagen en color.

Formalmente, una imagen digital en niveles de gris es una función bidimensional de la intensidad de luz *f: ZxZ→Z* cuyos valores se han obtenido muestreando la intensidad sobre una retícula rectangular. Por lo tanto, una imagen digital la denotaremos como *f(x, y)*, donde *x* e *y* son las coordenadas espaciales y el valor de *f* en cada punto *(x, y)* es proporcional a la intensidad de luz (nivel de gris) de ese punto.

Podemos decir que una imagen *f(x, y)* está formada por dos componentes: una es la cantidad de luz incidente en la escena y la otra es la cantidad de luz reflejada por los objetos. Estas dos componentes se llaman: iluminación, que denotaremos por *i(x, y)* y reflectancia, que denotaremos por *r(x, y)*. [6]

Imágenes binarias

Los píxeles en una imagen binaria contienen solo dos valores de intensidad en forma normalizada: 0 para el negro y 1 para el blanco, los que equivalen a los niveles 0 y 255, respectivamente, por estar cada píxel asociado a un byte. [3]

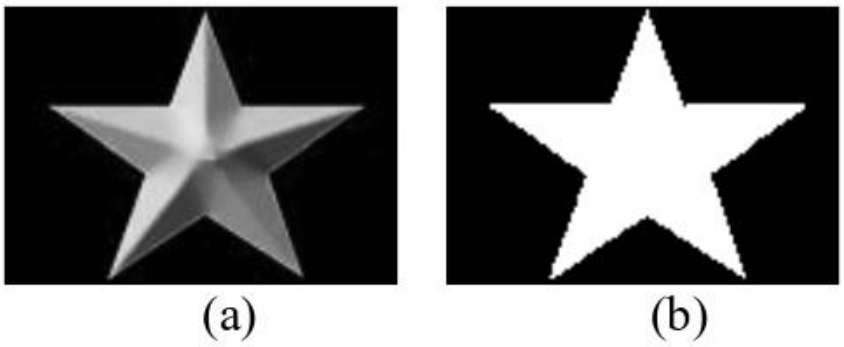


Figura 2: a) Imagen en nivel de gris; b) Imagen binaria

2.1.7. Relaciones de vecindad, distancias y conectividad

En una imagen *f(x, y)*, denotaremos los píxeles que la componen por letras minúsculas *p* o *q*, y denotaremos por *S* al conjunto de todos ellos.

Vecinos de un píxel

Un punto *p* de coordenadas *(x, y)* tiene 4 vecinos verticales y horizontales cuyas coordenadas son: *(x + 1, y)*, *(x – 1, y)*, *(x, y + 1)*, *(x, y - 1)*. A este conjunto de puntos se le llama los 4-vecinos de *p* y lo denotaremos como $N_4(p)$.

Los 4 vecinos diagonales de *p* tienen como coordenadas: *(x + 1, y + 1)*, *(x + 1, y - 1)*, *(x – 1, y + 1)*, *(x – 1, y – 1)* y los denotaremos como $N_D(p)$.

A estos 4 vecinos diagonales junto con $N_4(p)$ les llamaremos los 8-vecinos de *p*, denotado como $N_8(p)$. Gráficamente:

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

Conectividad

Para establecer si dos puntos están conectados debemos determinar si son adyacentes en algún sentido (por ejemplo 4-vecinos) y si su nivel de gris satisface algún criterio de similitud. Sea *V* el conjunto de niveles de gris usados para definir la conectividad.

Consideramos dos tipos de conectividad:

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

Medidas de distancia

Dados los puntos *p*, *q*, con coordenadas *(x, y)*, *(s, t)*, respectivamente, se definen entre otras los siguientes tipos de distancia:

Distancia Euclídea (o Euclidiana)

Se define la distancia Euclídea entre dos puntos *p* y *q* como:

$$D(p,q) = \sqrt{(x-s)^2 + (y-t)^2}$$

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

Distancia D4

Se define la distancia *D4* entre dos puntos *p* y *q* como:

$$D_4(p,q) = |x-s| + |y-t|$$

Los puntos que están a una distancia *D4* de *(x, y)* igual o menor a un valor *r* forman un diamante (o rombo) centrado en *(x, y)*.

Distancia D8

Se define la distancia *D8* entre dos puntos *p* y *q* como:

$$D_8(p,q) = máx(|x-s|, |y-t|)$$

Los puntos que están a una distancia *D8* de *(x, y)* igual o menor a un valor *r* forman un cuadrado centrado en *(x, y)*. [6]

2.1.8. Operaciones lógicas y aritméticas

Operaciones lógicas

Se pueden realizar operaciones lógicas *AND*, *OR*, *XOR*, etc. con las imágenes discretas, de tal forma que se puede, mediante máscaras y un proceso de convolución,

cubrir ciertas áreas de una imagen, efectuar comparaciones, etc.[3] Las operaciones se llevan a cabo píxel a píxel.Para definirlas, sean I_1 e I_2 dos imágenes binarias cualesquiera.

Intersección (*AND*): Sobre las imágenes binarias, la operación respeta la lógica convencional y se define de la siguiente manera:

$$O(x,y) = I_1(x,y)\ and\ I_2(x,y)$$

Para imágenes en niveles de gris, se toma el mínimo valor de los píxeles entre los cuales se realiza la operación; es definida de la siguiente manera:

$$O(x,y) = \min\{I_1(x,y), I_2(x,y)\}$$

Unión (*OR*): Para imágenes binarias, la operación *OR* se comporta de igual forma a la lógica convencional. Para dos imágenes I_1 e I_2 se define de la siguiente manera:

$$O(x,y) = I_1(x,y)\ or\ I_2(x,y)$$

Para imágenes en niveles de gris, la operación toma el máximo valor de los píxeles involucrados en la operación, por lo que se define:

$$O(x,y) = \text{máx}\{I_1(x,y), I_2(x,y)\}$$

Complemento (*NOT*): para imágenes binarias, podemos definir esta operación *O(x, y)* de una imagen de entrada *I(x, y)* de la siguiente manera:

$$O(x,y) = not\{I(x,y)\}$$

Por su parte, en escala de grises el resultado de esta operación (complemento) es igual a la diferencia del máximo valor posible que puede alcanzar un píxel menos el valor del píxel involucrado, definiéndose por lo tanto de la siguiente manera:

$$O(x,y) = (2^n - 1) - I(x,y)$$

Operaciones aritméticas

Las operaciones aritméticas implican dos imágenes y se efectúan píxel a píxel de la primera imagen con la segunda. Las operaciones más comunes son la suma y la resta. Las operaciones aritméticas son relativamente rápidas, pues tan solo se han de realizar *NxM* operaciones, donde *N* es el ancho y *M* es el alto de la imagen en píxeles. [3]

Suma (Resta): Podemos definir la suma (resta) de dos imágenes I_1 e I_2 de la siguiente manera:

$$S(x,y) = I_1(x,y) \pm I_2(x,y)$$

También podemos sumar (restar) una constante *C* a una imagen:

$$S(x,y) = I(x,y) \pm C$$

Producto (División): También podemos multiplicar (dividir) una constante *C* a una imagen:

$$S(x,y) = I(x,y) \cdot C$$

$$S(x,y) = I(x,y)/C$$

La imagen de salida dependerá de la implementación. Puesto que pueden presentarse problemas al obtenerse píxeles fuera del rango dinámico posible, en general hay que ajustar previamente el comportamiento deseado de la operación. Este puede ser limitado, cíclico o escalado.

La diferencia de dos imágenes es útil para detectar cambios producidos en la misma escena después de un determinado intervalo de tiempo, o para eliminar defectos de captura, entre otras aplicaciones.

2.1.9. Transformaciones geométricas

Estos algoritmos modifican las características geométricas de las imágenes, su uso fundamentalmente se aplica en la reconstrucción de imágenes deformadas, el giro y ajuste de las mismas o la deformación intencionada de ciertos rasgos para posteriores análisis.

Todos los algoritmos correspondientes a transformaciones geométricas se basan en realizar una nueva distribución de los píxeles según lo que se pretenda. De esta forma el proceso de transformación geométrica se fundamenta en:

Determinar las nuevas coordenadas de cada píxel *(x,y)* en la rejilla transformada, *(x',y')*. Estas nuevas coordenadas *(x',y')* generalmente, no serán valores enteros y será necesario un proceso de interpolación. Este proceso depende del tipo de transformación a realizar.

Una vez obtenidos *(x',y')* hay que calcular los valores de los píxeles *(x,y)* en la rejilla destino. Este proceso es común a todas las transformaciones y se denomina interpolación. La interpolación más utilizada es la interpolación bilineal. [7]

Traslación o desplazamiento

Consiste en sustituir cada píxel por el correspondiente a sus coordenadas más el desplazamiento en cada dirección *k* y *l*.

Si los desplazamientos son enteros para ambas direcciones el proceso consiste en sustituir cada píxel *(x,y)* por su píxel *(x+k, y+l)*, mientras que si tienen parte decimal es necesario realizar una interpolación.

Resumiendo, la función de translación sería:

$$O(x, y) = I(x + k, y + l)$$

En las aplicaciones prácticas el proceso de translación o desplazamiento se utiliza cuando se quiere posicionar cierto objeto detectado en un punto determinado para realizarle posteriores procesos, como por ejemplo: el uso de funciones lógicas AND, OR, XOR, etc.; con máscaras para obtener o eliminar ciertas partes de éste, procesos de unión o fusión con otras imágenes, etc.

Rotación o giro

Los algoritmos de giro son generalmente los más complejos y por lo tanto los más costosos en tiempo de procesado. Debido a esto, sólo se utilizan cuando es posible obtener una posición de giro que simplifique procesos posteriores.

Dado un punto *p(x,y)* rotado *θ* grados, las coordenadas *x'* y *y'* del nuevo punto serán:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

Por lo tanto, la versión girada de la imagen principal será:

$$O(x', y') = O(x \cdot \cos\theta - y \cdot \sin\theta, x \cdot \sin\theta + y \cdot \cos\theta) = I(x, y)$$

Después será necesario realizar una interpolación.

Por ejemplo, si se tiene una imagen formada por múltiples rectángulos y cuadrados situados de forma perpendicular entre ellos y la imagen está girada, es muy útil alinearlos respecto a los ejes *x* e *y* de la imagen. De esta forma los procesos posteriores de segmentación y análisis sobre estos rectángulos o líneas serán más rápidos y eficaces.

Escalado y zoom

Los algoritmos de escalado permiten reducir o aumentar la imagen, así como realizar el zoom de ciertas partes de la imagen.

Dado un factor de escalado *α* para la dimensión *x* y otro *β* para la dimensión *y,* obtendríamos:

$$O(x, y) = I(\alpha \cdot x, \beta \cdot y)$$

Requiriendo una interpolación cuando *x/α* o *y/β* no son enteros.

Existen procesos (erosionados, reducción de la resolución, etc.) que no necesitan tanta cantidad de información, reduciéndose la imagen para que esos procesos sean más rápidos ya que tratan con una imagen de tamaño menor.

Reducción de la resolución

Actualmente, existen técnicas que utilizan redes neuronales para el reconocimiento de objetos mediante el análisis de varias resoluciones de una misma imagen. Estas técnicas se basan en reconocer un objeto previo comparando la imagen de este objeto en diferentes resoluciones a la vez, para ellos se usan redes neuronales que relacionan unas con otras.

Estos algoritmos obtienen de una imagen de *MxN* píxeles otras del mismo tamaño, donde se han reducido sus resoluciones, a un 1/2, 1/4, etc.

El procedimiento es muy sencillo, por ejemplo para reducir la resolución a la mitad consiste en coger cuatro píxeles adyacentes, calcular su media e igualarlos todos a este resultado.

Normalización en tamaño de las imágenes

La normalización consiste en llevar a un tamaño estándar las dimensiones de una imagen, sin provocar en ella distorsión alguna. [3]

En morfología, debido a que los operadores y filtros morfológicos trabajan comúnmente con elementos estructurantes de un tamaño y forma bien definidos, se requiere que todas las imágenes posean el mismo tamaño relativo a un valor preestablecido con el objetivo de hacer mediciones en una imagen y poder basarse en dichas mediciones para detectar los patrones presentes en la misma.

2.1.10. Etapas del análisis digital de imágenes

La primera etapa en el análisis de imágenes la constituye la **adquisición** de la imagen. Para ello se requieren los sensores adecuados que pueden ser una cámara de color o monocromática, o un escáner. Si la imagen de salida de la cámara no está en formato digital, es necesario usar un conversor analógico-digital para digitalizarla. El escáner, por su diseño, ya entrega directamente una imagen digital.

Una vez obtenida la imagen en forma digital, la etapa siguiente consiste en la de **preprocesamiento**, si fuese necesario. El preprocesamiento puede consistir en mejorar el contraste, suprimir el ruido, modificar el brillo, su tamaño, etc. Hay que señalar que el procesado previo de la imagen depende de cuál sea el objetivo final que se quiere lograr al analizar la imagen, por lo que una misma imagen puede requerir varios tipos de preprocesados.

La tercera etapa es la **segmentación**. Su objetivo es dividir la imagen en las partes que la constituyen o en los objetos que la forman y el fondo. La salida del proceso de segmentación son imágenes que contienen la frontera de las regiones de interés, o los píxeles que conforman la región misma.

La imagen segmentada es procesada más tarde en un proceso denominado de **descripción y representación**, que constituye la cuarta etapa. La descripción está dirigida a extraer los rasgos de los objetos que diferencian una clase de objetos de otras. La representación le asigna una etiqueta a cada objeto basándose en la información numérica que proporcionan los descriptores.

En último lugar se encuentra la fase de **análisis** que consta, a su vez, de reconocimiento e interpretación. El **reconocimiento** consiste en el proceso que asigna una etiqueta a cada objeto basándose en la información que proporcionan los descriptores, mientras que la **interpretación** lleva a asignar significado al conjunto de objetos reconocidos. [7]

El proceso relatado se puede ilustrar con el siguiente diagrama de bloques:

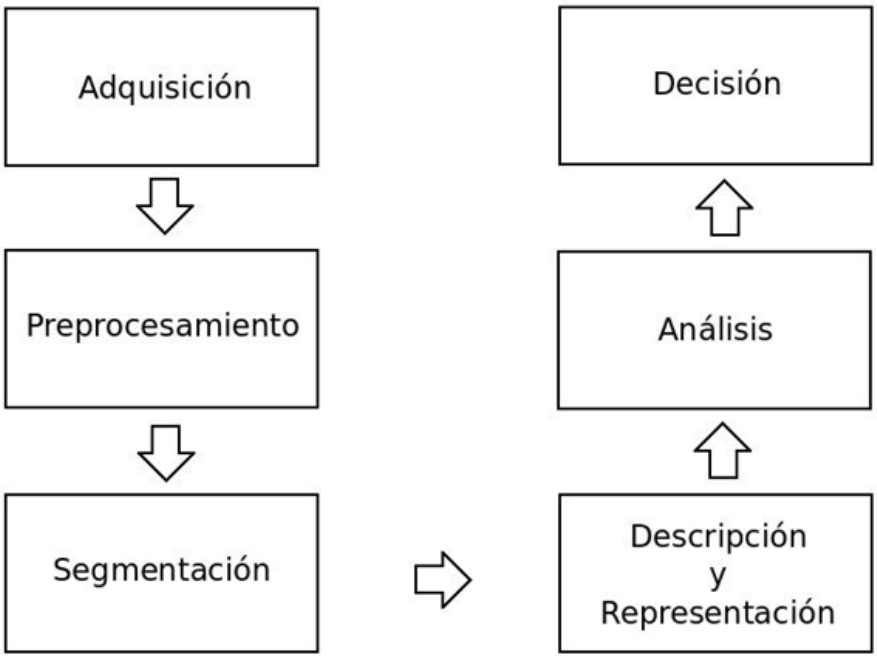


Figura 3: Etapas del análisis digital de imágenes

2.1.11. Histograma de una imagen

El histograma de la imagen digital es una representación estadística que representa la probabilidad con que un determinado nivel de gris aparece en la imagen. Se representa por el número de píxeles que tienen el mismo nivel de gris dentro del rango dinámico de la imagen.

En general se representa como un gráfico de barras en el que las abscisas son los distintos niveles de gris (o colores *RGB*) de la imagen y las ordenadas la frecuencia relativa con la que cada nivel de intensidad aparece en la misma. El histograma proporciona información global sobre el brillo y el contraste de la imagen.

Una definición más formal del histograma es la siguiente: El histograma de una imagen *f(x, y)* con *L* niveles de intensidad o de gris en el rango *[0, L-1]*, denotado como $h(r_k)$ es una función discreta:

$$h(r_k) = \frac{n_k}{N}$$

Donde r_k es el *k*-ésimo nivel de gris, n_k es el número de píxeles en la imagen con el nivel de intensidad r_k y *N* el número total de píxeles en la imagen. [5]

El histograma proporciona una descripción de la apariencia global de una imagen. Si los niveles de gris están concentrados hacia el extremo oscuro del rango dinámico, la apariencia global de la imagen será oscura; si sucede justo lo contrario, la imagen correspondiente será brillante. Por su parte, un histograma que presente un perfil

estrecho corresponderá a una imagen de bajo contraste y un histograma con una dispersión considerable de sus niveles de gris corresponderá a una imagen de alto contraste.

En el caso de que la imagen sea en colores, se tendrán tres histogramas, de forma que el tratamiento de imágenes en colores se complicará por la aparición de nuevos componentes. [3] En este caso el histograma no representará el número de píxeles con los tonos del negro al blanco, sino del negro al color correspondiente (rojo, verde o azul para el caso *RGB*).

2.1.12. Métodos de segmentación basados en umbralización

Las diferentes técnicas para segmentar una imagen mediante umbralizado pueden permitir separar un objeto dentro de la imagen del fondo que lo circunda. Se basan en comparar alguna propiedad de la imagen con un umbral fijo o variable: si el valor de la propiedad de un píxel supera el valor del umbral, entonces el píxel pertenece al objeto; en caso contrario, el píxel pertenece al fondo. [7]

Cuando la segmentación se realiza sobre la base del nivel de gris, el valor del nivel de gris de cada píxel debe ser comparado con el umbral para decidir si tal píxel pertenece al objeto o al fondo. La imagen de salida es siempre una imagen binaria en la cual aquellos píxeles cuyo valor es uno, pertenecen al objeto y los píxeles cuyo valor es cero, pertenecen al fondo.

La selección del valor del umbral se realiza generalmente a partir del histograma de la imagen. Si una imagen está compuesta de objetos que aparecen en la escena sobre un fondo más o menos homogéneo, entonces es de esperar que el histograma sea bimodal, es decir, si por ejemplo los objetos son más claros que el fondo, en el histograma aparecerán dos picos (modas), el ubicado en los valores de gris mayores correspondiente al objeto y otro pico para niveles de gris menores, correspondientes al fondo. En la figura 4 se muestra un histograma bimodal, en el cual el umbral se ubica en algún lugar entre los dos picos del histograma.

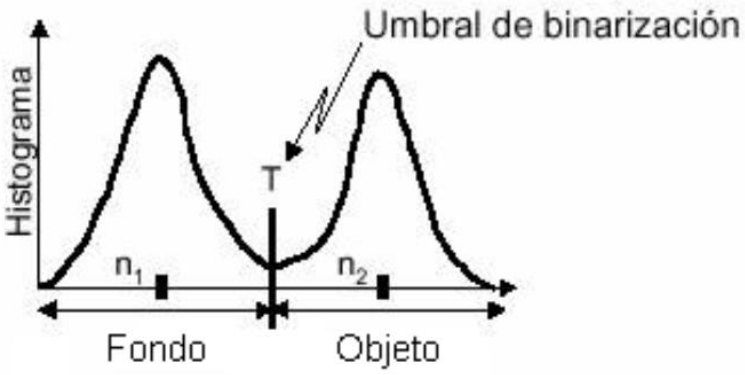


Figura 4: Histograma bimodal con umbral

La selección automática del umbral es un problema difícil, debido a que el histograma no siempre es bimodal, en cuyo caso resulta necesario combinar la información espacial presente en la imagen con la información referente a los niveles de gris. A continuación se describe una técnica para calcular de manera automática el umbral.

Método de umbralizado de Otsu

Este método, aunque se trata de uno de los primeros, sigue siendo usado en muchas aplicaciones para umbralizar automáticamente una imagen. Para su correcto funcionamiento el método de umbralizado de *Otsu* [7] supone que los píxeles de una imagen *f(x,y)* pueden ser separados a través de un umbral *u* (a determinar) en dos clases: C_1, la clase del objeto u objetos de interés, y C_2, la clase de los píxeles del fondo.

El método de *Otsu* se fundamenta en la técnica del análisis discriminante al maximizar alguna medida que permita separar las dos clases: la de los objetos y la del fondo. Una de estas medidas, de acuerdo con el trabajo de *Otsu*, es la siguiente:

$$J_1(u) = \frac{P_1(u)P_2(u)[\mu_1(u) - \mu_2(u)]^2}{P_1(u)\sigma_1^2 + P_2(u)\sigma_2^2}$$

dónde:

$$P_1(u) = P_r(C_1) = \sum_{r=0}^{u} p(r)$$

$$P_2(u) = P_r(C_2) = \sum_{r=u+1}^{L-1} p(r) = 1 - P_1(u)$$

$$\mu_1(u) = \sum_{r=0}^{u} r \cdot P_r(r|C_1) = \frac{1}{P_1(u)} \sum_{r=0}^{u} r \cdot p(r)$$

$$\mu_2(u) = \sum_{r=u+1}^{L-1} r \cdot P_r(r|C_2) = \frac{1}{P_2(u)} \sum_{r=u+1}^{L-1} r \cdot p(r)$$

$$\sigma_1^2 = \sum_{r=0}^{u} (r - \mu_1(u))^2 \cdot P_r(r|C_1) = \frac{1}{P_1(u)} \sum_{r=0}^{u} (r - \mu_1(u))^2 \cdot p(r)$$

$$\sigma_2^2 = \sum_{r=u+1}^{L-1} (r - \mu_2(u))^2 \cdot P_r(r|C_2) = \frac{1}{P_2(u)} \sum_{r=u+1}^{L-1} (r - \mu_2(u))^2 \cdot p(r)$$

Para poder maximizar el criterio dado por la ecuación, las medias de las dos clases deberían estar bastante separadas y las varianzas deberían ser lo más pequeñas posibles. Si esto no sucede, el valor del umbral obtenido simplemente no producirá el resultado deseado. Una imagen con un fondo muy grande comparado con el objeto u objetos en la imagen puede también dar lugar a valores de umbral que produzcan resultados indeseados.

El valor óptimo u^* puede encontrarse al buscar en el rango [0, L-1] el valor de *u* que maximice la ecuación. Esto es:

$$u^* = \arg \max_{0 \le u \le L-1} \{J_1(r)\}$$

En su trabajo, *Otsu* demostró que los siguientes criterios son equivalentes al dado por la ecuación:

$$J_2(u) = \frac{\sigma_1}{P_1(u)\sigma_1^2 + P_2(u)\sigma_2^2}$$

$$J_3(u) = \frac{P_1(u)P_2(u)[\mu_1(u) - \mu_2(u)]^2}{\sigma_2}$$

Donde, en este caso:

$$\sigma^2 = \sum_{r=0}^{L-1} (r - \mu)^2 \cdot p(r)$$

y:

$$\mu = \sum_{r=0}^{L-1} r \cdot p(r) = P_1(r)\mu_1(r) + P_2(r)\mu_2(r)$$

La ventaja principal del método de *Otsu* es que no hace ninguna suposición acerca de las densidades $P_1(u)$ y $P_2(u)$, pues asume que pueden ser descritas sólo en términos de sus medias y varianzas lo que no necesariamente es cierto en el caso general.

Una de las principales desventajas de este método es la suposición de que el histograma de la imagen es bimodal, esto es, que los píxeles de la imagen pueden ser clasificados en sólo dos clases. Para más de dos clases de píxeles en la imagen, el

método debe ser modificado de manera que varios umbrales puedan ser definidos de tal forma que permitan maximizar la varianza dentro de la clase y minimizar la varianza entre clases.

2.1.13. Etiquetado de regiones o componentes conectadas

Una de las operaciones más comunes en visión artificial es la de encontrar las componentes conectadas dentro de una imagen binaria. Los puntos en una componente conectada forman candidatos para la representación de un objeto, por lo que el etiquetado de componentes conexas permite encontrar el número de objetos que hay en una imagen [5]. Las etiquetas puestas a los objetos pueden ser mostradas en las imágenes a través de colores, niveles de gris, números, etc.

Algoritmo iterativo

Un algoritmo como éste no usa almacenamiento auxiliar para producir una imagen etiquetada a partir de una imagen binaria. Consta de tres pasos básicos, uno de etiquetado inicial, uno de propagación de arriba hacia debajo de las etiquetas y, finalmente, uno de propagado de etiquetas de abajo hacia arriba. [7] En más detalle, dada una imagen binaria $b(x, y)$, los tres pasos son los siguientes:

1. Barrer $b(x,y)$ hasta encontrar un píxel de tipo objeto (con valor *1* ó *L-1*) aún no etiquetado y asignarle una nueva etiqueta *E*. Esto da como resultado la imagen $e_1(x, y)$.

2. Barrer $e_1(x,y)$ de arriba hacia abajo hasta encontrar un píxel etiquetado y propagar su etiqueta a sus vecinos, según la métrica elegida. Esto da como resultado la imagen $e_2(x,y)$.

3. Barrer $e_2(x, y)$ de abajo hacia arriba hasta encontrar un píxel etiquetado y propagar su etiqueta a sus vecinos, según la métrica elegida. Esto da como resultado la imagen $e_f(x,y)$.

2.1.14. Morfología matemática

Las bases teóricas de la morfología matemática se deben al científico alemán nacido en Rusia, *Hermann Minkowski* (1864-1909). Entre los muchos y variados logros a lo largo de su fructífera vida profesional, trabajó en el tema de las figuras convexas y las relaciones entre sus formas, siendo precisamente en este campo donde creó las bases matemáticas fundamentales para la morfología matemática. *Minkowski* fue el primer ser

humano al que se le ocurrió sumar formas. Siendo *A* y *B* dos conjuntos cualesquiera, sobre cuyos elementos esté bien definida la operación binaria suma (+), el conjunto de la suma de las dos formas *A* y *B* contiene todos los elementos que resultan de sumar cada uno de los elementos del conjunto *A* con todos y cada uno de los elementos del conjunto *B*.

$$A + B = \{x = a + b, con\ a\ \in A\ y\ b \in B\}$$

No obstante, la resta de *Minkowski* no se debe a él, sino a un investigador alemán llamado *Hadwiger*, quien la propuso medio siglo después. La resta de *Minkowski* se define como la operación dual de la suma de *Minkowski* y apareció por primera vez en el año de 1957.

$$A - B = \{todas\ las\ x\ elementos\ de\ A, pero\ no\ de\ B\}$$

Basado en las ideas fundamentales de *Minkowski* y en los trabajos de *Hadwiger*, el francés *George Matheron* inició la morfología matemática a mediados de los 60 con sus trabajos de investigación en análisis de imágenes en el ámbito de los medios porosos. En 1982, su alumno *Jean Serra* dio un impulso más a la morfología matemática. *Matheron* y *Serra* bautizaron a las operaciones duales de *Minkowski* con los nombres actuales de las dos operaciones básicas de la morfología matemática: dilatación y erosión.

En términos simples, la morfología matemática procesa formas (que pueden ser imágenes digitales), con ayuda de formas especiales escogidas previamente, las cuales son generalmente pequeñas a las que se les ha llamado elementos estructurantes. Es decir, una forma dada se dilata o se erosiona tomando como base un elemento estructurante escogido previamente. [19] [20]

Elemento estructurante

El elemento estructurante actúa como un operador sobre una imagen para producir un resultado.[8] La forma, tamaño y orientación del elemento estructurante son escogidas de acuerdo a un conocimiento previo acerca de las estructuras geométricas relevantes presentes en la imagen y al objetivo que se persigue con la operación morfológica.

En los elementos estructurantes planos uni o bidimensionales todos los puntos del conjunto tienen el mismo valor, mientras que en los no planos tienen asociados a cada punto un valor o peso que los hacen similares a pequeñas imágenes en tonos de gris.

Cada elemento estructurante requiere la definición de un punto de origen (o de referencia) para su aplicación como operador morfológico. Esto permite que el elemento estructurante se pueda relacionar de una forma particular con los píxeles de la imagen.

Morfología binaria

Una imagen binaria Euclidiana es un subconjunto del espacio euclidiano de dimensión 2. Para la implementación digital se considera una imagen bidimensional como un conjunto de píxeles dispuestos en dos dimensiones, es decir, como una matriz bidimensional. [3]

El lenguaje de la morfología binaria es el de la teoría de conjuntos. En ésta se definen tres operaciones básicas: Sean *A* y *B* dos conjuntos en un espacio *n*-dimensional E^n con elementos $a = \{a_1, ..., a_n\}$ y $b = \{b_1, ..., b_n\}$, respectivamente, es decir, ambas son *n*-tuplas.

1. La traslación de *A* por $x \epsilon E^n$ que se denota A_x se define como aquellos elementos *c* tales que su nueva posición está dada por $a + x$ para toda $a \epsilon A$.

$$A_x = \{c \mid c = a + x, para\ toda\ a \in A\}$$

2. La diferencia de *A* y *B*, denotada por *A* - *B*, se define como aquellos puntos x que pertenecen a *A*, pero no pertenecen a *B*.

$$A - B = \{x \mid x \in a, x \notin B\}$$

3. El complemento de *A* se define como aquellos puntos *x* en el espacio que no pertenecen a *A*.

$$A^c = \{x \mid x \notin A\}$$

Erosión binaria

La erosión de un conjunto *A* por un conjunto *B* se denota por $A \Theta B$ y se define como:

$$A \Theta B = \{x \in E^n \mid x + b \in A\ para\ todo\ b \in B\}$$

Es el conjunto de todos los elementos *x* para los que $(x + b) \epsilon A$ para todo $b \epsilon B$. También se puede expresar la erosión de una imagen *A* por un elemento estructurante *B* como el conjunto de todos los elementos $x \epsilon E^n$ para los cuales *B* trasladado por su referencia *x* está contenido en *A*:

$$A \Theta B = \{x \in E^n \mid B_x \leq A\}$$

Las siguientes figuras muestran un ejemplo de la erosión binaria:

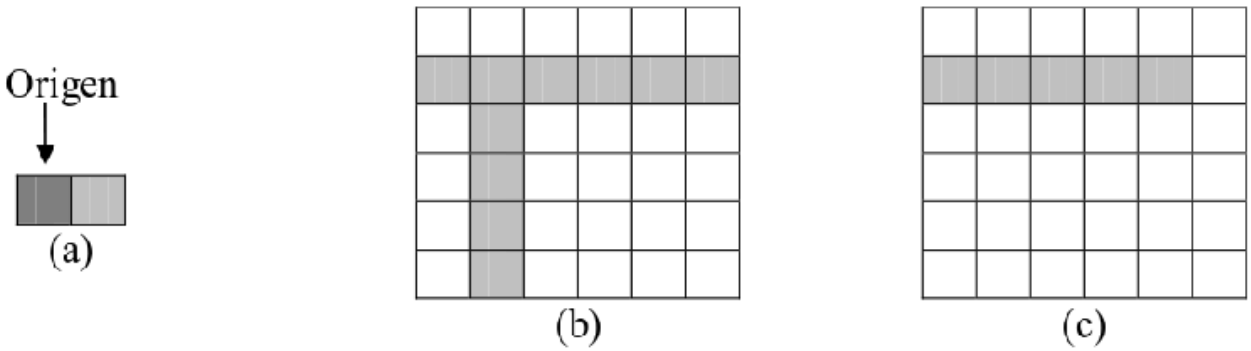


Figura 5: a) Elemento estructurante; b) Imagen original; c) Imagen erosionada

Dilatación binaria

La dilatación de un conjunto *A* por un conjunto *B* es denotada por $A \oplus B$ y se define como:

$$A \oplus B = \{c = E^n \mid c = a + b \; para \; todo \; a \in A \; y \; b \in B\}$$

Es el conjunto de todos los posibles elementos *c* que son suma de pares de elementos, uno de *A* y otro de *B*, para todo elemento $a \epsilon A$ y $b \epsilon B$. La dilatación es el dual de la erosión.

$$A \oplus B = [A^c \Theta (-B)]^c$$

De ahí que el elemento estructurante aparezca reflejado (*-B*). La figura muestra un ejemplo de dilatación binaria.

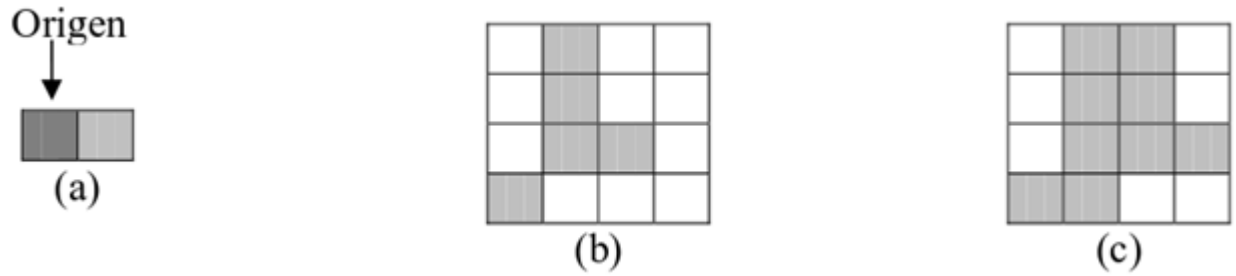


Figura 6: a) Elemento estructurante; b) Imagen original; c) Imagen dilatada

Propiedades de la erosión y la dilatación binarias

La existencia de un sistema de relaciones algebraicas como una característica de la morfología matemática de la que forman parte la erosión, la dilatación y las

operaciones básicas de la teoría de conjuntos, constituyen el álgebra de Minkowski. [9]

Entre de las propiedades más importantes del álgebra de Minkowski se encuentran las siguientes:

1. Conmutatividad	$A \oplus B = B \oplus A$
2. Asociatividad	$A \oplus (B \oplus C) = (A \oplus B) \oplus C$
3. Distributividad	$A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$
	$A \ominus (B \cup C) = (A \ominus B) \cup (A \ominus C)$
4. Monotonía creciente	$A_1 \leq A_2 \rightarrow A_1 \oplus B \leq A_2 \oplus B$
	$A_1 \leq A_2 \rightarrow A_1 \ominus B \leq A_2 \ominus B$
5. Dualidad	$A \oplus B = [A^c \ominus (-B)]^c$

Apertura y cierre

Las operaciones de apertura y cierre, útiles en el filtrado morfológico de imágenes, se crean a partir de la dilatación y la erosión.

La apertura de una imagen *A* por un elemento estructurante *B* se denota *A°B* y se define como:

$$A \circ B = (A \ominus B) \oplus B$$

La expresión anterior no es más que la erosión de una imagen *A* por un elemento estructurante *B* seguido de una dilatación con el mismo elemento estructurante B.

El cierre de una imagen *A* por un elemento estructurante *B* se denota *A•B* y se define como:

$$A \bullet B = (A \oplus B) \ominus B$$

La expresión anterior representa una dilatación de una imagen *A* por un elemento estructurante *B* seguida de una erosión con el mismo elemento estructurante *B*.

La apertura y el cierre tienden a suavizar los contornos de las imágenes. La apertura tiende a quitar las protuberancias de las imágenes, así como a desconectar conjuntos y a suprimir las componentes conexas más pequeñas que son cubiertas completamente por el elemento estructurante. El cierre, por su parte, tiende a conectar componentes cercanas de la imagen y a llenar los huecos, siempre que sean más pequeños que el elemento estructurante usado. En cualquier caso, la elección adecuada del elemento estructurante es de gran importancia si se desea eliminar ruido de la imagen y a la vez conservar las partes de interés. Véase la siguiente figura.

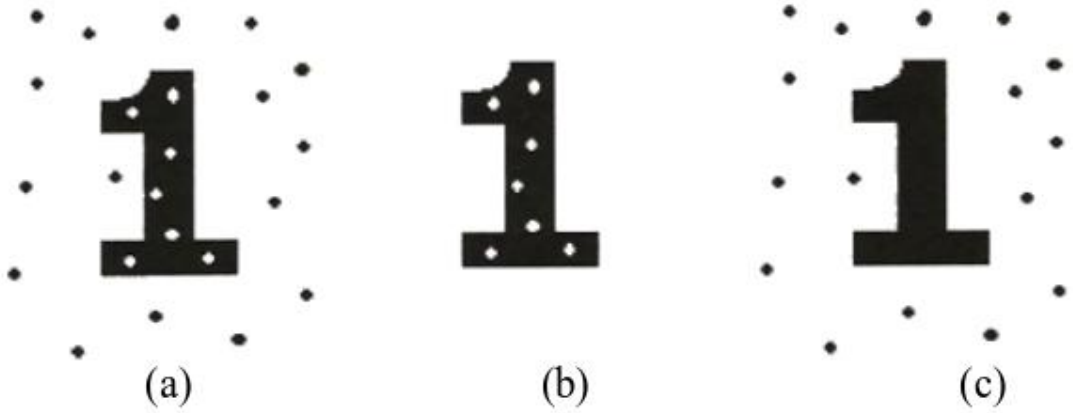


Figura 7: a) Imagen original; b) Imagen tras apertura; c) Imagen tras cierre

Propiedades de la apertura y el cierre

Entre las propiedades de la apertura y el cierre se encuentran las siguientes: [9]

1. Idempotencia

$$A \circ B = (A \circ B) \circ B$$
$$A \bullet B = (A \bullet B) \bullet B$$

2. Monotonía creciente

$$A_1 \leq A_2 \rightarrow A_1 \circ B \leq A_2 \circ B$$
$$A_1 \leq A_2 \rightarrow A_1 \bullet B \leq A_2 \bullet B$$

3. Dualidad

$$(A \bullet B)^c = A^c \circ B$$
$$(A \circ B)^c = A^c \bullet B$$

Morfología en niveles de gris

Las operaciones de la morfología binaria son extendidas al tratamiento de imágenes en tonos de gris. En este caso al elemento estructurante se le llama función estructurante. Trabajando en Z^2 discreto, donde *f(x, y)* denota la imagen y *b(x, y)* el elemento estructurante.

Erosión en niveles de gris

La erosión de imágenes en niveles de gris, denotada por $f \ominus b$, se define mediante la siguiente ecuación:

$$(f \ominus b)(s,t) = \min\{f(s+x, y+t) - b(x,y) \mid (s+x, y+t) \in D_f(x,y) \in D_b\}$$

donde D_f y D_b son los dominios de *f* y *b* respectivamente.

La siguiente figura muestra un ejemplo de la erosión en niveles de gris.

Figura 8: a) Imagen original en nivel de gris; b) Imagen erosionada

La imagen que aparece en la anterior fue obtenida al trasladar el elemento estructurante por toda la imagen y tomar en cada posición el mínimo valor de todos los obtenidos en la zona de análisis de la ventana en el dominio de la función estructurante y reducirle a los elementos correspondientes de la imagen cada nivel en dicho dominio. Como se observa, el resultado da una imagen más oscura que la original.

Dilatación en niveles de gris

La dilatación de una imagen en niveles de gris *f* por *b* se denotará $f \oplus b$ y se define como:

$$(f \oplus b)(s,t) = \text{máx}\{f(s-x, y-t) - b(x,y) \mid (s-x, y-t) \in D_f(x,y) \in D_b\}$$

donde D_f y D_b son los dominios de *f* y *b* respectivamente. La siguiente figura muestra un ejemplo de la dilatación en niveles de gris.

Figura 9: a) Imagen original en nivel de gris; b) Imagen dilatada

La imagen que parece en la figura fue obtenida al trasladar el elemento estructurante por toda la imagen y tomar en cada posición el máximo valor de todos los obtenidos en la zona de análisis de la ventana en el dominio de la función estructurante y añadirle a los elementos correspondientes de la imagen cada nivel en dicho dominio. Como se observa, el resultado da una imagen más clara que la original.

Apertura y cierre en niveles de gris

Las operaciones de apertura y cierre son también extendidas a niveles de gris y definidas de igual forma:

La apertura se define como:

$$f \circ B = (f \ominus B) \oplus B$$

El cierre se define como:

$$f \bullet B = (f \oplus B) \ominus B$$

En la siguiente figura se puede observar un ejemplo de ambos operadores por un elemento estructurante plano circular de 5 píxeles de diámetro. Aquí se puede ver cómo se suaviza la imagen cuando la apertura suprime los picos (claros) mientras que el cierre rellena los valles (oscuros). [3]

Figura 10: a) Imagen original en nivel de gris; b) Imagen tras apertura; c) Imagen tras cierre

Detectores de cimas y valles

Al sustraer la apertura de una imagen de la original, quedan marcadores, los cuales son zonas donde existen picos de intensidad. En la morfología de escala de grises, esto puede ser útil para detectar pequeños grupos de píxeles más oscuros (o menos claros) en entornos claros, o pequeños grupos de píxeles más claros (o menos oscuros) en entornos oscuros. También se puede usar para detectar bordes en imágenes que presenten poco ruido. [7]

El operador conocido como Detector de Cimas se denomina Sombrero de Copa ("TOPHAT") y se define de la siguiente forma:

$$TopHat(f) = f - (f \circ g)$$

Donde *f* es la imagen y *g* es el elemento estructurante, ambos en tonos de gris. Como la

apertura es antiextensiva, la imagen de la apertura quedará siempre por debajo de la imagen original, por lo que la operación TOPHAT nunca será negativa.

El dual del operador TOPHAT es el Detector de Valles (“BOTTOMHAT”) y se define como:

$$BottomHat(f) = (f \bullet g) - f$$

La imagen resultante nunca será negativa, ya que el cierre es una operación extensiva y siempre quedará por encima de la imagen original. En la siguiente figura se ilustra cómo actúan ambos operadores.

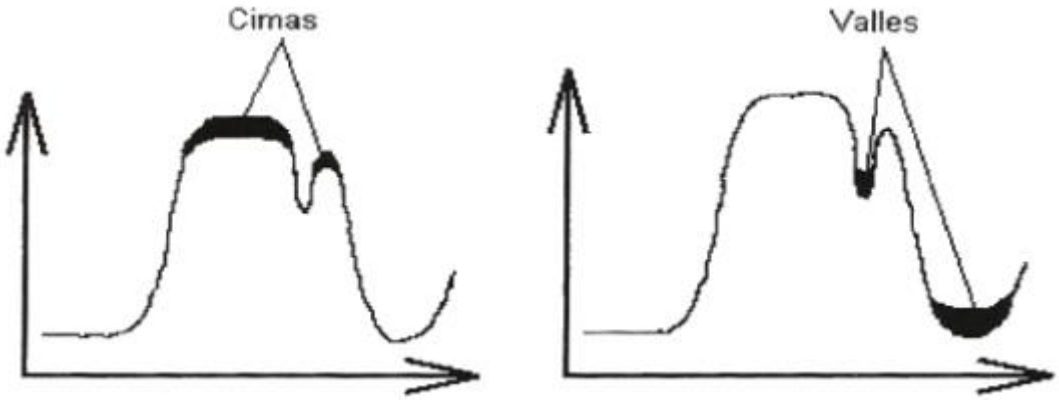


Figura 11: a) Detector de cimas "Top-Hat"; b) Detector de valles "Bottom-Hat"

Existen casos en los que resulta necesario condicionar o restringir la aplicación de ciertos algoritmos de morfología a la conectividad existente entre los píxeles de una imagen, sobre todo cuando se realizan dilataciones y erosiones. Dicha restricción es generalmente aportada por una imagen máscara y da lugar a lo que se conoce como morfología geodésica.

2.1.15. Transformaciones Geodésicas

Transformaciones geodésicas

Las transformaciones geodésicas consideran dos imágenes de entrada. Se le aplica una transformación morfológica a la primera imagen y después es forzada a mantenerse acotada por la segunda imagen.

Dilatación geodésica

Una dilatación geodésica [9] envuelve dos imágenes: una imagen marcador y una imagen máscara, ambas en el mismo dominio de definición. La imagen máscara debe ser de mayor o igual tamaño que la imagen marcador (desde el punto de vista de sus niveles de gris).

La imagen marcador es primero dilatada con un elemento isotrópico elemental y después la imagen dilatada es forzada a mantenerse por debajo de la imagen máscara. Por lo tanto, la imagen máscara actúa como un límite a la propagación de la dilatación de la imagen marcador. Denotamos a *f* la imagen marcador y *g* a la imagen máscara.

$$D_f = D_g \; y \; f \leq g$$

La dilatación geodésica de tamaño uno de la imagen marcador *f* con respecto a la imagen máscara *g* se denota por:

$$\partial_g^{(1)} = \partial^{(1)}(f) \wedge g$$

Las siguientes figuras muestran la dilatación geodésica para una imagen binaria bidimensional y para una señal de una dimensión.

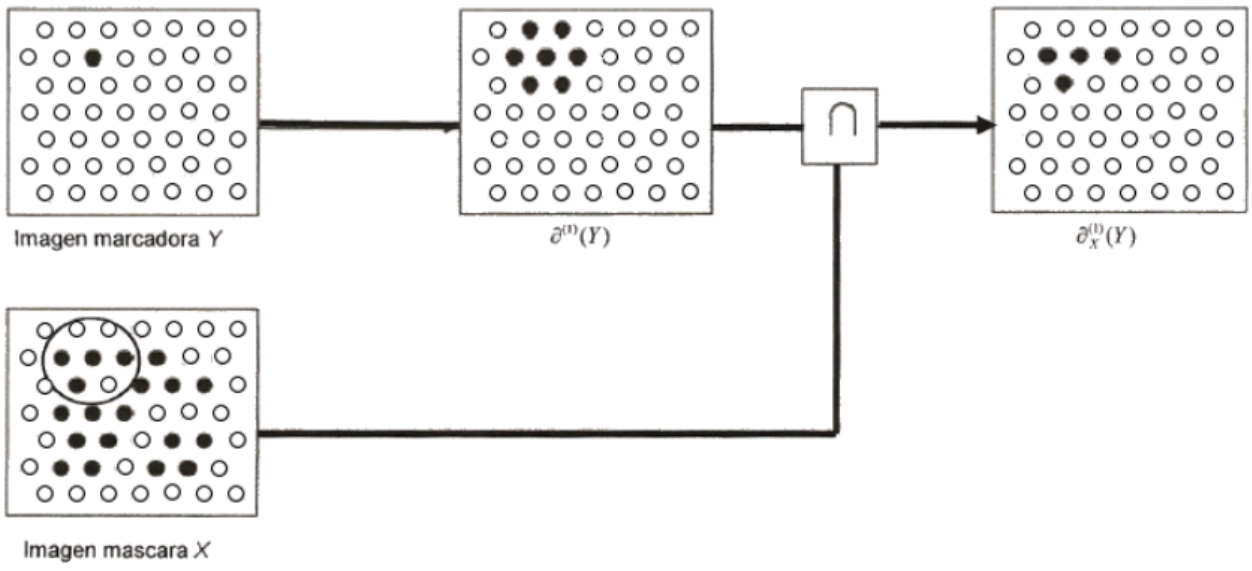


Figura 12: Dilatación geodésica sobre imagen bidimensional

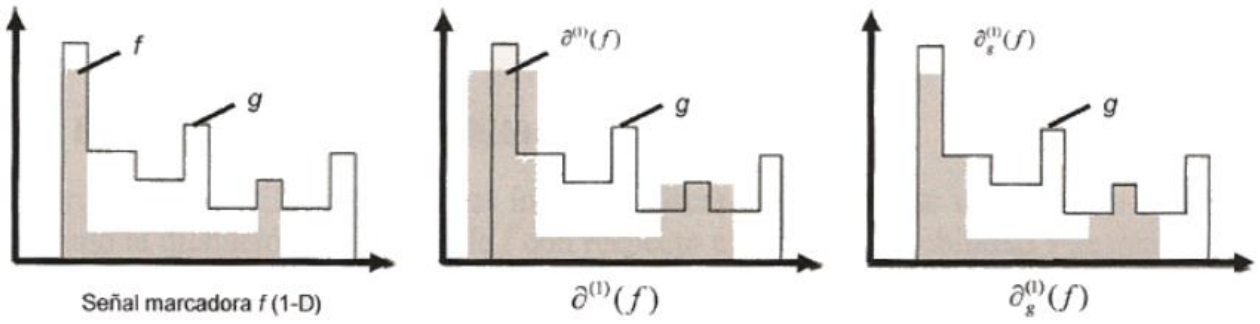


Figura 13: Dilatación geodésica sobre señal unidimensional

Erosión geodésica

La erosión geodésica es la transformación dual de la dilatación geodésica con respecto a la complementación:

$$\varepsilon_g^{(1)}(f) = [\partial^{(1)}(f^c) \wedge g^c]^c$$

De aquí, la imagen marcador es primero erosionada geodésicamente y después se calcula píxel a píxel el máximo con respecto a la imagen máscara.

Dada la dualidad entre las erosiones y las dilataciones geodésicas, las erosiones geodésicas son crecientes y antiextensivas. La imagen marcador es primero erosionada y después se calcula el máximo de tipo puntual con la imagen máscara. La figura muestra la erosión geodésica de una señal.

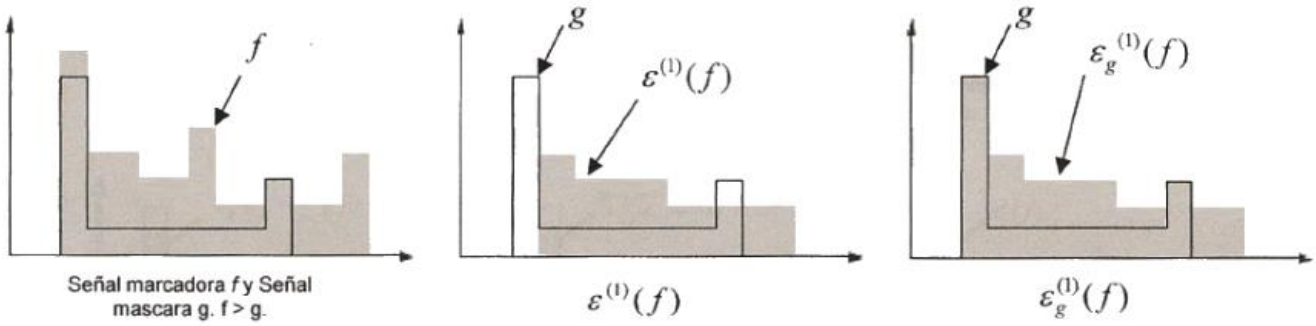


Figura 14: Erosión geodésica sobre señal unidimensional

Reconstrucción morfológica de imágenes

Las erosiones y dilataciones geodésicas de un tamaño dado son raras veces usadas en la práctica. Sin embargo, cuando son iteradas hasta la estabilidad, permiten la definición de potentes algoritmos de reconstrucción morfológica.

Las dilataciones y erosiones de imágenes limitadas siempre convergen después de un número finito de iteraciones, es decir, hasta que la propagación o acotamiento de la imagen marcador es impedido totalmente por la imagen máscara.

Reconstrucción geodésica por dilatación

La reconstrucción por dilatación [9] de una imagen máscara *g* a partir de una imagen marcador *f* es definida como la dilatación geodésica de *f* con respecto a *g* hasta la estabilidad y se denota por:

$$R_g^{\partial} = \partial_g^{(i)}(f)$$

donde *i* es tal que:

$$\partial_g^{(i)}(f) = \partial_g^{(i+1)}(f)$$

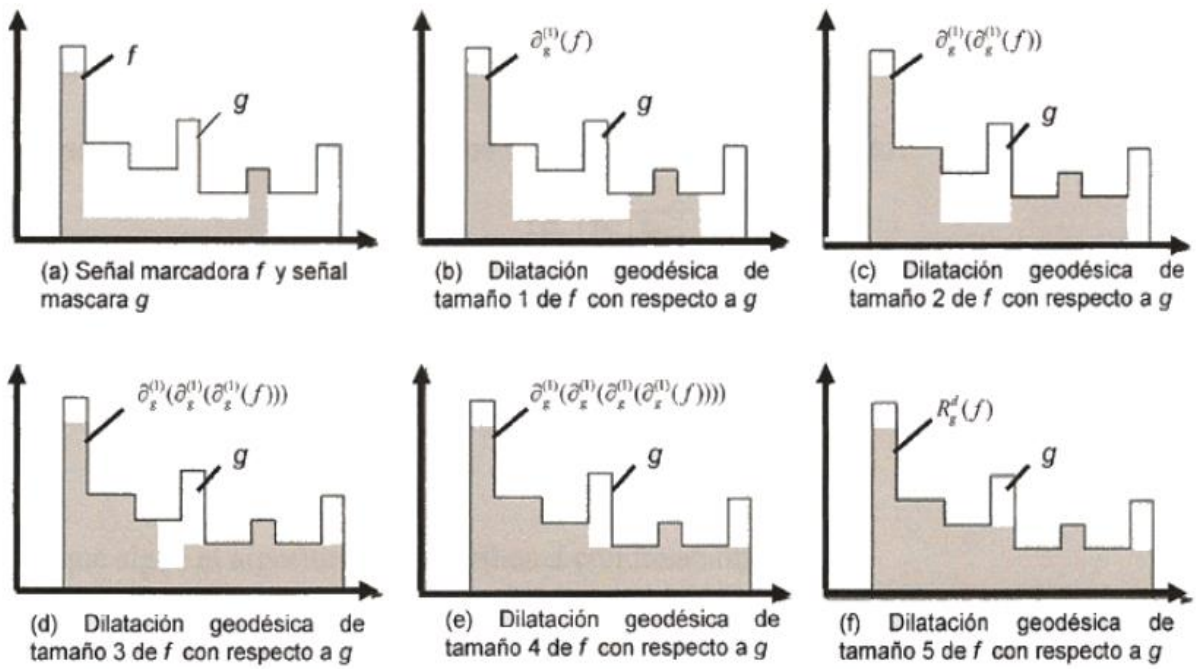


Figura 15: Proceso de reconstrucción geodésica

La reconstrucción por dilatación en señales 1-D de tonos de gris es ilustrada a continuación, donde la estabilidad se alcanza después de la quinta dilatación geodésica.

Reconstrucción geodésica por erosión

La erosión geodésica [9] es la transformación dual de la dilatación geodésica:

$$\varepsilon_g^{(1)}(f) = [\partial^{(1)}(f^c) \wedge g^c]^c$$
$$\varepsilon_g^{(1)}(f) = [\partial^{(1)}(f^c) \wedge g^c]^c = \varepsilon^{(1)}(f) \vee g$$

dónde: $f \geq g$, y $\varepsilon^{(1)}$ denota la erosión elemental.

Dada la dualidad entre las dilataciones y las erosiones geodésicas, estas últimas son crecientes y antiextensivas. La imagen marcador es primero erosionada por un elemento estructurante isotrópico elemental y después se calcula el máximo de tipo puntual con la imagen máscara. Por lo tanto, la imagen máscara actúa como un límite en el acotamiento de la imagen marcador. La erosión geodésica de tamaño *n* de una imagen marcador *f* con respecto a una imagen máscara *g*, se obtiene ejecutando *n* erosiones geodésicas sucesivas de *f* con respecto a *g*.

2.1.16. Reconocimiento óptico de caracteres (OCR)

Una vez se ha logrado segmentar la imagen quedando como resultado, únicamente, los objetos de interés dentro de la misma (en nuestro caso, los dígitos que forman la matrícula) se procede a llevar a cabo el análisis de los mismos, es decir, su clasificación y posterior interpretación. Para tal fin, se utilizan los sistemas de reconocimiento de caracteres (OCR, por sus siglas en ingles).

La tecnología OCR (**Optical Character Recognition**) engloba a un conjunto de técnicas que complementándose entre sí, se emplean para distinguir de forma automática entre los diferentes caracteres alfanuméricos existentes. En realidad no se reconocen exactamente los caracteres de un determinado alfabeto, sino que es posible distinguir entre cualquier conjunto de ideogramas. Sin embargo, se debe tener en cuenta que la precisión que se obtiene en la práctica al intentar distinguir entre un conjunto de símbolos no es del 100%. Por lo tanto, es fácil deducir que cuanto más numeroso es el conjunto de símbolos entre los que se debe decidir, mayor es la probabilidad de que se produzca un fallo de clasificación.

En todo sistema de reconocimiento óptico de caracteres (OCR) se distinguen al menos estas 4 etapas:

1. Adecuación de la imagen (preproceso).
2. Selección de la zona de interés (segmentación).
3. Representación digital de la imagen (extracción de características).
4. Distinción del carácter contenido en la imagen (reconocimiento).

Y para cada una de las cuatro etapas es posible aplicar multitud de técnicas ya existentes o desarrollar alguna específica en función de las condiciones en las que se presentan las imágenes de entrada. [10]

2.2. Librería OpenCV

OpenCV (**Open source Computer Vision library**) es una librería abierta de visión por computador desarrollada por Intel. La primera aparición se remonta a 1999, donde se presentó la versión alfa de la misma. Esta librería proporciona una gran colección de funciones para el procesado de imágenes y permiten a los programadores crear aplicaciones poderosas en el dominio de la visión digital. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

OpenCV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos, calibración de cámaras, visión estéreo y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado, realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multinúcleo. OpenCV puede además utilizar el sistema de primitivas de rendimiento integradas de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel.

La librería implementa una gran variedad de herramientas para la interpretación de la imagen. Es compatible con "*Intel Image Processing Library*" (IPL) que implementa algunas operaciones en imágenes digitales. A pesar de primitivas como binarización, filtrado, estadísticas de la imagen, pirámides, etc. OpenCV es principalmente una librería que implementa algoritmos para las técnicas de la calibración (calibración de la cámara), detección de rasgos, para estimación de movimiento (flujo óptico), análisis de la forma (geometría, contorno que procesa), análisis del movimiento (plantillas del movimiento, estimadores), reconstrucción 3D (transformación de vistas), segmentación de objetos y reconocimiento (histograma, etc.).

El rasgo esencial de la librería junto con su funcionalidad y la calidad, es su desempeño. Los algoritmos están basados en estructuras de datos muy flexibles,

acoplados con estructuras IPL; más de la mitad de las funciones ha sido optimizadas aprovechándose de la arquitectura de Intel.

OpenCV usa la estructura Iplimage para crear y manejar imágenes. Esta estructura tiene gran cantidad de campos. Provee, además, una interfaz gráfica llamada highGUI. Esta interfaz gráfica es muy importante porque se necesita bajo OpenCV para visualizar imágenes.

Hay varios tipos de datos fundamentales en OpenCV, y varios tipos de datos auxiliares para hacerla más simple y uniforme.

Los tipos de los datos fundamentales incluyen tipos de vectores como: **IplImage** (imagen de IPL), **CvMat** (matriz), colecciones con capacidad de crecimiento: **CvSeq** (secuencia), **CvSet**, **CvGraph** y tipos mixtos: **CvHistogram** (histograma multi-dimensional).

Los tipos de datos auxiliares incluyen: **CvPoint** (2d punto), **CvSize** (anchura y altura), **CvTermCriteria** (criterio de la terminación para los procesos iterativos), **IplConvKernel** (núcleo de la circunvolución), **CvMoments** (momentos espaciales), etc.

Además, cabe resaltar que el software de OpenCV utiliza las siguientes convenciones:

- Los identificadores constantes están en mayúsculas.
- Todos los nombres de las funciones usadas para el procesado de imagen tienen el prefijo “cv”.
- Todas las funciones externas de OpenCV empiezan con el prefijo “cv”, y todas las estructuras con prefijo “Cv”.
- Cada nueva parte de una función empieza con un carácter en mayúscula.
- Los nombres de funciones en OpenCV tienen el siguiente formato: “*cv [action] [target] [mod] ()”*; donde:
    - ***action***: la acción indica la funcionalidad del centro.
    - ***target***: indica el área donde el procesado de la imagen está siendo establecido. En la mayoría de los casos, target consiste en dos o más palabras.
    - ***mod***: es un campo opcional; indica una modificación de la funcionalidad de la función.
- Algunos nombres de funciones consisten en una acción u objetivo solamente.

En el *Anexo I*, se presentan las funciones de OpenCV más interesantes para el desarrollo del proyecto y, además, puede resultar ilustrativo de las convenciones a las que se hace referencia en este punto.

Quizá de los pocos inconvenientes que se pueden encontrar en ella sea en el caso del reconocimiento de dígitos, en el cual, el principal inconveniente es que no ofrece un producto completo (no incluye un OCR propio), tan sólo algunas piezas que sirven como base para montar sobre ellas un producto final.

Otro de los inconvenientes que tiene es la necesidad de utilizar la librería IPL para tener acceso a funciones de bajo nivel.

Sin embargo, la presencia de funciones muy interesantes, y las posibilidades ya comentadas que ofrece la librería hacen que estos inconvenientes no sean realmente significantes. [11]

2.3. Tesseract-OCR

Tesseract es un motor de reconocimiento óptico de caracteres (*OCR*) desarrollado en los laboratorios de Hewlett-Packard en Greeley (Colorado), como software propietario, entre 1989 y 1994. En 1996 se realizaron las modificaciones necesarias para su portabilidad en Windows y, más tarde, en 1998 se migró el sistema de C a C++.

Tras casi 10 años sin ningún desarrollo, fue liberado como código abierto en el año 2005 por **Hewlett-Packard** y la **Universidad de Las Vegas** (Nevada).

Actualmente es desarrollado por Google y distribuido bajo licencia Apache, versión 2.0, habiendo llegado a ser considerado uno de los OCR libres con mayor precisión disponibles actualmente. [12]

2.3.1. Funcionamiento de Tesseract-OCR

En primer lugar se realiza un análisis de las componentes conectadas y se almacenan los contornos pertenecientes a los objetos de la imagen binaria aportada a la entrada. Esto puede ser computacionalmente costoso pero supone una gran ventaja, ya que aporta información sobre los contornos que forman cada carácter. En este paso, se agrupan los objetos de contornos similares en amasijos (“blobs”).

Dichos amasijos se organizan en líneas de texto y éstas son diferenciadas según posean un ancho de carácter fijo o variable. La siguiente imagen puede resultar orientativa a la hora de entender cuando hablamos de ancho de carácter fijo o variable.

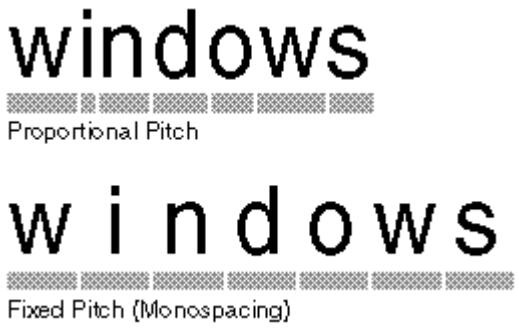


Figura 16: Comparativa entre ancho proporcional y fijo

Las líneas de textos se dividen entonces en palabras de acuerdo con el espaciado entre los caracteres y, si se trata de palabras con caracteres de ancho fijo, se introduce cada uno de ellos en una celda de carácter. En el caso de palabras con caracteres de ancho variable se divide únicamente en palabras según valores de espaciados predefinidos.

En ese momento, el reconocimiento entra en un proceso de dos fases. En la primera de ellas, se hace un intento por reconocer, mediante un clasificador, cada una de las palabras (o carácter) que conforman el texto. Las que se reconocen positivamente se pasan al un clasificador adaptativo como patrón de entrenamiento, consiguiendo mayor capacidad de acierto según avancemos en el análisis del texto.

Esto puede provocar que se hayan errado intentos en las primeras palabras (o caracteres) del texto que se pretende reconocer, lo que lleva a la segunda fase. Esta segunda fase realiza otra pasada sobre el texto, con todo lo aprendido por el clasificador durante la primera iteración y vuelve a decidir sobre las palabras (o caracteres) con menor porcentaje de fiabilidad.

Por último, se ejecutan algoritmos para resolver la distinción entre caracteres en mayúscula o minúscula. [12]

Cabe destacar que sea cual sea el método de reconocimiento que se lleve a cabo, es muy aconsejable un analizador sintáctico de los datos obtenidos, con el fin de corregir fallos parciales del clasificador, mejorar la robustez del sistema y la capacidad de acierto del mismo. Esto se debe a que los OCR actuales disponen de una colección de clases muy superior a los caracteres válidos para el sistema a implementar.

En el *Anexo I*, se presentan las funciones de Tesseract-OCR más interesantes para el desarrollo del proyecto y, además, puede resultar ilustrativo del proceso al que se hace referencia en este punto.

2.4. MySQL

MySQL (cuyas siglas en inglés se trasladan a **My Structured Query Language** o Lenguaje de Consulta Estructurado) es un sistema gestor de bases de datos (SGBD) multiplataforma muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Aunque carece de algunas características avanzadas disponibles en otros SGBD del mercado, es una opción atractiva tanto para aplicaciones comerciales como no comerciales precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo licencia GPL le otorgan como beneficios adicionales (no menos importantes) contar con un alto grado de estabilidad y un rápido desarrollo.

MySQL es un SGBD que ha ganado popularidad por una serie de atractivas características:

- Está desarrollado en C/C++.
- Se distribuyen ejecutables para cerca de diecinueve plataformas diferentes.
- La API se encuentra disponible en C, C++, Eiffel, Java, Perl, PHP, Python, Ruby y TCL.
- Está optimizado para equipos de múltiples procesadores.
- Es muy destacable su velocidad de respuesta.
- Se puede utilizar como cliente-servidor o incrustado en aplicaciones.
- Cuenta con un rico conjunto de tipos de datos.
- Soporta múltiples métodos de almacenamiento de las tablas, con prestaciones y rendimiento diferentes para poder optimizar el SGBD a cada caso concreto.
- Su administración se basa en usuarios y privilegios.
- Se tiene constancia de casos en los que maneja cincuenta millones de registros, sesenta mil tablas y cinco millones de columnas.
- Sus opciones de conectividad abarcan TCP/IP, sockets UNIX y sockets NT, además de soportar completamente ODBC.
- Los mensajes de error pueden estar en español y hacer ordenaciones correctas con palabras acentuadas o con la letra 'ñ'.
- Es altamente confiable en cuanto a estabilidad se refiere.

Al comprender sus principios de diseño, se puede explicar mejor las razones de algunas de sus carencias. Por ejemplo, el soporte de transacciones o la integridad referencial (la gestión de claves foráneas) en MySQL está condicionado a un esquema de almacenamiento de tabla concreto, de forma que si el usuario no va a usar transacciones, puede usar el esquema de almacenamiento "tradicional" (MyISAM) y obtendrá mayor rendimiento, mientras que si su aplicación requiere transacciones, deberá usar el esquema que lo permite (InnoDB), sin ninguna otra restricción o implicación.[13]

2.5. PHP

PHP, acrónimo de "***PHP: Hypertext Preprocessor***", es un lenguaje de 'scripting' de propósito general, multiplataforma y de código abierto que está especialmente pensado para el desarrollo web y que puede ser embebido en páginas HTML. Su sintaxis es similar a C, Java y Perl, siendo así sencillo de aprender. La meta principal de este lenguaje es permitir el desarrollo rápido de páginas web dinámicas; aunque se puede hacer mucho más.

PHP tal y como se conoce hoy en día es en realidad el sucesor de un producto llamado PHP/FI. Creado en 1994 por *Rasmus Lerdorf*, la primera encarnación de PHP era un conjunto simple de ficheros binarios *Common Gateway Interface* (CGI) escritos en el lenguaje de programación C. Originalmente utilizado para rastrear visitas de su currículum online, llamó al conjunto de scripts "*Personal Home Page Tools*", más frecuentemente referenciado como "*PHP Tools*". Con el paso del tiempo se quiso más funcionalidad, y Rasmus reescribió *PHP Tools*, produciendo una implementación más grande y rica. Este nuevo modelo fue capaz de interaccionar con bases de datos, y mucho más, proporcionando un entorno de trabajo sobre el cual los usuarios podían desarrollar sencillas aplicaciones web dinámicas tales como libros de visitas.

Lo que distingue a PHP de lenguajes del lado del cliente como Javascript es que el código es ejecutado en el servidor, generando HTML y enviándolo al cliente. El cliente recibirá el resultado de ejecutar el script, aunque no se sabrá el código subyacente. El servidor web puede ser configurado incluso para que procese todos los ficheros HTML con PHP, por lo que no hay manera de que los usuarios puedan saber qué se tiene debajo de la manga.

Lo mejor de utilizar PHP es su extrema simplicidad para el principiante, pero a su vez ofrece muchas características avanzadas para los programadores profesionales.

Una de las características más potentes y destacables de PHP es su soporte para un amplio abanico de bases de datos. Escribir una página web con acceso a una base de datos es increíblemente simple utilizando una de las extensiones específicas de bases de datos (p.ej., para MySQL), o conectarse a cualquier base de datos que admita el estándar de Conexión Abierta a Bases de Datos por medio de la extensión ODBC. Otras bases de datos podrían utilizar cURL o sockets, como lo hace CouchDB. [14]

En el *Anexo II*, se presentan las funciones PHP más interesantes para el desarrollo del proyecto y, además, puede resultar ilustrativo de las características a las que se hace referencia en este punto.

2.6. cURL

PHP (a partir de la versión 4.0.2) soporta libcurl, una librería creada por *Daniel Stenberg* que permite conectarse y comunicarse con diferentes tipos de servidores y

diferentes tipos de protocolos. Actualmente, libcurl admite los protocolos http, https, ftp, gopher, telnet, dict, file y ldap. libcurl también admite certificados HTTPS, HTTP, POST, HTTP PUT, subidas mediante FTP (también se puede hacer con la extensión FTP de PHP), subidas basadas en formularios HTTP, proxies, cookies, y autenticación usuario+contraseña.[14]

La librería cURL provee de una interfaz C para la comunicación con servidores HTTP, dicha interfaz es síncrona, eficiente, rápida y, además, permite la transferencia de archivos.

El procedimiento a seguir para la comunicación a través de la librería cURL podría resumirse en las siguientes fases:

- Inicio de la sesión y obtención de una clave (*"handle"*) que se usará para las siguientes funciones.
- Configuración de las características de la sesión, entre las que se encuentra el establecimiento de la URL del servidor.
- Configuración de las características de la transferencia.
- Ejecución de las operaciones que se deseen realizar sobre el enlace implementado.
- Liberación del canal de comunicación. [15]

En el *Anexo II*, se presentan las funciones de cURL más interesantes para el desarrollo del proyecto que, además, se corresponde con el procedimiento enunciado en este punto.

2.7. Single-board computer (SBC)

Un "***single-board computer***" (*SBC*) es un ordenador completo integrado en una sola placa, incluyendo: microprocesador/es, memoria, E/S y otras características requeridas en un ordenador funcional.

Los *SBC* fueron conceptuados como sistemas demostrativos o de desarrollo y, más tarde, como controlador computacional embebido. Actualmente, muchos tipos de *PC* o dispositivos portátiles integran todas sus funciones en una sola placa de circuito impreso.[16]

La evolución de estos sistemas, en conjunto con el desarrollo de los *SoC* ("*System-on-a-chip*"), ha permitido una mayor integración en ellas y la posibilidad de unos costes muy bajos, generalizando su uso para aplicaciones desde puramente docentes, hasta su utilización en sistemas comerciales.

El lanzamiento de dispositivos como: *BeagleBoard*, *UDOO*, *Arduino* o *Raspberry-Pi* ha generalizado su uso y ha abierto una gran posibilidad en el campo del desarrollo de aplicaciones sobre dispositivos embebidos por la facilidad de acceder económicamente a ellos.

2.7.1. Raspberry-Pi

Raspberry-Pi es un *SBC* de bajo coste, desarrollado en Reino Unido por la *Fundación Raspberry-Pi*, con el objetivo de proporcionar un hardware portable, completo y accesible con fines educativos y comerciales.

El diseño más extendido incluye un *System-on-a-chip* de la compañia **Broadcom**, modelo *BCM2835*, que contiene un procesador central (CPU) *ARM1176JZF-S* a 700 MHz (el firmware incluye unos modos "Turbo" para que el usuario pueda hacerle *overclock* de hasta 1 GHz sin perder la garantía), un DSP, un procesador gráfico(GPU) VideoCore IV, y 512 MB de memoria RAM (aunque originalmente eran 256 MB). El diseño no incluye un disco duro ni unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente. El 29 de febrero de 2012 la fundación empezó a aceptar órdenes de compra del modelo B, y el 4 de febrero de 2013 del modelo A.

Ya en 2015 se han comercializado las Raspberry-Pi 2 modelo B, que supone el primer modelo de segunda generación de la compañía. Las mejoras incluidas en este modelo está en la sustitución de la CPU por una quad-core ARM Cortex-A7 a 900 MHz, además de la ampliación de la memoria RAM hasta 1 GB. Cabe destacar ciertas mejoras que ya se incluyeron en modelos superiores de la primera generación como eran:

- 4 puertos USB
- 40 pines GPIO
- Puerto HDMI
- Puerto Ethernet
- Conector combinado audio jack 3.5mm y vídeo compuesto
- Interfaz para cámara (CSI)
- Interfaz para display (DSI)
- Funcionamiento con Micro SD

La fundación da soporte para las descargas de las distribuciones para arquitectura ARM, Raspbian (derivada de Debian), RISC OS 5, Arch Linux ARM (derivado de Arch Linux) y Pidora (derivado de Fedora); y soporta lenguajes como son Python, Tiny BASIC, C, PERL y Ruby.

Aunque las versiones más potentes son capaces de aceptar plataformas que van desde Windows 10, AROS, Linux, Plan 9, RISC OS 5 o UNIX. [17]

2.7.2. WiringPi

WiringPi es una librería "*third-party*" desarrollada por *Gordons Projects*, que se atribuye la autoría de librerías, también basadas en C, para otras plataformas embebidas como Arduino.

Esta librería provee de funciones para el control a nivel hardware del dispositivo Raspberry-Pi tales como: control de las entradas/salidas de propósito general (GPIO), control de modulación por ancho de pulso (PWM), interrupciones y trabajo *multithread* sobre el dispositivo. [18]

3. Análisis

En este apartado se incluye una descripción detallada de la solución elegida para la problemática expuesta anteriormente. Dicha descripción se enfocará desde un primer punto de vista de alto nivel y se irá descendiendo a niveles inferiores de abstracción.

3.1. Arquitectura general del sistema

Se expone una primera descripción general del sistema desde el punto de vista funcional de cada uno de los componentes físicos que lo forma. En la siguiente imagen se puede ver un diagrama de bloques que representa los distintos componentes que forman el sistema y el medio físico de interconexión entre ellos:

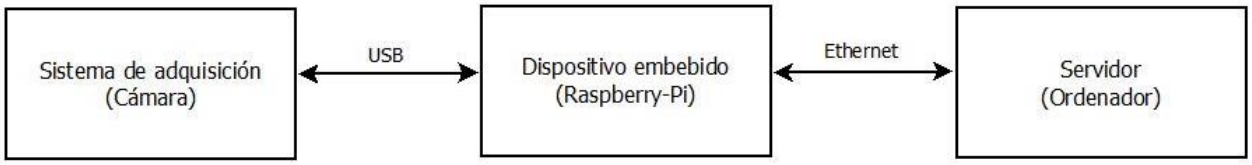


Figura 17: Diagrama de los componentes del sistema

El extremo inicial del sistema lo forma el dispositivo de adquisición que, en éste caso, es una cámara digital la cual, a través de una interfaz USB, transporta un stream de video digital o imágenes digitales hacia el dispositivo embebido a petición de éste.

Entrando en más detalle en el bloque del dispositivo embebido, se puede ver cómo se gestiona internamente la comunicación y se podrá realizar una descripción con mayor fidelidad del funcionamiento planificado para el sistema. Dicho funcionamiento se ilustra con la siguiente imagen:

Figura 18: Diagrama del interior de la Raspberry-Pi

Las imágenes son solicitadas a la cámara por el programa de procesado a través de la interfaz USB y recibidas del mismo modo.

Una vez realizado el reconocimiento por parte del programa, se llevan a cabo las consultas (mediante cURL) dirigidas a la base de datos. Esto se realiza mediante una llamada al método GET enviado al servidor HTTP estándar (Apache) a través de la interfaz Ethernet del dispositivo embebido.

En la siguiente imagen se puede ver de forma detallada los bloques que conforman el interior del servidor:

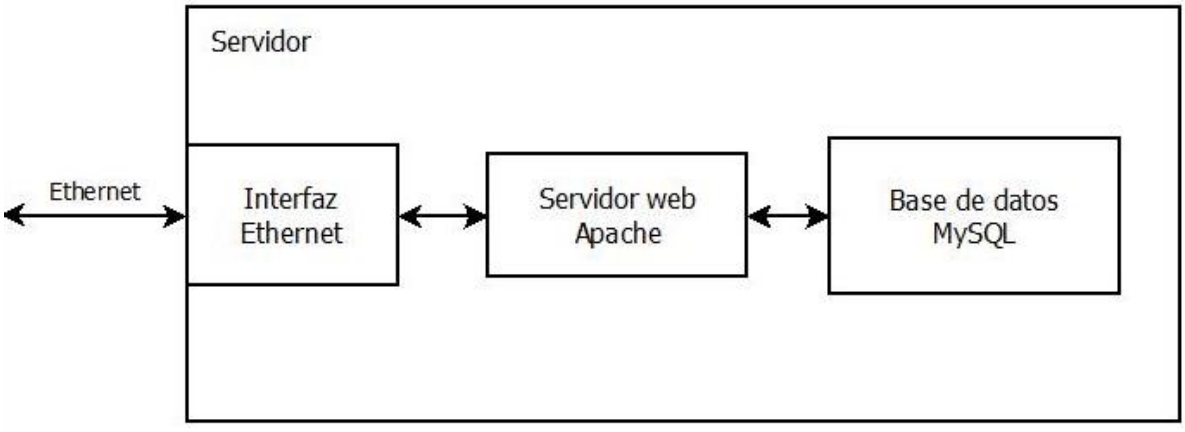


Figura 19: Diagrama interno del Servidor

Tal y como se ve, el servidor web (Apache) recibe la petición a través de la interfaz Ethernet del servidor y ejecuta el script oportuno de entre los que tiene alojados. Estos serán los encargados de establecer comunicación y enviar las órdenes destinadas a la base de datos (mediante comandos SQL).

La respuesta generada por parte de la base de datos seguirá el camino en dirección contraria siendo recibido por el programa de procesado. Éste será el encargado de decidir y enviar las órdenes oportunas al control de las GPIO.

3.2. Escenario propuesto

Con el fin de acotar el entorno en el que se va a trabajar, se establece un escenario genérico que reúna todas las características comunes en este tipo de sistemas con la idea de mantener una gran capacidad de generalización dentro de las limitaciones impuestas por las tecnologías elegidas.

En este punto, cabe destacar que los elementos elegidos para el sistema de gestión suponen un prototipo demostrativo el cual habría que adaptar o extender a las necesidades concretas de un sistema real.

Se propone, por tanto, un escenario con llegada frontal o lateral indistintamente, aunque con la condición de una posición frontal en los últimos metros. Relativamente indistinto a la posición en pendiente en los últimos metros, siempre y cuando ésta no sobrepase un 10% respecto a la horizontal en cualquiera de los dos sentidos.

Se incluye, aunque de forma no crucial, un elemento de cubierta sobre el sistema con el fin de su protección frente a los fenómenos ambientales y una mayor independencia de éste con respecto a la luminosidad variable.

El sistema estará formado por una columna de acero plegado de un metro de altura, que alojará el dispositivo embebido así como el elemento de adquisición de imágenes. Este último no tendrá frente a él ningún elemento de protección que pueda devaluar la calidad de las imágenes. Dicha columna estará comunicada a través de la interfaz Ethernet del dispositivo a la red interna del recinto y, a través de la interfaz *USB*, al elemento de adquisición.

Se precisa de la incorporación de una barrera de acceso de las características que el escenario concreto precise, interconectada con el dispositivo embebido a través de las entradas y salidas analógicas de las que disponga y que controlarán la apertura o cierre de la misma.

Además, para prevención de posibles casos en los que, por el motivo que sea, no se pueda llevar a cabo el reconocimiento visual, se prevé la posible incorporación de un sistema auxiliar de intercomunicación con el departamento encargado del acceso.

Los distintos elementos que se han enunciado para el escenario de partida propuesto, se pueden resumir e ilustrar en siguiente diagrama:

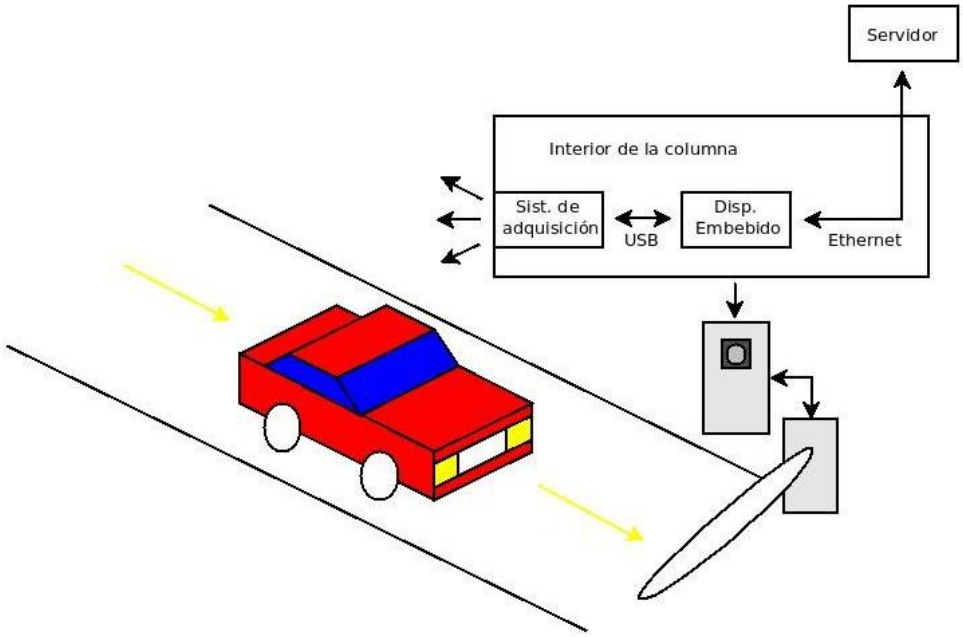


Figura 20: Diagrama ilustrativo del escenario propuesto

Se observa la disposición comentada de los elementos dentro de la columna y su interconexión, con lo que se puede ejemplificar la idea que se pretende desarrollar.

En el Anexo III, se adjunta un presupuesto basado en el escenario que en éste apartado se expone.

3.3. Sistema de Adquisición

El dispositivo de adquisición deberá cumplir previamente las condiciones dispuestas en las premisas establecidas para el proyecto, es decir, una cámara digital de bajo coste que sea capaz de aportar la suficiente calidad de imagen que permita la correcta definición de los dígitos de la matrícula.

Por otro lado, con vistas a la interconexión con el dispositivo embebido, se hace necesaria una interfaz de salida *USB* y, su capacidad de adquisición en condiciones adversas o nocturnas se verá sujeta a las necesidades de su aplicación concreta de uso.

Existen gran cantidad de dispositivos comerciales que reúnan las características solicitadas, en diferentes gamas de prestaciones. Se prevé la posibilidad de la utilización de cámaras de gama media con la incorporación de una etapa de calibración y/o preprocesado software con vistas a obtener rendimientos mayores.

Finalmente, con el fin de establecer una cota de prestaciones mínimas, se ha optado por una cámara de la marca *Creative*, concretamente el modelo "*Webcam Live!*

Ultra". La cámara puede ilustrarse con la siguiente imagen:

Figura 21: Elemento de adquisición

Y atesora las siguientes especificaciones técnicas:

Característica	Descripción
Sensor:	CCD
Lente:	76° ultra wide-angle
Máxima resolución de imagen:	1280 X 960 (realzada por sw)
Mayor resolución de vídeo:	640 X 480; 30 fps
Foco:	Anillo manual de ajuste focal
Interfaz:	High-Speed USB 2.0 (también compatible con USB 1.1)
Zoom:	4X Digital

Este dispositivo hará las veces de sensor de presencia mediante un algoritmo de umbralización por *frame* de referencia con el fin de minimizar el despliegue y sintetizar, lo máximo posible, el proceso de adquisición y posterior tratamiento de la imagen.

Respecto a las características de la información adquirida por la cámara, se optará por la elección de vídeo a 30 fps y a la máxima resolución permitida (640x480) para llevar a cabo la función a la que se hacía referencia. Las imágenes serán adquiridas en escala de grises con el fin de simplificar, tanto el trabajo de sensorización como el posterior procesado de imágenes fijas.

3.4. Dispositivo embebido

El dispositivo embebido es el centro del proyecto ya que será el encargado de realizar el control de todo el sistema: desde la adquisición de la cámara, el procesado local y extracción de información, comunicación y verificación con la base de datos alojada en el servidor remoto, así como, el control de la valla de seguridad.

Estas funciones hacen capital la necesidad de una notable capacidad de procesado por parte del dispositivo, además de una traducción de ésta en un tiempo de realización aceptable (aunque no es un parámetro limitante de la aplicación) y un gran porcentaje de acierto.

Todas estas especificaciones, unidas al objetivo de coste mínimo, llevan a decantarse por los actuales *SBC* y, concretamente, a la elección del dispositivo Raspberry-Pi 2 con distribución Linux/Raspbian. Dicho dispositivo provee, además de las entradas y salidas necesarias para la interconexión del sistema, de un *SoC* que contiene una CPU *quadcore* a 900 MHz, lo cual lo equipara, con la misma o mayor capacidad de procesado, a cualquier sistema tradicional a un precio mucho más bajo.

A continuación se muestra una imagen de la placa del dispositivo:

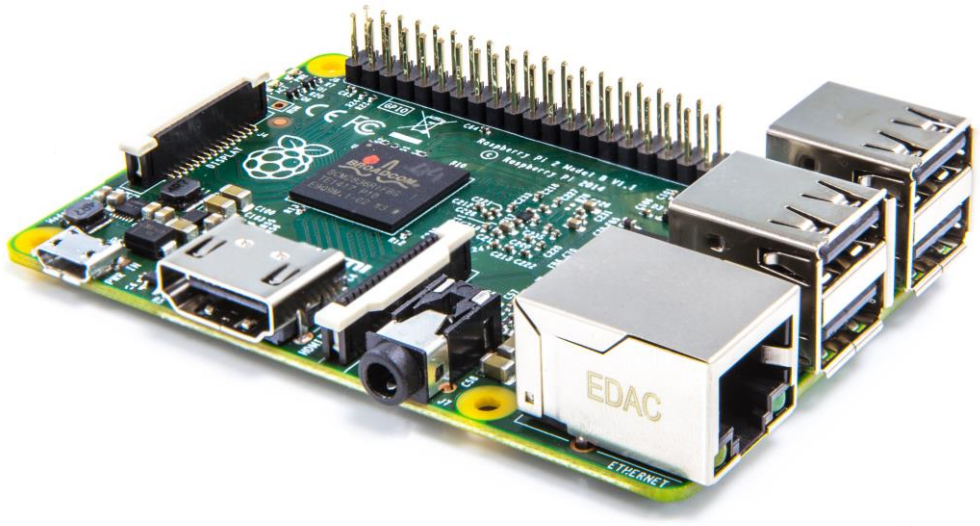


Figura 22: Dispositivo embebido Raspberry-Pi 2 Model B

Cabe destacar la disposición de las interfaces de E/S y las GPIO que posee el dispositivo, que lo hacen propicio para esta aplicación y permitirán cumplir todas las necesidades que se solicitan a la unidad de procesado.

En referencia a la unidad de procesado de imagen embebida dentro del dispositivo, cabe destacar que se opta por la algoritmia basada en Morfología Matemática

sobre escala de grises, lo que permite una mayor independencia respecto a la escena por la diversidad de colores de los vehículos objetivo. Estas herramientas, permiten el alejamiento de las técnicas tradicionales más enfocadas a la extracción de bordes y posterior evaluación de los límites de la matrícula, según características de color.

La comunicación con el servidor se realizará a través de la interfaz Ethernet que, mediante un cable UTP, conectará el dispositivo con el servidor central o con el elemento de *routing* de la red interna más cercano.

3.5. Servidor

Con el fin de modelar un sistema real, se prevé el uso de un computador capaz de alojar un servidor web junto con un servidor MySQL y la capacidad de realizar un procesado y comunicación con tiempos que no lastren al sistema, lo que no limita demasiado la elección del mismo. Para la realización del modelo, se opta por un *laptop* de Sony, concretamente el modelo "*Vaio*", con sistema operativo Linux/Ubuntu.

La algoritmia utilizada para la comunicación será identificable como un sistema de cliente-servidor, en el que el dispositivo hará peticiones de confirmación y/o registro al servidor y éste, tras realizarlas, confirmará su correcta o incorrecta ejecución.

4. Diseño

En este apartado se desarrolla una descripción de los módulos y de la realización del proyecto, dividido en fases. Dicha descripción se enfocará desde un punto de vista de la metodología de desarrollo para la consecución del proyecto.

4.1. Introducción

Respecto al diseño y la implementación de la aplicación es importante considerar una serie de factores iniciales.

El procesado de imagen y la librería OpenCV realizan, en ocasiones, operaciones que resultan poco eficientes en memoria, esto implica la necesidad de ir liberando la memoria de programa asociada al procesado además de ser realmente cuidadoso con el uso de la misma. Como consecuencia, se optó por una solución formada por dos programas:

- *CSensorPresencia*: Programa de inicialización de la aplicación y maestro del sistema.

- *CProcesadorMorfologico*: Programa esclavo, activado de forma remota, encargado de realizar las funciones de procesado y comunicación con la base de datos.

Esto permite que el programa esclavo, el cual supone el mayor gasto de memoria, no sea persistente en la ejecución de programa, evitando así que pueda acumular uso de recursos en las repetidas ejecuciones del mismo.

Teniendo en cuenta estas consideraciones, se muestra un diagrama que ilustra la interconexión de las clases principales del sistema, que posteriormente serán explicadas con detalle:

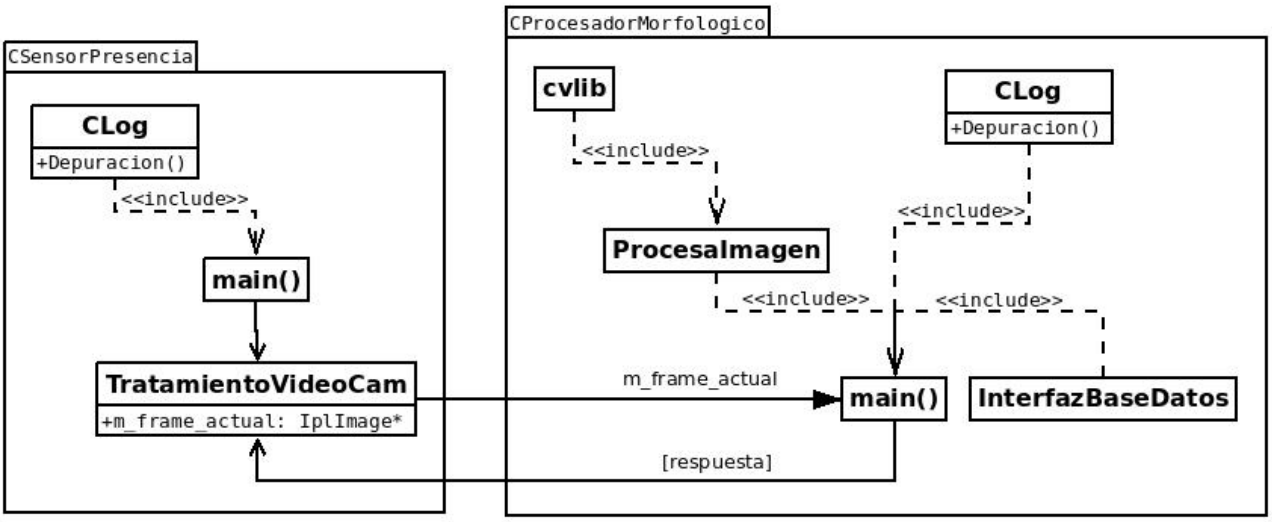


Figura 23: Programas y clases principales de la aplicación

Del diagrama anterior, que pretende dar una visión global del sistema, cabe explicar que *Clog* es una clase implementada para aportar una salida de depuración en un archivo concreto. Esta herramienta ha sido utilizada durante toda la implementación del proyecto con fines de *debug* y control del tiempo de procesado. Finalmente, se ha restringido su uso con la idea de mantener un historial de procesado en el dispositivo local.

4.2. Sensor de presencia (clase *TratamientoVideoCam*)

La clase *TratamientoVideoCam* es la clase principal del programa *CSensorPresencia*. Dicho programa se encarga de instanciar y lanzar un objeto de la clase, la cual habilita y configura las salidas digitales del dispositivo embebido, habilita el dispositivo de adquisición, lo configura, realiza su calibración y, tras establecer una imagen fondo de referencia, comienza a obtener la diferencia entre la imagen adquirida y la de fondo.

El valor diferencia mencionado (`acum_pixel`) dictamina el momento en el que el vehículo tiene una posición adecuada respecto al dispositivo de adquisición para obtener la imagen fija que será analizada. En ese momento, se almacena la imagen que posee la diferencia superior al umbral, se realiza una bifurcación del proceso y se activa el programa de procesado (*CProcesadorMorfologico*), dejando éste a la espera del resultado.

Una vez realizado el procesado de la imagen almacenada, análisis y comunicación con la base de datos, se recibe el resultado de dicho proceso (`resultado_procesado`) y, según éste, realiza una acción concreta.

Frente a un reconocimiento positivo con adecuado registro, se procede a activar la salida de la *GPIO* que activa la subida de la barrera y pausa el programa los sesenta segundos aproximados que tardaría el vehículo en acceder completamente al recinto. Tras eso, da orden a la barrera para bajar, reinicia sus registros y vuelve a su estado de reposo.

En el caso de un reconocimiento negativo o un reconocimiento positivo con imposibilidad de registro, se deja un margen de diez *frames* hasta tomar otra imagen y, siempre que el valor diferencia no haya salido de la zona de alerta, se realizan una serie de reintentos con un límite establecido (`num_analizadas`).

Llegado a ese límite, se negará la entrada, se pausará el programa y se emplazará al conductor a contactar con el responsable del recinto con el fin de que, si procede, le dé acceso manual. Tras un tiempo establecido para tal fin, reinicia sus registros y vuelve a su estado de reposo.

La clase actúa como una máquina de estados que establece tres estados reconocibles: estado de "*Inicialización*", "*Reposo*" y "*Alerta*". El funcionamiento enunciado se puede ilustrar mediante el siguiente diagrama:

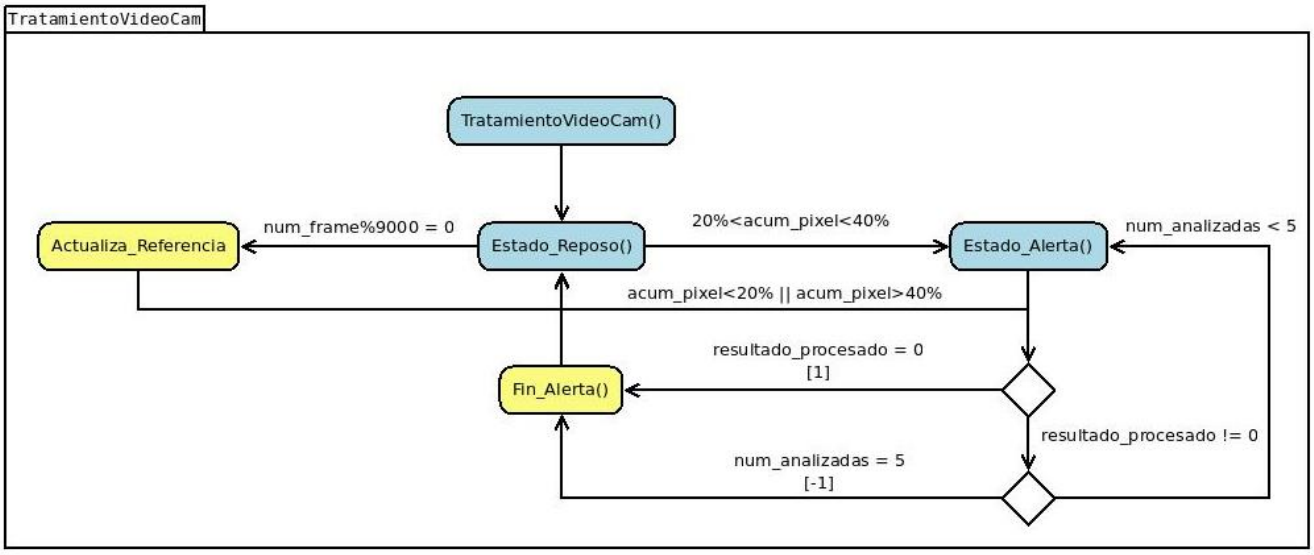


Figura 24: Clase TratamientoVideoCam

Los elementos del diagrama resaltados en color azul, representan los estados mencionados, mientras que en el caso de los resaltados en amarillo se trata simplemente de los estados de paso: *Actualiza_Referencia* y *Fin_Alerta*, que realizan la actualización periódica (controlado por `contador_acumulado`; variable que se incrementa con imágenes con el mismo valor de `acum_pixel`) de la imagen de referencia y la re-

inicialización de los registros de la clase, respectivamente.

4.3. Ampliación de la librería OpenCV (clase *cvlib*)

Dada la solución seleccionada y los requerimientos de la misma, se hizo capital la implementación de funciones que, haciendo uso de las pertenecientes a la librería OpenCV, ampliara la misma y aportara operaciones concretas que resultan de uso recurrente en el desarrollo de la aplicación.

El primer paso para el diseño del programa de procesado fue implementar una ampliación de la librería OpenCV mediante la clase *cvlib*. Se puede definir dicha clase como un juego de funciones estáticas enfocadas al tratamiento de imagen y al corrector/analizador sintáctico necesarios para el funcionamiento del sistema.

Dichas funciones están enunciadas y explicadas de forma detallada en el segundo apartado del *Anexo I*.

4.4. Procesador (clase *ProcesaImagen*)

La clase *ProcesaImagen* es la componente de procesado incluida dentro del programa *CProcesadorMorfologico* y se encarga de la extracción de información de interés a partir de la imagen que *CSensorPresencia* ha seleccionado para tal fin.

Crea un objeto “procesador” y, mediante la función *Procesar,* acepta dicha imagen a su entrada. Imagen que, tras un proceso de segmentación y posterior análisis, trata de devolver los dígitos que conforman la matrícula del vehículo. Hace uso de las funciones definidas en *cvlib* para llevar a cabo su labor.

El siguiente diagrama da una idea del contenido y forma de trabajar de la función *Procesar*:

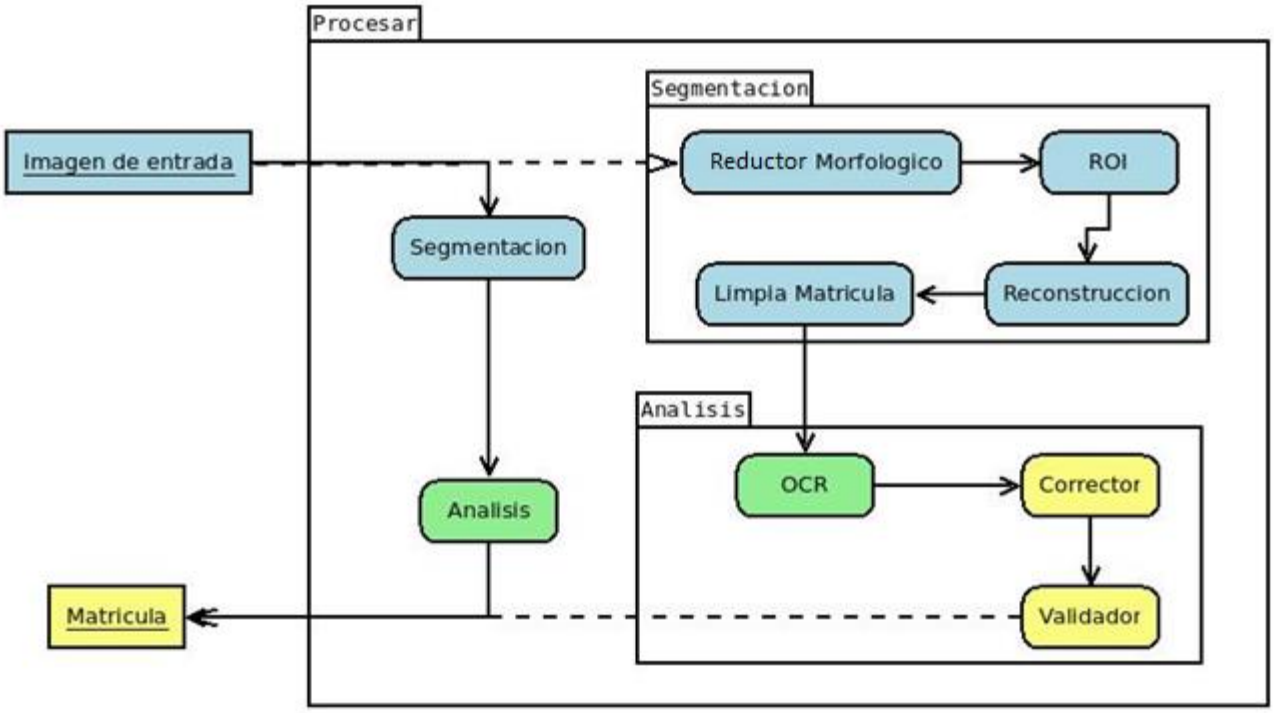


Figura 25: Función "Procesar" operación principal de la clase "ProcesaImagen"

Se puede apreciar como la función de procesado se divide en tres tipos de fases: fase de tratamiento y operaciones sobre la imagen (remarcadas en azul), fase de conversión de imagen a dígito (remarcada en verde) y fase de tratamiento y operaciones sobre los dígitos (remarcadas en amarillo).

Desgranando un poco más la función puede observarse que el procedimiento que usa es el siguiente. En primer lugar se realiza una reducción de la información de fondo de la imagen mediante operaciones morfológicas y umbralizado, de dicha reducción obtenemos formas rectangulares cerradas que podrían ajustarse a las características de la matricula esperada, se trata de las consideradas *ROIs*. Las *ROIs* son posteriormente filtradas a partir de su tamaño y relación de aspecto, dejando finalmente una sola región de interés.

De la ROI que queda tras el filtrado mencionado se guardan datos como su ángulo respecto a la horizontal y el punto central de la misma.

Se continúa entonces con una fase de *Reconstrucción Morfológica* a partir de una imagen obtenida en el proceso de reducción y el uso de la región como máscara, cuyo resultado se rota con los datos obtenidos de la *ROI* para facilitar el trabajo del *OCR*.

Finalmente, se filtra la imagen reconstruida para eliminar cualquier elemento que, por tamaño, no pueda ser un dígito de la matrícula.

Pasando ahora al análisis de la imagen obtenida en la segmentación, comenzamos ejecutando el *OCR* y, tras él, mediante el corrector, eliminando todos los caracteres obtenidos que no puedan formar parte de una matrícula, es decir: espacios,

signos de puntuación, letras en minúscula, caracteres especiales, etc... Trabajando únicamente con las letras en mayúscula y los caracteres numéricos.

Como último paso del procesado se contrasta la cadena de caracteres obtenida del corrector con máscaras que representan todas las codificaciones reconocidas por la Unión Europea. Este método de validación permite, además de comprobar si existe correspondencia, obtener una o varias posibles procedencias del vehículo que porta la matrícula.

Cabe destacar que entre cada una de las fases que operan con imágenes se realiza una comprobación de imagen vacía que aborta la ejecución y, de forma similar, en las fases que operan con caracteres se evalúa la longitud de la cadena para dictaminar si se continua hacia el siguiente paso.

4.5. Base de datos y *scripts PHP*

La base de datos MySQL, llamada *Control_Acceso* y creada mediante *PHPMyAdmin* para ejemplificar un uso real en el modelo, consta de tres tablas: "*Id_Matricula"*, "*Llegadas"* y "*Salidas"*; que se pueden observar de forma gráfica en el siguiente diagrama.

Figura 26: Tablas de la base de datos Control_Acceso

Como se puede apreciar, las tablas están interrelacionadas mediante el identificador de entrada (*ID*), el cual es la clave primaria en cada una de ellas. Esto permite acceder a los datos de llegada o de salida de un vehículo simplemente por su índice de entrada.

Cada entrada de la base de datos poseerá, por defecto, un identificador que estará reflejado en las tres tablas (*ID*), un código de matrícula que defina al vehículo (*Matricula*), el país de procedencia del mismo (*Procedencia*), fecha en la que se espera su llegada (*Fecha_esperada*) y hora de la misma (*Hora_esperada*), además de una variable que dictamina la posición relativa del vehículo con respecto al recinto (*Situación*), la cual puede ser: '*Esperando*', '*En_Base*' o '*Servido*'. Dicha variable estará reflejada en las tablas de *Llegadas* y de *Salidas*, y será utilizada a modo de *flag* para indicar cuales de las variables, por defecto nulas, deberán estar completas.

Los *scripts* PHP serán los encargados de trabajar como intermediarios entre el objeto interfaz y la base de datos, modelando y ejecutando las peticiones que el objeto les solicite y devolviendo a éste la respuesta a dicha ejecución. También asegurarán la no corruptibilidad de las tablas, es decir, que las *ID's* sean correlativas, que no existan *ID's* o matriculas/procedencias repetidas, que no se puedan manipular las entradas con *Situación = Servida*, etc...

Los *scripts* implementados se pueden ver de forma detallada e ilustrativa en el último apartado del *Anexo II*.

4.6. Comunicación con BBDD (clase *InterfazBaseDatos*)

La clase *InterfazBaseDatos* modela las operaciones de comunicación con la base de datos necesarias en la aplicación y, junto con la clase *ProcesaImagen*, completa las funciones asignadas al programa *CProcesadorMorfológico*.

Dicha clase genera objetos de tipo "interfaz" que poseen la capacidad de realizar operaciones sobre la base de datos mediante el protocolo *HTTP* y la ejecución de *scripts* PHP. Las operaciones que realiza la clase son las de: búsqueda y obtención de datos (*BusquedaBD*), inserción de entradas (*InsertaEntradaBD*), actualización de una celda (*ActualizaElementoBD*), eliminación de entrada (*EliminaEntradaBD*) y, con vistas a la aplicación, registro de llegada (*RegistraLlegada*) y registro de salida (*RegistraSalida*).

De las operaciones enunciadas, el modelo desarrollado para la aplicación utiliza únicamente la de búsqueda (*BusquedaBD*) y la de registro de entrada (*RegistraLlegada*), que se explicarán con más detalle, aunque se han implementado el resto con vistas a las necesidades que pueda requerir la integración del sistema en un recinto real con una base de datos ya implementada.

La operación de búsqueda se aplica con el fin de comprobar la existencia de una entrada en la base de datos asociada a la matrícula obtenida por el procesador. Dicha operación se puede encontrar frente a dos situaciones:

- Que esta entrada exista, en cuyo caso le será devuelta la fila de la tabla "*Id_Matricula*" que la contiene, obteniendo así el *ID* de la entrada y la procedencia real del vehículo a fin de realizar una segunda comprobación contrastando esta procedencia con la supuesta por el procesador.

- Que la entrada no exista, esto provocará que reciba una negativa por parte del *script*.

El funcionamiento enunciado para la operación de búsqueda, se puede ilustrar con el siguiente diagrama:

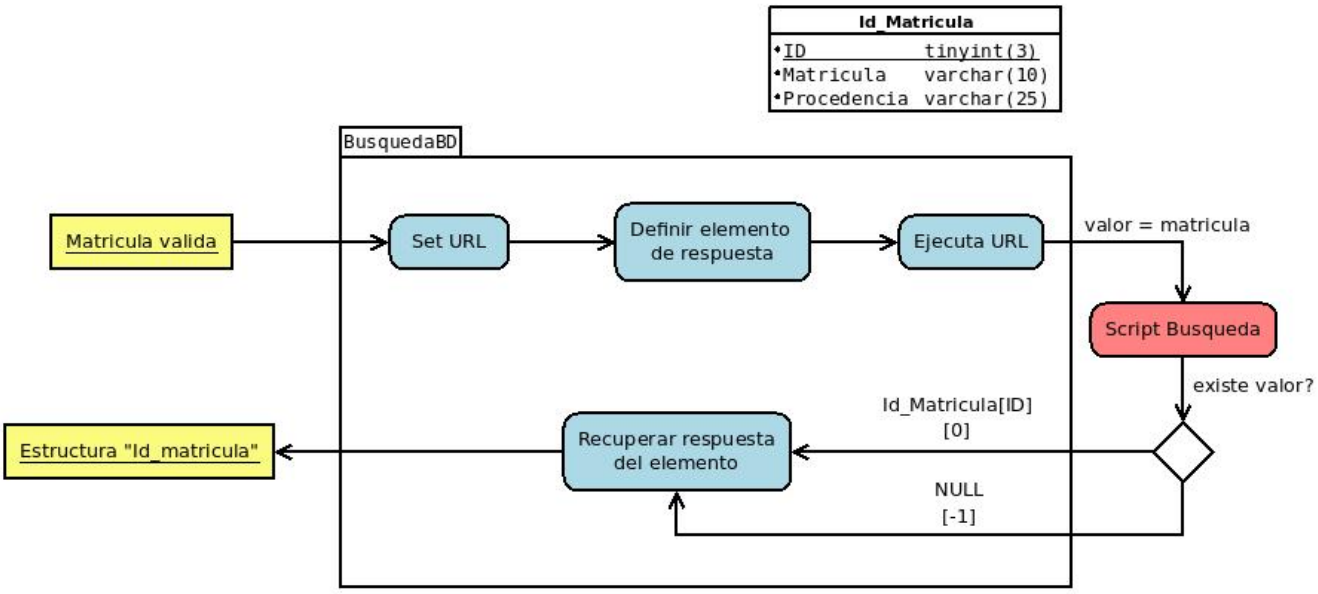


Figura 27: Función "BusquedaBD" operación de la clase "InterfazBaseDatos"

Como se puede ver, la matricula obtenida por el procesador aporta la información que se incluirá en la *URL* como valor buscado y se asociará dicha URL al mensaje que manda la interfaz a través de *cURL*. Se define un elemento de memoria que alojará la respuesta obtenida por el *script* y, tras eso, se ejecuta el *script* a través de la *URL* modelada.

Tras recibir la respuesta, se trata de forma local según sea su valor y, si éste es es positivo, se guardan los datos obtenidos en la estructura creada para tal fin.

Una vez obtenida la confirmación de la existencia de la matricula dentro de la base de datos y la comprobación de procedencia, entra en juego la operación de registro de llegada.

Esta operación se encarga de registrar la fecha y hora de llegada del vehículo y calcular la fecha y hora de salida estimada para el mismo, siempre y cuando éste se encuentre en situación '*Esperando*'. El dato de *ID* obtenido mediante la búsqueda, permite acceder directamente a su entrada en las tablas de "*Llegadas*" y "*Salidas*", comprobar cuál es su situación y, en caso de que se le estuviera esperando, hacer un registro del mismo según la fecha y hora. Una vez registrada la llegada y calculada la salida, se modifica también su situación pasando a estar '*En_Base*'.

En caso de que la situación del vehículo respecto al recinto no sea '*Esperando*' se supone que el procesador ha cometido un error en la identificación del vehículo y, por tanto, el *script* devuelve una negativa como respuesta a la petición de la interfaz.

El funcionamiento enunciado para la operación de registro de llegada, se puede ilustrar con el siguiente diagrama:

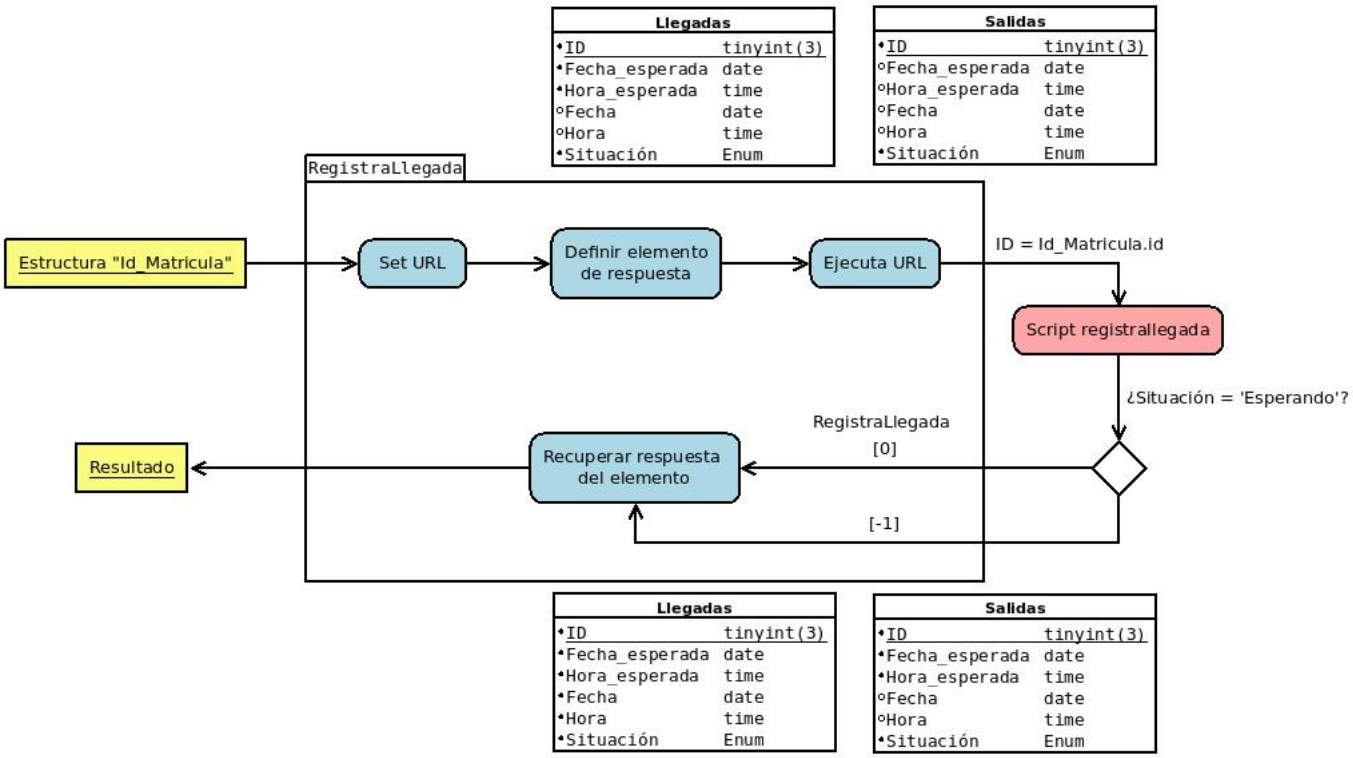


Figura 28: Función "RegistraLlegada" operación de la clase "InterfazBaseDatos"

Se puede apreciar como la mecánica de la función es similar a la de búsqueda, con la salvedad de que en este caso no se precisa que el *script* devuelva información más allá del resultado positivo o negativo de la operación de registro.

Por otro lado, en las tablas adjuntas al diagrama, se puede ver cómo, en caso de que la situación de la entrada sea la adecuada, el *script* modificará valores a priori nulos en ambas.

El ciclo se completa con la salida del vehículo del recinto que, en caso de implantar un sistema bidireccional, sería fácilmente ampliable ya que se dispone de la operación de registro de salida. Esta operación tiene un funcionamiento similar al registro de entrada explicado, con la diferencia de que el valor de situación que se debe encontrar en la entrada que contenga la matricula reconocida, será '*En_base*' en lugar de '*Esperando*'. Tras el registro de la salida, el valor de la situación será actualizado a '*Servido*'.

5. Resultados

En este apartado se desarrolla un ejemplo de funcionamiento de la aplicación desarrollada, además de un análisis y evaluación de los resultados cuantificables obtenidos del mismo. Dicho ejemplo describirá los distintos casos contemplados en la ejecución de la aplicación partiendo de un funcionamiento correcto y completo.

5.1. Introducción

Tal y como se ha indicado en el prólogo del capítulo, se va a realizar un ejemplo pormenorizado del funcionamiento de la aplicación detallando y explicando los distintos casos que se pueden producir durante su ejecución.

En primer lugar, veremos una ejecución que podemos definir como un caso ideal, en el cual, todas las fases de la aplicación se desarrollan con toda normalidad y, a partir de él, iremos desgranando el funcionamiento en casos que presenten más adversidad.

Previo al arranque y consecuente comienzo de la ejecución de la aplicación es vital la correcta interconexión de la cámara al dispositivo embebido, así como la conexión de éste con el servidor a través de la interfaz Ethernet.

5.2. Caso con ejecución ideal

El arranque del dispositivo embebido supone a su vez el comienzo de la ejecución. Tal y como se ha explicado en los capítulos anteriores, las tareas de instanciación de las salidas digitales del dispositivo, de la cámara, así como su configuración y calibración, serán los primeros pasos de la aplicación previos al comienzo de la adquisición a través de la misma.

La calibración del dispositivo consiste en la extracción de parámetros intrínsecos y extrínsecos del éste, con el fin de establecer unos mapas de compensación de distorsión y curvatura de la lente. Dichos mapas serán utilizados sobre la imagen adquirida, maximizando así la calidad de la misma.

Estos primeros pasos se pueden ver reflejados en el *log* de depuración incluido en la aplicación. Dicho *log*, reporta el siguiente mensaje:

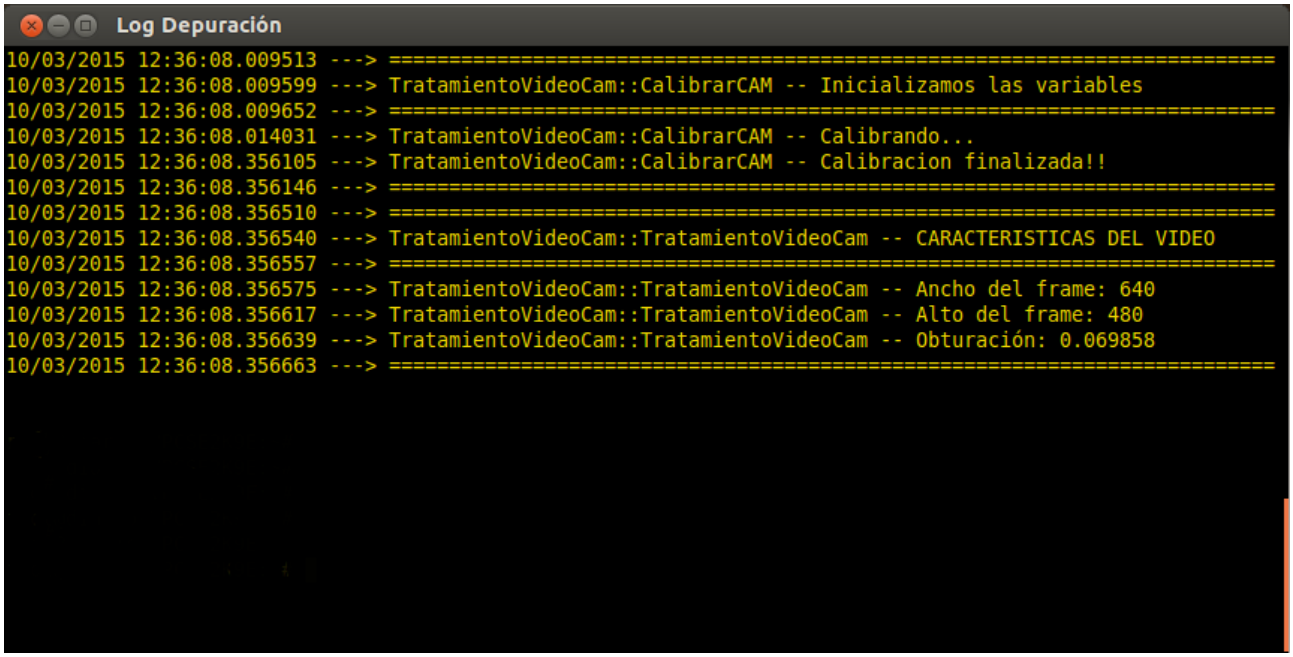


Figura 29: Depuración tras el arranque de la aplicación

Se puede observar cómo, tras realizar el proceso, la depuración aporta las características del vídeo que comenzará a adquirir la cámara.

Tras eso, la aplicación tomará una instantánea de referencia y comenzará a adquirir, realizará un filtrado pasobajo de la imagen adquirida y, tras él, calculará la imagen diferencia umbralizada del *frame* obtenido y la imagen de referencia. Ilustrado en su punto inicial por las siguientes imágenes:

Figura 30: Imagen de referencia inicial

Cabe mencionar que se ha configurado un valor de obturación alto, con el fin de independizar el correcto funcionamiento del sistema de los cambios de luminosidad por efectos atmosféricos y compensar la existencia de un voladizo como cubierta del acceso.

Figura 31: Imagen fija de lo obtenido por la cámara

Resulta obvia la gran similitud existente entre la imagen de referencia inicial y los *frames* siguientes en ese momento, ya que se puede estar hablando de unos pocos segundos de diferencia.

El resultado de la diferencia se umbraliza mediante el método de Otsu solo para niveles de umbral superiores a un valor fijo entre 0 y 255 (concretamente, 20) lo que lo hace independiente frente a pequeñas diferencias de luminosidad y el ruido que escapa a la calibración de la cámara. Esto evitará que se produzcan falsos positivos ya que el ruido acumulado no sobrepasará el umbral que lo haría saltar al estado de "*Alerta*".

Frente a diferencias más notorias este ruido de fondo desaparecería, como se puede ver en las siguientes imágenes.

Figura 32: Frame actual

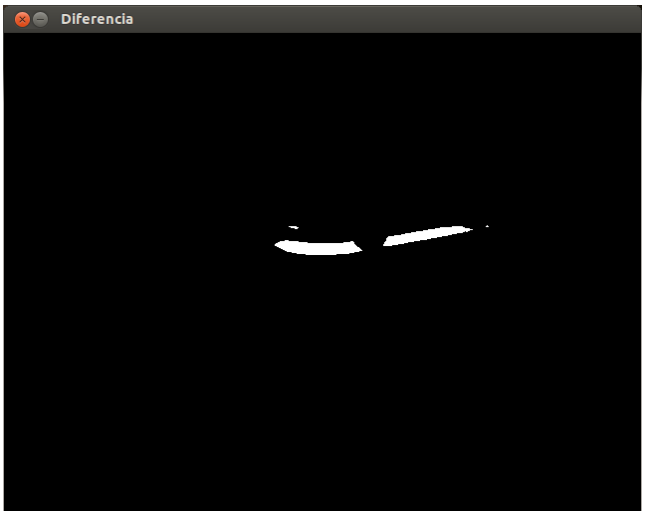


Figura 33: Diferencia binarizada a partir del frame actual

Estas imágenes pueden dar una idea del comportamiento del sensor de presencia frente a cambios de la escena ajenos a los objetivos de interés para el sistema.

La ejecución del sensor de presencia continúa en ésta línea, es decir, enmarcado en el estado de "*Reposo*" hasta el momento en que aparece un vehículo con intenciones de acceder al recinto.

Ese momento se traduce como un aumento en la acumulación de píxeles en la imagen diferencia, tal y como se puede ver en las siguientes imágenes:

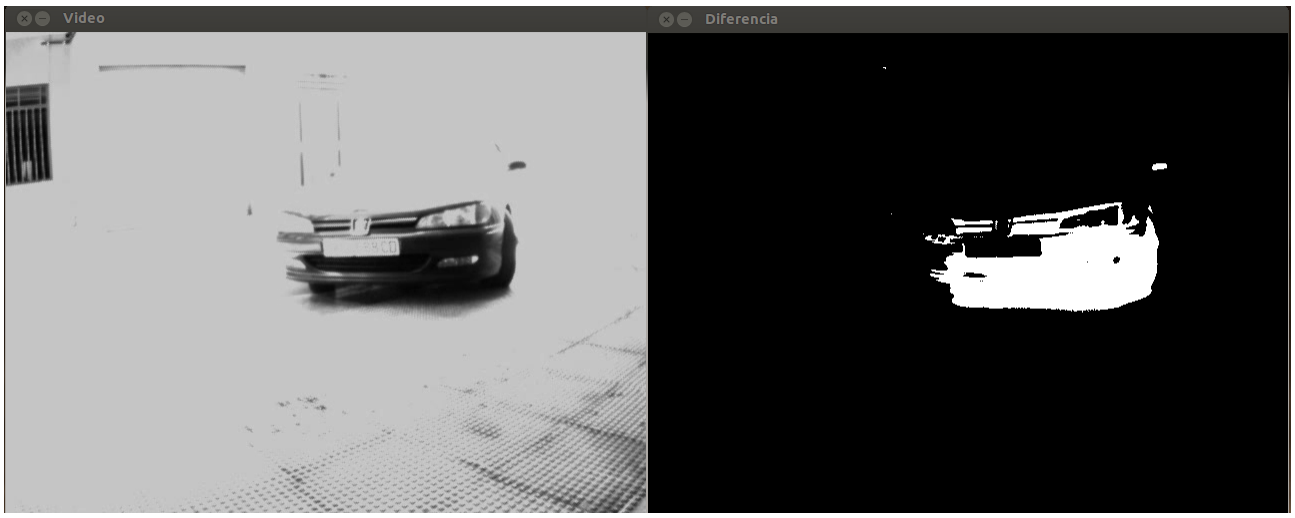


Figura 34: Frame actual y diferencia binarizada a partir de él

Sin haber sobrepasado, en ese instante, el umbral que dispara el salto al estado de "*Alerta*", es fácil ver como el número de píxeles presentes en la imagen

diferencia va aumentando y continuará su aumento en los instantes posteriores según se acerque el vehículo.

Dicho aumento llevará al punto en el que, inevitablemente, se produzca el salto al estado de "*Alerta*". Como ya se explicó, el primer *frame* del vídeo con el que se acceda al estado será siempre enviado al procesador, con el fin de que la respuesta del sistema sea la más rápida posible. En caso de que el procesador sea capaz de reconocer de forma correcta la matrícula a partir de éste *frame* la respuesta será óptima.

En el ejemplo que se sigue, el *frame* que activa el proceso enunciado y la diferencia que se encuentra a partir de él, se pueden ver en las siguientes imágenes:

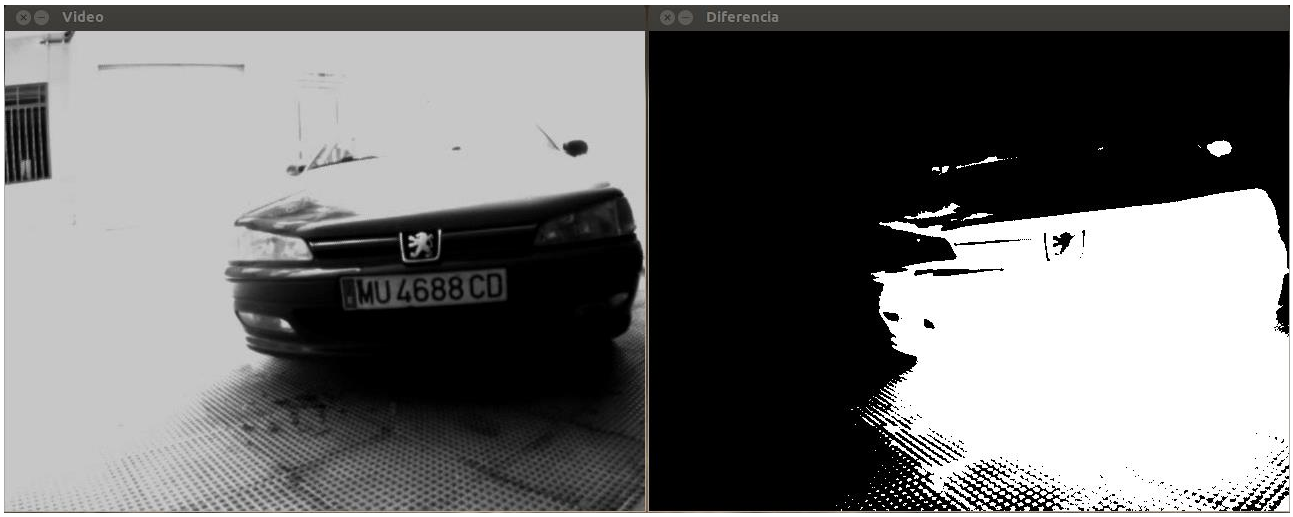


Figura 35: Frame que activa el estado de alerta y diferencia umbralizada a partir de él

Además, en el *log*, se registra la entrada en el estado de "*Alerta*" y se indica el número del *frame* (contado desde el inicio) que será analizado. En la siguiente imagen se ilustra éste registro del *log:*

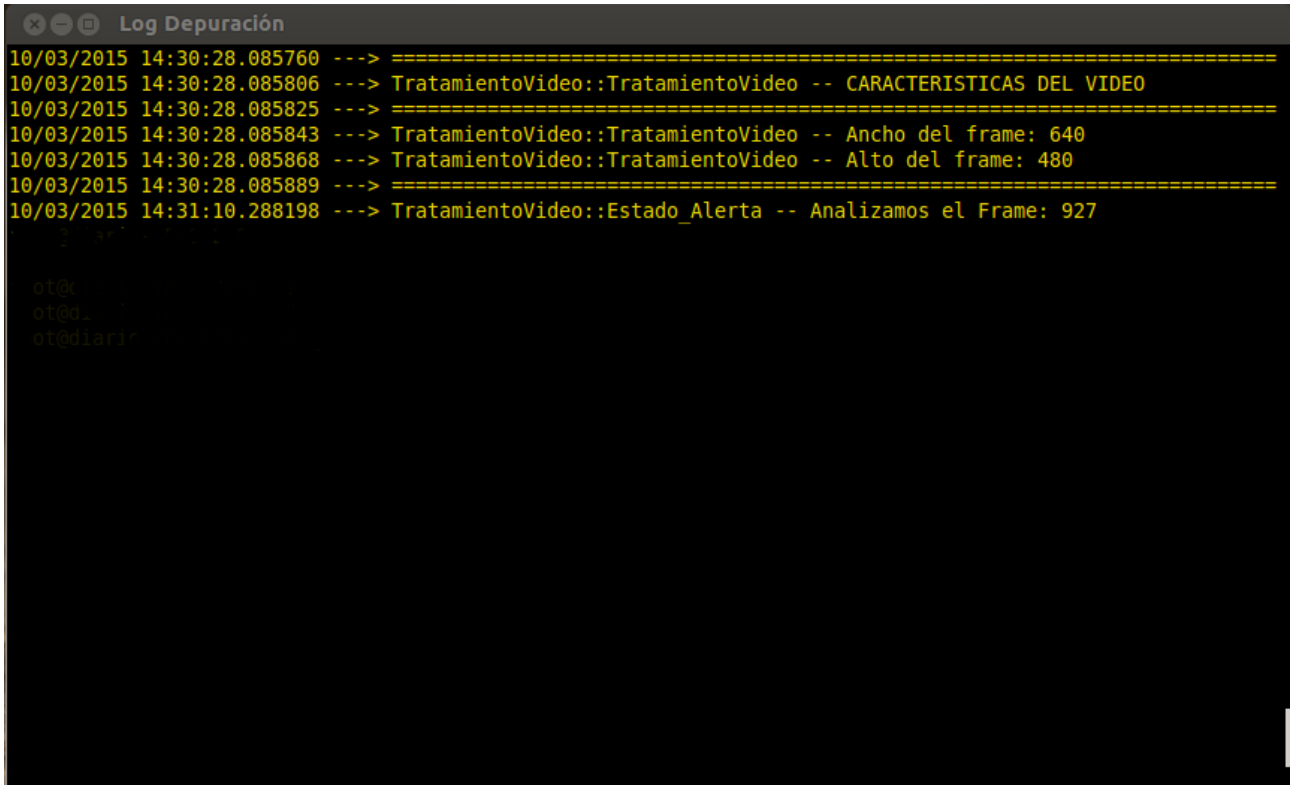


Figura 36: Depuración obtenida del paso al estado de alerta

En ese momento, el *frame* en cuestión es almacenado y se procede, por un lado, a la ejecución del programa esclavo y, por otro, a la congelación del programa maestro a la espera de resultados.

Se puede tomar una primera lectura de la memoria utilizada por la aplicación en dicho momento, el cual ya ha soportado un proceso de adquisición continuado y el paso por los dos estados principales del programa. El dato de memoria se puede ver en la siguiente imagen:

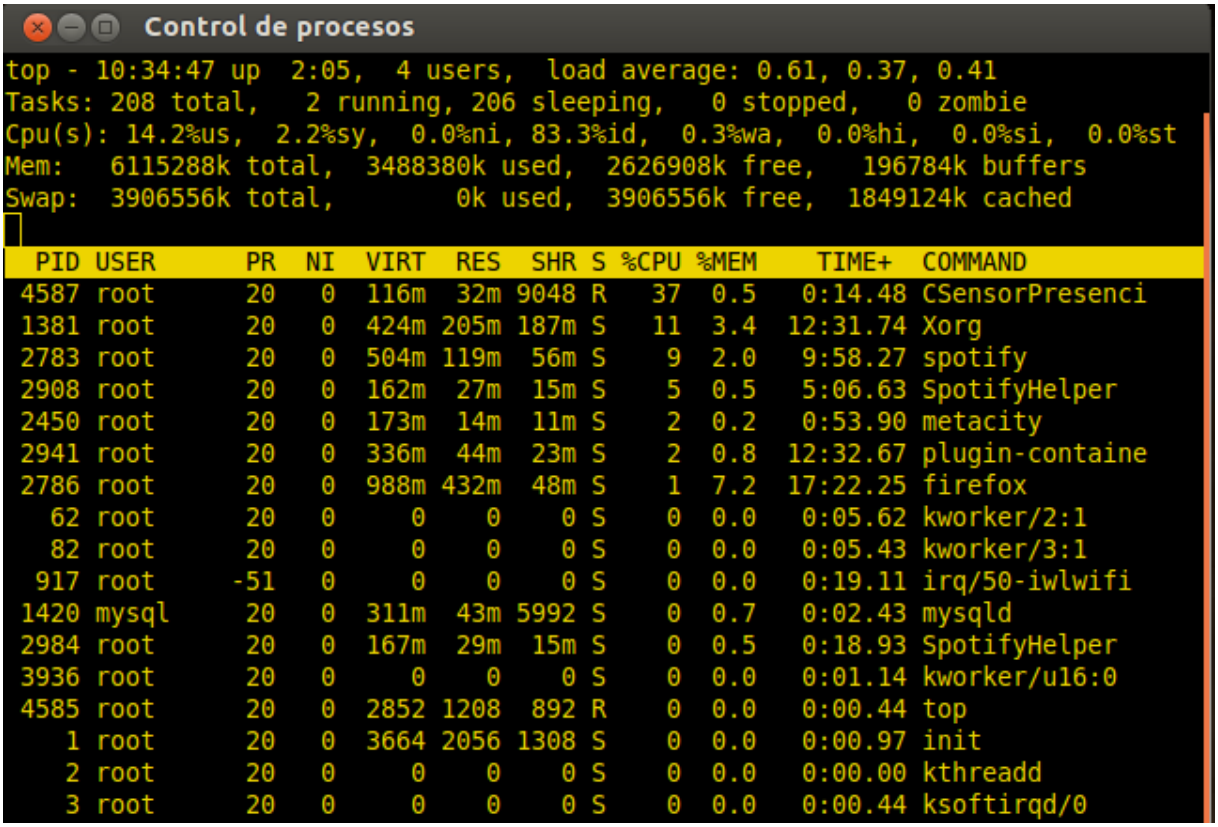


Figura 37: Control de los procesos activos en el SO

En la primera línea, se puede ver el identificador de proceso 4587 correspondiente al programa *CSensorPresencia* y, como primeras impresiones, se aprecia que aunque el uso de *CPU* es relativamente exigente, en memoria solo ocupa un 0'5% de la ofrecida. Esto puede resultar engañoso con vistas a la ejecución sobre el dispositivo embebido, puesto que esta ejecución se está realizando sobre el servidor que dispone de 5'8 GB de memoria, frente a 1 GB en el caso de *R-Pi*. Extrapolando el dato al dispositivo embebido, se asume un 2.9% de memoria usada en este instante.

Siguiendo el camino tomado por la ejecución, la atención se centra ahora en *CProcesadorMorfológico*. El programa instancia el objeto procesador y la interfaz con la base de datos. Volviendo a la línea de ejecución, se procede a la carga y orden de procesado del *frame* almacenado.

Por la forma en la que se ha configurado el dispositivo de adquisición, se evita en gran medida tener que realizar una costosa etapa de preprocesado, pudiendo comenzar con la segmentación de la imagen y suponiendo una mejora en el tiempo de procesado global. Además, con el fin de evitar procesados de imágenes que no sean correctas, se evalúan las características de la imagen cargada, abortando el procesado si éstas no fueran las esperadas.

Dentro ya de la etapa de segmentación, en primer lugar se realiza una mejora de la imagen mediante una apertura y posterior cierre morfológico con el fin de eliminar

las posibles manchas que la imagen y, concretamente, la matrícula puedan tener o haber cogido en la adquisición.

La primera operación de verdadera repercusión en la segmentación es una detección de los valles de la imagen (*Bottom-Hat* morfológico). Las matrículas en el formato europeo son, de forma general, con dígitos negros sobre un fondo de color claro además de reflectante (lo que incrementa su luminiscencia).

La operación morfológica se realiza mediante un elemento estructurante con un tamaño muy aproximado al que se presupone para los dígitos de la matrícula con lo que discriminaremos los elementos de mayor tamaño.

El resultado gráfico de la operación aplicada sobre el *frame* inicial, se puede apreciar en la siguiente imagen:

Figura 38: Imagen tras "Bottom-Hat" con cierre rectangular

Se puede ver cómo, salvo en zonas con luminosidad muy variante, se ha podido eliminar una importante parte del fondo de la imagen manteniendo los dígitos y obteniendo una imagen en negativo.

A continuación, tras evaluar la imagen resultante, se procede a la eliminación de los elementos conectados a los bordes de la imagen. Esto tiene su fundamento en el

conocimiento de que, por el enfoque del elemento de adquisición, se puede presuponer que la *ROI* se situará en la parte central inferior de la imagen.

Este proceso se lleva a cabo mediante un umbralizado y posterior aislamiento de los píxeles en contacto directo con los bordes de la imagen. Las regiones que contienen dichos píxeles se vuelven a reproducir por un proceso de crecimiento basado en reconstrucción geodésica, obteniendo una imagen que las elimina de la imagen de entrada con una simple resta.

El resultado obtenido de la eliminación de los bordes se puede ver de forma ilustrada en la siguiente imagen:

Figura 39: Imagen tras proceso de eliminación de bordes

Si se compara con la imagen resultado del *Bottom-Hat*, se puede ver como en las zonas resaltadas se han eliminado las regiones de luminosidad alta que existían y como, por supuesto, estas no tienen influencia alguna sobre la matrícula.

En este caso, por el contenido de la imagen concreta, el proceso de eliminación de bordes puede parecer que no supone un gran avance en la reducción de la información del fondo. El siguiente paso, se centrará en las regiones interiores de la

imagen y el conjunto de ambas supondrá un gran avance en la extracción de la región de interés.

Dicho paso consiste en una serie de detecciones de cimas (*Top-Hat*) mediante elementos estructurantes lineales de longitud superior al alto y ancho de los dígitos y rotados 30º de forma general respecto a los demás, componiendo así un haz de imágenes que en todo caso contendrán los dígitos de la matrícula. La búsqueda del valor mínimo de luminosidad de píxel en dicho haz de imágenes, aportará una imagen con una reducción de información importante en las zonas de bajo interés.

El resultado del proceso explicado se puede ilustrar con la siguiente imagen:

Figura 40: Imagen tras proceso de reducción

La comparativa del resultado obtenido con las imágenes anteriores da fiel cuenta de lo determinante que resulta este paso en la segmentación de la imagen, habiendo conseguido eliminar la mayor parte del fondo de la misma.

Llegada la etapa de segmentación a este punto, se hace necesaria la obtención de regiones compactas a partir de los píxeles que se han mantenido tras la reducción llevada a cabo. Esto se realiza mediante un cierre y posterior apertura de la imagen, con elementos estructurantes lineales de longitud igual al ancho y alto de los caracteres, respectivamente.

Se obtiene así, la siguiente imagen resultado con regiones compactadas:

Figura 41: Imagen tras proceso de compactación de regiones

Se puede ver que, aunque se ha devaluado notablemente el nivel de luminosidad de los píxeles, se han conseguido regiones geométricas entre las que se incluye la formada por la matrícula. Esto, unido a la posterior binarización (con bajo umbral) de la imagen, permite tener regiones reconocibles por forma y tamaño.

Además, considerando las dimensiones teóricas que se han dado a la matrícula, se puede aplicar una apertura para eliminar los espurios de menor tamaño. El resultado de dicha apertura sobre la imagen binarizada, consigue la imagen siguiente:

Figura 42: Imagen tras binarización por umbralizado y apertura para eliminación de espurios

Resulta sencillo ahora discernir, basándose en el tamaño y forma de la región, cuál de ellas es la que se superpone a la matrícula. Así actúa el selector de *ROI*, aportando a su salida, además de la imagen con la región única de interés, el punto central de ésta y su ángulo respecto a la horizontal. Esos datos serán de suma importancia para la adecuación de los dígitos con vistas a la lectura por parte del *OCR*.

Podemos ver en el *log* de depuración, además de la constatación de los pasos que se han ido realizando hasta ahora por parte del procesador, los valores que, en el ejemplo, tienen los datos mencionados:

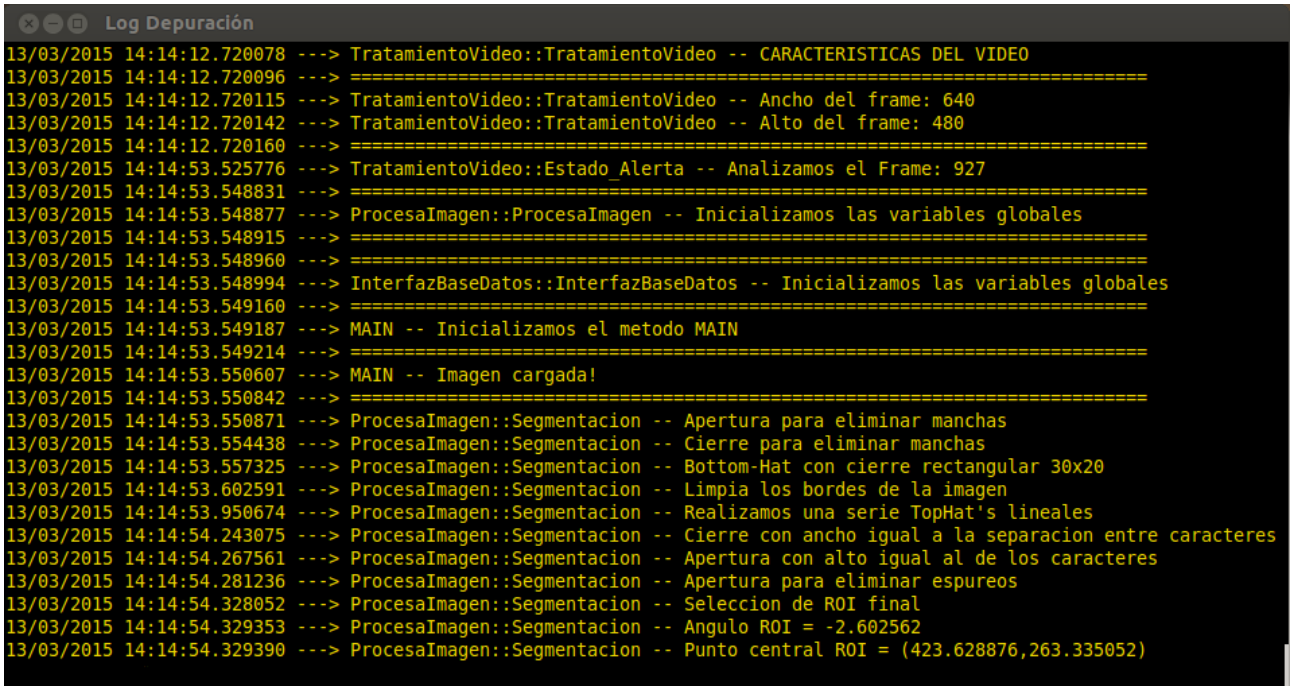


Figura 43: Depuración obtenida tras la selección de ROI final

Por tanto, en el ejemplo, el ángulo de la *ROI* será de -2'60º respecto a la horizontal, mientras que su punto central estará situado en torno al píxel situado en (424, 263).

Una vez obtenida la región aislada de la matrícula, es necesario volver a los caracteres que se encuentran bajo ella. Esto se consigue mediante la operación de reconstrucción geodésica utilizando la imagen que posee la *ROI* como máscara (Figura 41) y el resultado del *Bottom-Hat* inicial (Figura 36), como imagen marcadora.

Cabe decir que tanto la reconstrucción que se va a realizar ahora como la usada en el proceso de eliminación de bordes se realizan mediante dilatación geodésica. Esto permite realizar la operación actual, ya que tras las sucesivas dilataciones, se guarda el valor de luminosidad mínimo de cada píxel, y no el máximo como en la reconstrucción basada en erosión.

La imagen resultado de la reconstrucción es la siguiente:

Figura 44: Imagen tras reconstrucción geodésica

Junto con los caracteres que conforman la matrícula del vehículo, se introducen ciertos espurios provenientes de los elementos colindantes o de la propia matrícula, como puede ser el distintivo de la Unión Europea. Además, se aprecia la inclinación existente en ellos provocada por la rampa que posee el escenario del ejemplo.

La forma de subsanar estas deficiencias consiste en la siguiente secuencia de procesos. En primer lugar, se realiza un giro de la imagen de un ángulo igual al obtenido de la *ROI*, momento que permite realizar también un aumento por escalado de la misma que mejore la legibilidad de cara al *OCR*. Tras eso, y la binarización de la imagen rotada, se sigue un proceso similar al utilizado para seleccionar la *ROI*, es decir, desechar los elementos de la imagen que por tamaño (aunque en éste caso, no por forma) no puedan ser caracteres. Se obtiene así la imagen final de la segmentación:

Figura 45: Imagen final segmentada

Salvo por posibles defectos en la matrícula y deformidades producidas en algún carácter por la oblicuidad del vehículo respecto al dispositivo de adquisición, se obtiene una imagen con la definición y claridad necesarias para suponer un buen trabajo del *OCR*.

Además, en el *log* se puede ver el proceso de segmentación al completo y hacer una medida del tiempo necesitado para él:

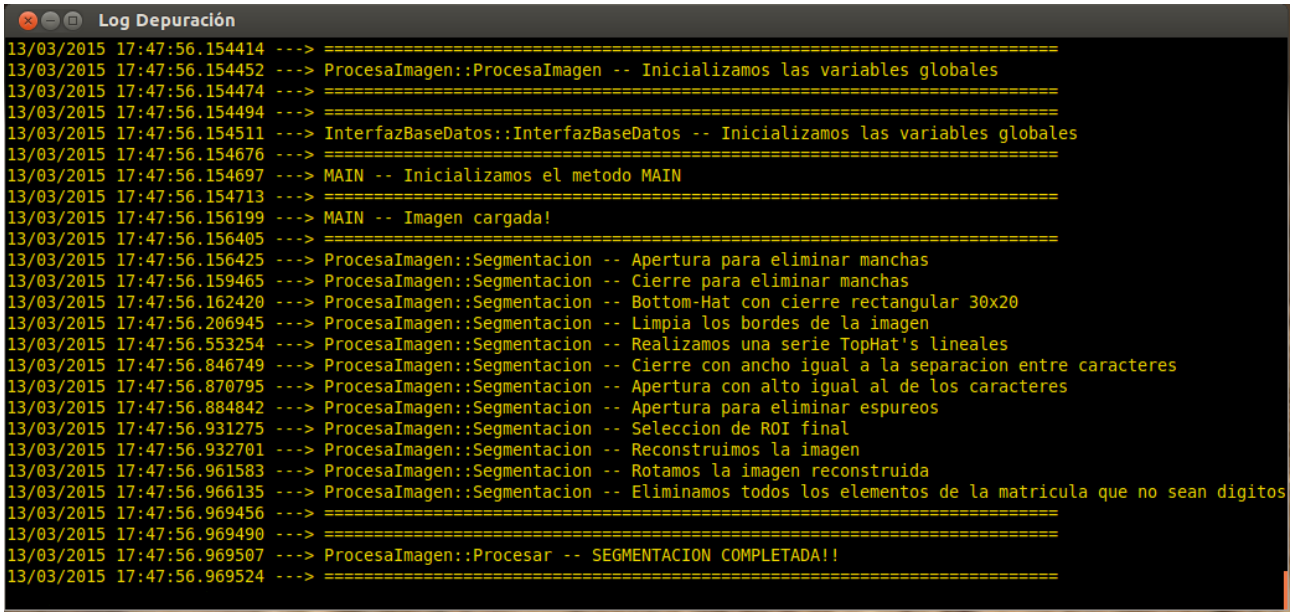


Figura 46: Depuración obtenida tras finalizar la segmentación

Del *log* mostrado se pueden sacar varias conclusiones importantes, teniendo siempre presente que se está realizando el ejemplo con el servidor y no con el dispositivo embebido. En primer lugar, cabe decir que el proceso de segmentación en su totalidad tarda aproximadamente **813 ms** en este caso. Este dato es válido y aceptable, dado que concentra la parte más pesada de la aplicación, además el tiempo no supone un parámetro limitante en éste tipo de aplicaciones.

Se puede apreciar también que las fases más costosas en tiempo son las encargadas de hacer una profunda reducción de la información del fondo de la imagen debido a la reiteración de operaciones morfológicas que, a su vez, evalúan y trabajan sobre zonas de la imagen no más grandes que los elementos estructurales con las que se realizan.

Una vez realizada la segmentación de la imagen, corresponde realizar el análisis e interpretación de la imagen obtenida.

Ya dentro de este proceso, es necesario invertir la imagen segmentada para facilitar la tarea del *OCR* y, tras eso, ejecutarlo para llevar a cabo el reconocimiento de los dígitos. La salida de éste consiste ya en una cadena de caracteres, que se puede ver en el siguiente fragmento del *log*:

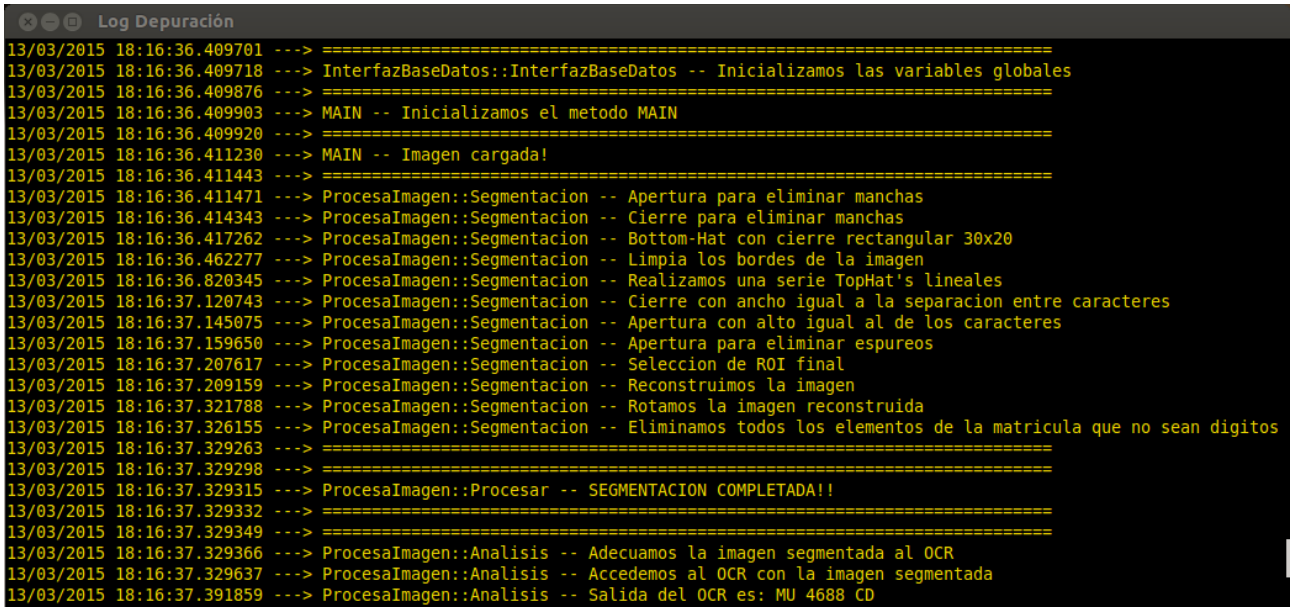


Figura 47: Depuración obtenida tras la actuación del OCR

Con el fin de eliminar los espacios y los caracteres que no se contemplen para este uso, se realiza una corrección sintáctica de la cadena obtenida, consiguiendo finalmente la cadena definitiva.

Por último, y con la doble intención de comprobar la validez y obtener información sobre la posible procedencia del vehículo, se analiza la codificación de la cadena obtenida a partir de la longitud de la misma, de la posición y tipo de carácter.

El *log* obtenido tras el proceso que se ha comentado es el siguiente:

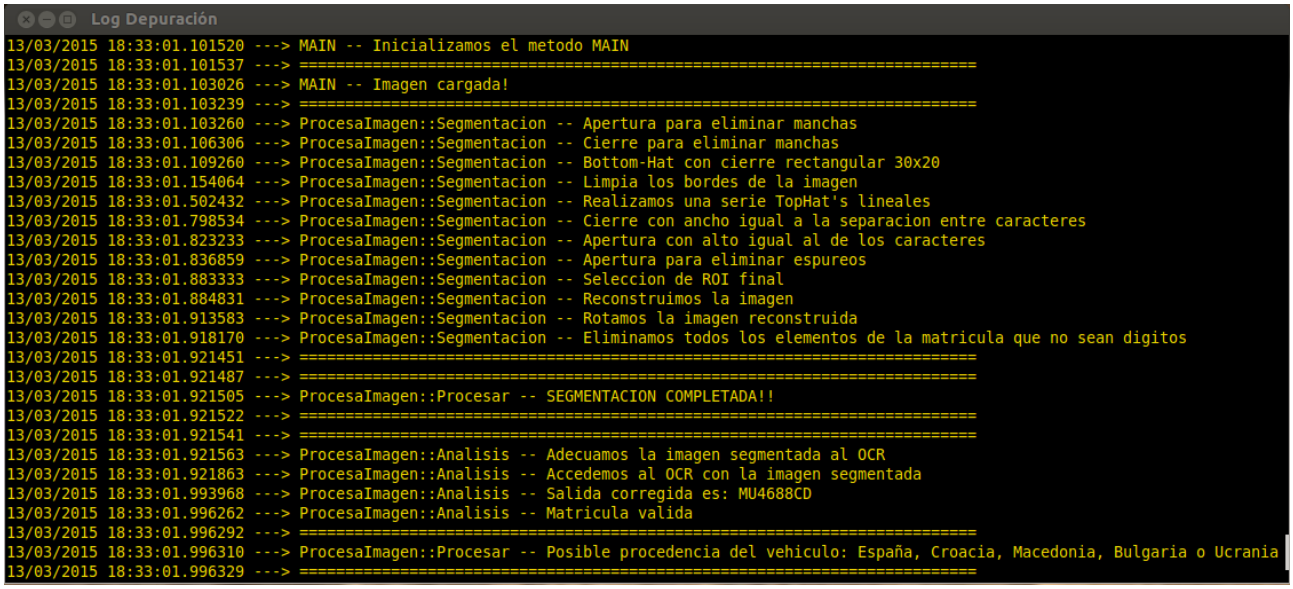


Figura 48: Depuración obtenida tras el corrector y validador sintáctico

Con estas operaciones se finaliza el proceso de análisis e interpretación y, tras almacenar adecuadamente tanto la cadena que representa la matrícula, la posible procedencia y la bandera que certifica que ésta es válida, se termina también el procesado del *frame*. Haciendo una medida de tiempo total empleado en el procesado, se obtienen **893 ms**, donde se puede ver que el análisis tiene una influencia casi anecdótica.

Una vez terminado el procesado de la imagen, entra en juego el objeto interfaz y la comunicación con la base de datos. Cabe ilustrar el contenido de la base de datos en este momento para tener una percepción del estado inicial de la misma. A continuación se muestra el contenido de cada una de las tres tablas que la componen:

ID	Matricula	Procedencia
001	2279FTN	España
002	A1584TF	España
003	2894FMN	España
004	28520MN	Luxemburgo
005	1324RM623	Francia
006	MU4688CD	España
007	4688FTX	España

Figura 49: Estado inicial de la tabla "Id_Matricula"

ID	Fecha_Esperada	Hora_Esperada	Fecha	Hora	Situacion
001	2015-01-16	12:30:00	2015-01-16	12:32:48	Servido
002	2015-01-20	17:00:00	2015-01-20	17:50:26	Servido
003	2015-06-08	10:30:00	*NULL*	*NULL*	Esperando
004	2015-06-08	10:30:00	2015-03-09	12:30:00	En_Base
005	2015-03-21	13:15:00	*NULL*	*NULL*	Esperando
006	2015-02-15	14:00:00	*NULL*	*NULL*	Esperando
007	2015-04-15	17:30:00	*NULL*	*NULL*	Esperando

Figura 50: Estado inicial de la tabla "Llegadas"

ID	Fecha_Esperada	Hora_Esperada	Fecha	Hora	Situacion
001	2015-01-16	18:00:00	2015-02-05	14:02:48	Servido
002	2015-01-20	23:30:00	2015-01-20	23:16:22	Servido
003	*NULL*	*NULL*	*NULL*	*NULL*	Esperando
004	2015-03-09	14:00:00	*NULL*	*NULL*	En_Base
005	*NULL*	*NULL*	*NULL*	*NULL*	Esperando
006	*NULL*	*NULL*	*NULL*	*NULL*	Esperando
007	*NULL*	*NULL*	*NULL*	*NULL*	Esperando

Figura 51: Estado inicial de la tabla "Salidas"

Las tablas anteriores representan, por orden: "*Id_Matricula*", "*Llegadas*" y "*Salidas*"; y se puede ver en ellas una implementación para el ejemplo que se está desarrollando. El paso siguiente de la ejecución del programa será la comprobación de la existencia de la matrícula obtenida, la comprobación de la procedencia y, si procede, el registro de la llegada del vehículo.

Dado que la matrícula obtenida posee la bandera de validez activada, se ejecutará la búsqueda. En el siguiente *log* se puede comprobar paso a paso el proceso de búsqueda y registro en la base de datos:

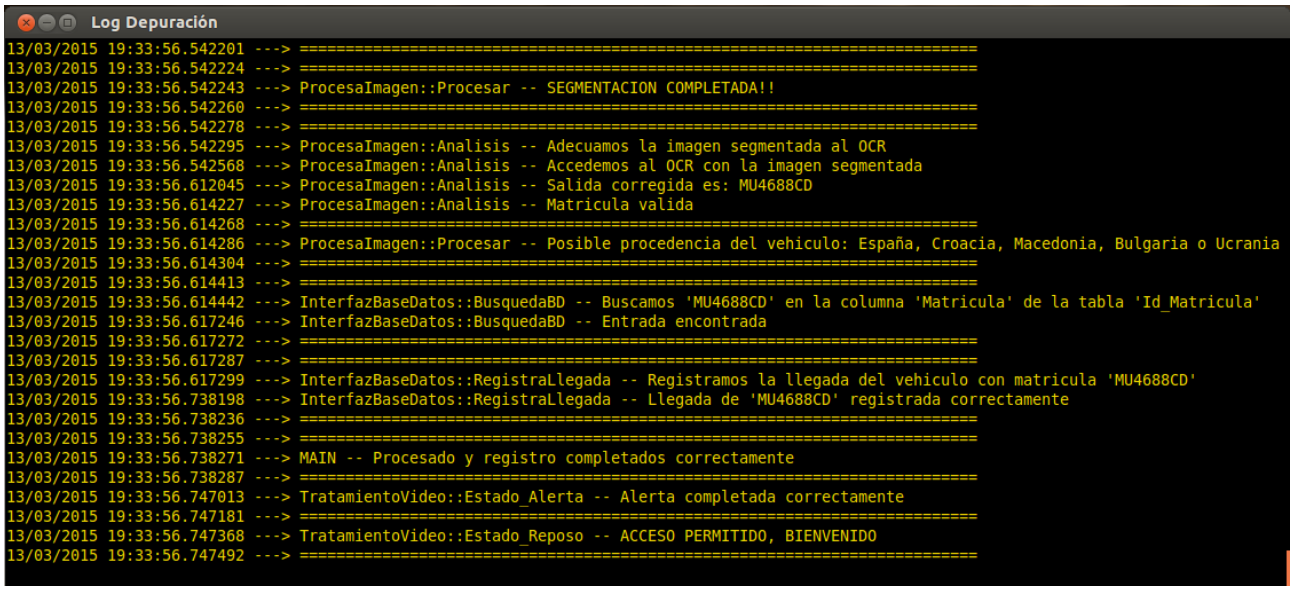


Figura 52: Depuración obtenida tras la actuación sobre la base de datos

Se puede ver cómo, tras encontrar la matrícula dentro de la base de datos (con la *ID* número 6), se comprueba la procedencia de la misma entre las sugeridas por la codificación y, al obtener un resultado positivo, se lleva a cabo el registro de la llegada del vehículo, dando como resultado las siguientes modificaciones en las tablas "*Llegadas*" y "*Salidas*":

ID	Fecha_Esperada	Hora_Esperada	Fecha	Hora	Situacion
001	2015-01-16	12:30:00	2015-01-16	12:32:48	Servido
002	2015-01-20	17:00:00	2015-01-20	17:50:26	Servido
003	2015-06-08	10:30:00	*NULL*	*NULL*	Esperando
004	2015-06-08	10:30:00	2015-03-09	12:30:00	En_Base
005	2015-03-21	13:15:00	*NULL*	*NULL*	Esperando
006	2015-02-15	14:00:00	2015-03-13	19:33:56	En_Base
007	2015-04-15	17:30:00	*NULL*	*NULL*	Esperando

Figura 53: Estado de la tabla "Llegadas" tras el registro de llegada

ID	Fecha_Esperada	Hora_Esperada	Fecha	Hora	Situacion
001	2015-01-16	18:00:00	2015-02-05	14:02:48	Servido
002	2015-01-20	23:30:00	2015-01-20	23:16:22	Servido
003	*NULL*	*NULL*	*NULL*	*NULL*	Esperando
004	2015-03-09	14:00:00	*NULL*	*NULL*	En_Base
005	*NULL*	*NULL*	*NULL*	*NULL*	Esperando
006	2015-03-13	22:03:56	*NULL*	*NULL*	En_Base
007	*NULL*	*NULL*	*NULL*	*NULL*	Esperando

Figura 54: Estado de la tabla "Salidas" tras el registro de llegada

Comparando con las tablas iniciales aportadas anteriormente, se puede ver como se han llevado a cabo las modificaciones de la entrada afectada en cada una de las tablas.

Habiendo realizado correctamente el registro finaliza la ejecución de *CProcesadorMorfológico*, el cual devuelve una respuesta positiva a *CSensorPresencia* con el fin de que éste ejecute las acciones oportunas, es decir, activación de la salida digital que actúa sobre la barrera (pin 7), congelación del proceso durante un periodo de 60 segundos y una segunda actuación sobre la barrera (pin 11) con la intención de bajarla. Tras eso, solo quedaría poner los registros del objeto *CSensorPresencia* a cero y comenzar a adquirir de nuevo.

5.3. Caso con ejecución abortada

Tal y como se ha ido indicando en diferentes puntos de la ejecución del programa *CProcesadorMorfologico* varios son los motivos que pueden provocar que ésta sea abortada: una imagen completamente en negro, que no exista una *ROI* que satisfaga las condiciones de tamaño y forma, que la matricula obtenida del *OCR* no tenga el tamaño o codificación valida, que no se encuentre o no se pueda registrar la entrada para esa matrícula en la base de datos, etc..

Todos esos motivos provocan, además de la mencionada finalización forzada, una respuesta negativa devuelta a *CSensorPresencia*. Es ahí donde se deciden las acciones a realizar frente a la respuesta obtenida del programa esclavo.

Tal y como se indicó en el diseño, existe un máximo de respuestas negativas que el programa es capaz de aceptar, ordenando sucesivos reintentos. Llegados a ese máximo, se supone que el procesador ha sido capaz de reconocer correctamente la matrícula y que el problema es provocado por la no existencia de la entrada en la base de datos o por estar tratando de modificar una llegada ya registrada. En ese momento, el programa emplaza al vehículo a solicitar acceso manual al recinto a través de la persona responsable de la seguridad y pausa el programa el tiempo estimado para que la entrada se despeje.

No entrando en los motivos con los que se han simulado las finalizaciones forzadas del programa esclavo, se puede mostrar el *log* que se obtiene en dicho caso:

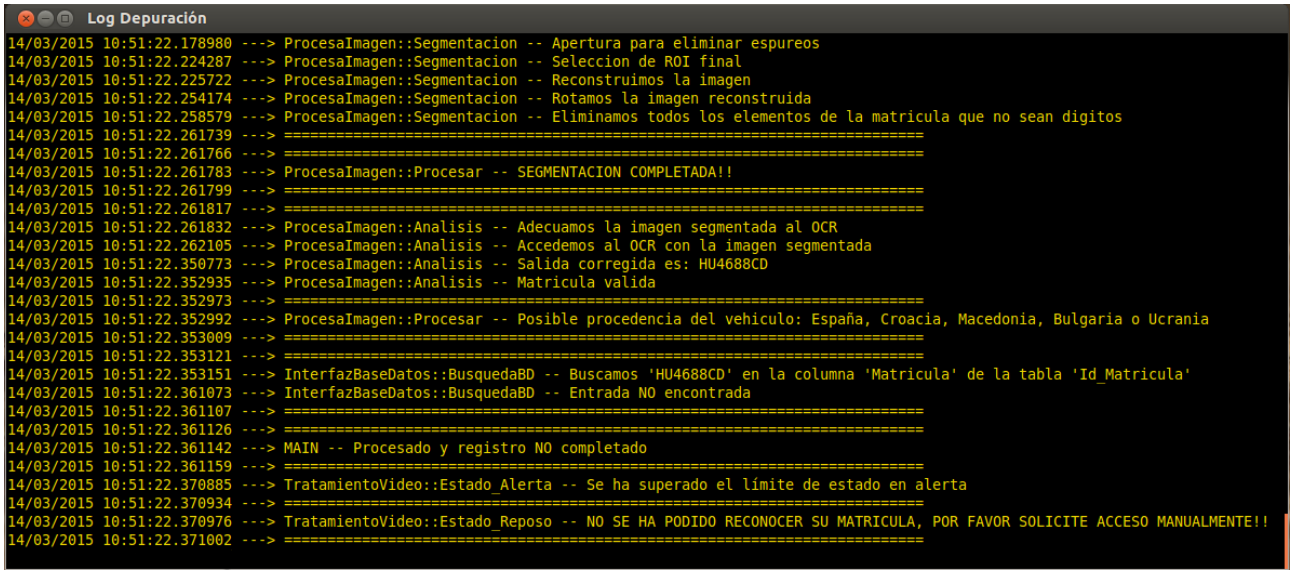


Figura 55: Depuración obtenida tras superar el límite de reintentos

En el *log* anterior se puede ver como el error que ha sobrepasado el límite de intentos ha sido no conseguir encontrar la matrícula en la base de datos por un mal reconocimiento del primer carácter por parte del *OCR*.

5.4. Resumen de los resultados

Como colofón al capítulo se pueden sacar las siguientes conclusiones del ejemplo realizado:

- El tiempo necesitado por el programa esclavo se encuentra dentro de unos márgenes aceptables, habiendo considerado éste como un parámetro no limitante. Con una media por debajo de un segundo para los casos en los que se ejecuta de forma completa.

- Gracias a la utilización de un sistema maestro-esclavo, el uso de memoria del dispositivo no experimenta crecimiento continuo y se mantiene constante en una cantidad asumible por el dispositivo.

- La implementación de una evaluación continua de resultados dentro del programa esclavo, permite que se aborte la ejecución del mismo en el momento que se tiene certeza de que no se va a poder alcanzar el objetivo, lo cual lo dota de una mayor eficiencia.

- En el ejemplo de ejecución ideal, se puede ver que el sistema ha sido capaz de analizar, encontrar y registrar correctamente la matrícula con el primer *frame* obtenido, lo que puede dar una idea de la notable eficacia del sistema.

6. Conclusiones

En el presente capítulo se exponen las conclusiones sobre el desarrollo, el sistema y las líneas futuras asociadas a la aplicación que se ha realizado. Desde el punto de vista del autor se podrán conocer datos como la duración del proyecto, las principales problemáticas encontradas durante el desarrollo y la visión que éste tiene sobre el futuro.

El desarrollo del proyecto ha precisado de una duración próxima a los seis meses desde su planteamiento inicial, pasando por la investigación, desarrollo de prototipo y simulaciones, etc. Se considera que se trata de un plazo razonable teniendo en cuenta que, además de todo lo aportado por los tutores, el grueso del desarrollo se ha realizado de forma autónoma.

Respecto a los problemas encontrados durante el desarrollo del mismo, dejando a un lado las dificultades circunstanciales, cabe destacar la constante necesidad de explotar al máximo las capacidades de dispositivos de bajo coste. Esto incluye desde la optimización software de las imágenes adquiridas hasta una cuidada implementación del algoritmo de procesado desde el punto de vista del uso de memoria, pasando por los ajustes necesarios para maximizar, en la medida de lo posible, los parámetros de eficiencia y eficacia que se fijaban como objetivos iniciales.

Se considera que, en gran medida, los objetivos fijados se cumplen de forma aceptable, como consecuencia de un alto aprovechamiento de los medios de los cuales se dispone. Esto ha permitido crear un sistema con una planta realista que acerque tanto a *pymes* como al usuario particular una tecnología práctica actualmente al alcance de pocos.

Además, la forma modular en la que se ha planteado la solución final posibilita, además de adecuar el sistema para las necesidades de cada cliente, las mejoras y crecimiento tanto del bloque encargado de la segmentación como del OCR y, en general, cualquier parte del mismo sin influir en el funcionamiento del resto.

La conjunción de los elementos que se han utilizado en el presente proyecto y, sobretodo, el bajo coste que se ha alcanzado, hace pensar en su aplicación directa para distintos fines enmarcados en el ámbito público e ideas tan actuales como son las *Smart Cities*.

Habiendo elegido los elementos que componen el sistema como cota mínima

para un funcionamiento adecuado del mismo, se puede pensar que cualquier cambio hacia elementos de mayores capacidades supondrá una mejora de la respuesta del sistema. Dando prioridad en éste sentido a la utilización de un mejor dispositivo de adquisición, con capacidad de obtención de imágenes infrarrojas para entornos poco iluminados.

Fijando ahora el objetivo en las posibles ampliaciones que éste sistema acepta, más allá del gran número de aplicaciones que puede tener, se visualiza dotar al mismo de una parte de *Front-End* encaminada de forma mixta hacia dispositivos móviles y fijos.

Tal y como se plantea la aplicación, se puede planificar una serie de aplicaciones embebidas sobre un servidor web, alojado también en la *R-Pi*, que permitan el acceso remoto en tiempo real a lo que se está adquiriendo a través de la cámara y el control de las operaciones del dispositivo.

Dicho sistema, junto con una aplicación de alarmas, puede avisar al responsable de seguridad en los momentos en los que la aplicación no sea capaz de permitir el acceso a un vehículo y, tras las comprobaciones oportunas, dar acceso y registro al vehículo independientemente del lugar donde se encuentre el responsable, reactivando la aplicación en el momento que éste considere.

Es sencillo vislumbrar esta posibilidad, asumiendo que el dispositivo móvil se encuentre conectado a la misma red que cualquier otro dispositivo fijo y, por supuesto, el/los dispositivo/os embebido/os. Mediante scripts de actuación similares a los usados para la interfaz de la base de datos, se podrían realizar las operaciones que se deseen sobre él, desde cualquier punto del recinto.

7. Anexos

En este apartado se incluyen los anexos a los que se ha hecho referencia a lo largo del presente documento. Se pretende, además, complementar las tecnologías y herramientas utilizadas en el proyecto, de forma ilustrativa para la mejor compresión del mismo.

7.1. Anexo I: Funciones utilizadas en las clases de procesado de imagen

Dentro de este anexo se incluyen las funciones utilizadas en las clases que se encargan puramente de la parte de procesado de imagen del sistema. Cabe separar dichas funciones en tres grupos, que serían:

- Funciones nativas de OpenCV
- Funciones implementadas a partir de nativas de OpenCV
- Funciones de Tesseract-OCR

Una vez realizada esta distinción se procede a enumerar y explicar las funciones que componen cada grupo.

7.1.1. Funciones nativas de OpenCV

cvCreateImage

Crea la cabecera de una imagen, asigna memoria para los datos de la misma y devuelve el puntero hacia ella.

```
IplImage* cvCreateImage(CvSize size, int depth, int channels)
```

Parámetros:

- **`size`** – Ancho y alto de la imagen.
- **`depth`** – Resolución en bits de los elementos de la imagen.

- **channels** – Número de canales por píxel.

cvCreateMat

Crea la cabecera de una matriz, asigna memoria para los datos de la misma y devuelve el puntero hacia ella.

```
CvMat* cvCreateMat(int rows, int cols, int type)
```

Parámetros:

- **rows** – Numero de filas de la imagen.
- **cols** – Numero de columnas de la imagen.
- **type** – Tipo de los elementos de la matriz expresado mediante:
  CV_<resolución de bit><S|U|F>C<Número de canales>,
  dónde: S = con signo, U = sin signo y F = flotante.

cvLine

Dibuja un segmento lineal que conecta dos puntos.

```
void cvLine(CvArr* img,
            CvPoint pt1,
            CvPoint pt2,
            CvScalar color,
            int thickness = 1,
            int line_type = 8,
            int shift = 0)
```

Parámetros:

- **img** – Imagen donde se dibujará el segmento.
- **pt1** – Primer punto del segmento lineal.
- **pt2** – Segundo punto del segmento lineal.
- **color** – Color de la linea.
- **thickness** – Grosor de la linea.
- **line_type** – Tipo de linea:
  - 8 – Linea 8-conectada.
  - 4 – Linea 4-conectada.
  - CV_AA – Linea *antialiased*.
- **shift** – Numero de bits en los puntos coordenada.

cvCreateStructuringElementEx

Devuelve un elemento estructurante del tamaño y forma especificado para operaciones morfológicas.

```
IplConvKernel* cvCreateStructuringElementEx(int cols,
                                            int rows,
                                            int anchor_x,
                                            int anchor_y,
                                            int shape,
                                            int* values = NULL)
```

Parámetros:

- **cols** – Ancho del elemento estructurante.
- **rows** – Altura del elemento estructurante.
- **anchor_x** – Coordenada *x* del punto de anclaje del elemento.
- **anchor_y** – Coordenada *y* del punto de anclaje del elemento.
- **shape** – Forma de elemento estructurante:
  - CV_SHAPE_RECT – Elemento estructurante rectangular.
  - CV_ SHAPE_ELLIPSE – Elemento estructurante eliptico.
  - CV_SHAPE_CROSS – Elemento estructurante en cruz.
  - CV_SHAPE_CUSTOM – Elemento estructurante personalizado.
- **values** – Array de enteros de (**cols** * **row**) elementos que especifica la forma personalizada del elemento estructurante, cuando **shape** = CV_SHAPE_CUSTOM.

cvCreateMemStorage

Crea un objeto de almacenamiento de memoria y devuelve un puntero asociado al mismo.

```
CvMemStorage* cvCreateMemStorage(int block_size = 0)
```

Parámetros:

- **block_size** – Tamaño de los bloques de almacenamiento en *bytes*. Si su valor es 0, el bloque de almacenamiento utiliza el valor por defecto, 64K.

cvReleaseImage

Libera la memoria utilizada por una imagen, tanto por la cabecera como por los datos de la misma.

```
void cvReleaseImage(IplImage** image)
```

Parámetros:

- **image** – Doble puntero dirigido a la cabecera de la imagen.

cvReleaseMat

Libera la memoria utilizada por una matriz, tanto por la cabecera como por los datos de la misma.

```
void cvReleaseMat(CvMat** mat)
```

Parámetros:

- **mat** – Doble puntero dirigido a la matriz.

cvReleaseStructuringElement

Libera la memoria utilizada por un elemento estructurante, tanto por la cabecera como por los datos del mismo.

```
void cvReleaseStructuringElement(IplConvKernel** strel)
```

Parámetros:

- **strel** – Doble puntero dirigido al elemento estructurante.

cvReleaseMemStorage

Libera la memoria utilizada por un objeto de almacenamiento de memoria, tanto por la cabecera como por los datos del mismo.

```
void cvReleaseMemStorage(CvMemStorage** memstorage)
```

Parámetros:

- **memstorage** – Doble puntero dirigido al objeto de almacenamiento de memoria.

cvLoadImage

Carga una imagen desde archivo y devuelve el puntero asociado a la misma.

```
IplImage* cvLoadImage(const char* filename,
                      int iscolor = CV_LOAD_IMAGE_COLOR)
```

Parámetros:

- **filename** – Ruta del archivo que va a ser cargado.
- **iscolor** – Especifica el tipo de la imagen cargada:
  - CV_LOAD_IMAGE_ANYDEPTH
  - CV_LOAD_IMAGE_COLOR
  - CV_LOAD_IMAGE_GRAYSCALE

cvSaveImage

Guarda una imagen en un archivo concreto y devuelve 0 en caso de éxito o -1 en caso contrario.

```
int cvSaveImage(const char* filename,
                const CvArr* image,
                const int params = 0)
```

Parámetros:

- **filename** – Ruta del archivo
- **image** – Imagen que va a ser guardada
- **params** – Opciones del formato en la imagen de destino:
  - CV_IMWRITE_JPEG_QUALITY – De 0 a 100.
  - CV_IMWRITE_PNG_COMPRESSION – De 0 a 9.
  - CV_IMWRITE_PXM_BINARY – 0 o 1.

cvGetReal2D

Obtiene y devuelve el valor de un elemento especifico de una distribución bidimensional.

```
double cvGetReal2D(const CvArr* arr, int idx0, int idx1)
```

Parámetros:

- **arr** – Array de entrada. Debe ser monocanal.
- **idx0** – Primera componente, basada en cero, del índice.
- **idx1** – Segunda componente, basada en cero, del índice.

cvSetReal2D

Cambia el valor de un elemento específico de una distribución bidimensional.

```
void cvSetReal2D(const CvArr* arr,
                 int idx0,
                 int idx1,
                 double value)
```

Parámetros:

- **arr** – Array de entrada. Debe ser monocanal.
- **idx0** – Primera componente, basada en cero, del índice.
- **idx1** – Segunda componente, basada en cero, del índice.
- **value** – Valor que se desea asignar.

cvCloneImage

Realiza una copia completa de una imagen, incluyendo cabecera, datos y ROI, y devuelve el puntero asociado a la nueva copia.

```
IplImage* cvCloneImage(const IplImage* image)
```

Parámetros:

- **image** – Imagen original.

cvConvertImage

Convierte una imagen en otra y permite realizar un giro vertical simultaneamente.

```
void cvConvertImage(const CvArr* src, CvArr* dst, int flags = 0)
```

Parámetros:

- **src** – Imagen fuente.
- **dst** – Imagen destino.
- **flags** – Activa el giro vertical.

cvNot

Invierte el valor de todos los valores de un array.

```
void cvNot(const CvArr* src, CvArr* dst)
```

Parámetros:

- **src** – Imagen fuente.
- **dst** – Imagen destino.

cvCmp

Realiza una comparación elemento a elemento de dos arrays.

```
void cvCmp(const CvArr* src1,
           const CvArr* src2,
           CvArr* dst,
           int cmp_op)
```

Parámetros:

- **src1** – Primera imagen fuente. Debe ser monocanal.
- **src2** – Segunda imagen fuente. Debe ser monocanal.

- **dst** – Imagen destino, de igual tamaño y tipo que las de entrada.
- **cmpop** – operador que especifica la comparación entre las imágenes:
  - CMP_EQ – **src1** es igual a **src2**.
  - CMP_GT – **src1** es mayor que **src2**.
  - CMP_GE – **src1** es mayor o igual a **src2**.
  - CMP_LT – **src1** es menor que **src2**.
  - CMP_LE – **src1** es menor o igual que **src2**.
  - CMP_NE – src1 es desigual a src2.

cvMax

Calcula el máximo elemento a elemento de dos arrays.

```
void cvMax(const CvArr* src1, const CvArr* src2, CvArr* dst)
```

Parámetros:

- **src1** – Primera imagen fuente.
- **src2** – Segunda imagen fuente.
- **dst** – Imagen destino, de igual tamaño y tipo que src1.

cvMin

Calcula el mínimo elemento a elemento de dos arrays.

```
void cvMin(const CvArr* src1, const CvArr* src2, CvArr* dst)
```

Parámetros:

- **src1** – Primera imagen fuente.
- **src2** – Segunda imagen fuente.
- **dst** – Imagen destino, de igual tamaño y tipo que **src1**.

cvMul

Calcula el producto escalado elemento a elemento de dos arrays.

```
void cvMul(const CvArr* src1,
           const CvArr* src2,
           CvArr* dst,
           double scale = 1)
```

Parámetros:

- **src1** – Primera imagen fuente.
- **src2** – Segunda imagen fuente, de igual tamaño y tipo que **src1**.

- **dst** – Imagen destino, de igual tamaño y tipo que **src1**.
- **scale** – Factor de escalado opcional.

cvSum

Calcula la suma de los elementos de un array y devuelve el valor en un vector de cuatro elementos, uno por cada canal y otro por el canal alfa.

```
CvScalar cvSum(const CvArr* arr)
```

Parámetros:

- **arr** – Array de entrada. Debe tener de 1 a 4 canales.

cvSub

Calcula la resta elemento a elemento de dos arrays.

```
void cvSub(const CvArr* src1,
           const CvArr* src2,
           CvArr* dst,
           const CvArr* mask = NULL)
```

Parámetros:

- **src1** – Primera imagen de entrada.
- **src2** – Segunda imagen de entrada.
- **dst** – Imagen de salida de igual tamaño y tipo que **src1**.
- **mask** – Operación con máscara opcional.

cvDilate

Realiza la dilatación morfológica de una imagen mediante un elemento estructurante específico.

```
void cvDilate(const CvArr* src,
              CvArr* dst,
              IplConvKernel* element = NULL,
              int iterations = 1)
```

Parámetros:

- **src** – Imagen de entrada.
- **dst** – Imagen de salida, de igual tamaño que **src**.
- **element** – Elemento estructurante utilizado en la dilatación.
- **iterations** – Número de veces que se aplica la dilatación.

cvErode

Realiza la erosión morfológica de una imagen mediante un elemento estructurante específico.

```
void cvErode(const CvArr* src,
             CvArr* dst,
             IplConvKernel* element = NULL,
             int iterations = 1)
```

Parámetros:

- **src** – Imagen de entrada.
- **dst** – Imagen de salida, de igual tamaño que **src**.
- **element** – Elemento estructurante utilizado en la erosión.
- **iterations** – Número de veces que se aplica la erosión.

cvMorphologyEx

Realiza la operación morfológica especificada sobre una imagen mediante un elemento estructurante específico.

```
void cvMorphologyEx(const CvArr* src,
                    CvArr* dst,
                    IplConvKernel* element,
                    int operation,
                    int iterations = 1)
```

Parámetros:

- **src** – Imagen de entrada.
- **dst** – Imagen de salida, de igual tamaño que **src**.
- **element** – Elemento estructurante utilizado en la operación.
- **operation** – Tipo de operación a realizar:
  - CV_MOP_OPEN – Apertura morfológica.
  - CV_MOP_CLOSE – Cierre morfológico.
  - CV_MOP_GRADIENT – Gradiente morfológico.
  - CV_MOP_TOPHAT – Operación *Top-Hat*.
  - CV_MOP_BLACKHAT – Operación *Bottom-hat*.
- **iterations** – Número de veces que se aplica la operación.

cv2DRotationMatrix

Calcula una matriz de transformación para una rotación 2D.

```
CvMat* cv2DRotationMatrix(CvPoint2D32f center,
                          double angle,
```

```
                                    double scale,
                                    CvMat* map_matrix)
```

Parámetros:

- **center** – Centro de la rotación en la imagen fuente.
- **angle** – Ángulo de rotación en grados.
- **scale** – Factor de escala isotrópico.
- **map_matrix** – Mapa de salida de la transformación.

cvRemap

Aplica una transformación geométrica genérica a una imagen.

```
            void cvRemap(const CvArr* src,
                         CvArr* dst,
                         const CvArr* mapx,
                         const CvArr* mapy,
int flags = CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS,
                        CvScalar fillval = cvScalarAll(0))
```

Parámetros:

- **src** – Imagen fuente.
- **dst** – Imagen destino.
- **mapx** – Mapa con los valores de *x*.
- **mapy** – Mapa con los valores de *y*.
- **flags** – *Flags* que indican tipo de interpolación y el tratamiento de los bordes.
- **fillval** – Valor de llenado.

cvWarpAffine

Aplica una transformación afín a una imagen.

```
          void cvWarpAffine(const CvArr* src,
                            CvArr* dst,
                            const CvMat* map_matrix,
int flags = CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS,
                          CvScalar fillval = cvScalarAll(0))
```

Parámetros:

- **src** – Imagen fuente.
- **dst** – Imagen destino del mismo tamaño y tipo que **src**.
- **map_matrix** – Matriz de transformación
- **flags** – *Flags* que indican tipo de interpolación y el tratamiento de los bordes.

- **fillval** – Valor de llenado.

cvInitUndistortMap

Computa el mapa para la compensación de la distorsión de la cámara de adquisición.

```
void cvInitUndistortMap(const CvArr* camera_matrix,
                        const CvMat* dist_coeffs,
                        CvArr* mapx,
                        CvArr* mapy)
```

Parámetros:

- **camera_matrix** – Matriz de entrada de la camara.
- **dist_coeffs** – Vector de coeficientes de distorsión.
- **mapx** – Mapa de salida con los valores de x.
- **mapy** – Mapa de salida con los valores de y.

cvThreshold

Aplica un umbral constante a cada elemento de un array.

```
double cvThreshold(const CvArr* src,
                   CvArr* dst,
                   double threshold,
                   double max_value,
                   int threshold_type)
```

Parámetros:

- **src** – Imagen de entrada, debe ser monocanal.
- **dst** – Imagen de salida, de igual tamaño y tipo que **src**.
- **threshold** – Valor umbral.
- **max_value** – Máximo valor para un umbralizado binario.
- **threshold_type** – Tipo de umbralizado:
  - CV_THRESH_BINARY – Umbralizado binario.
  - CV_THRESH_BINARY_INV – Umbralizado binario inverso.
  - CV_THRESH_TRUNC – Umbralizado truncado.
  - CV_THRESH_TOZERO – Umbralizado a cero.
  - CV_THRESH_TOZERO_INV – Umbralizado a cero inverso.
  - CV_THRESH_OTSU – Umbralizado por el método Otsu.

cvFindContours

Encuentra y etiqueta los contornos de una imagen binaria.

```
int cvFindContours(CvArr* image,
                   CvMemStorage* storage,
                    CvSeq** first_contour,
                int header_size = sizeof(CvContour),
                    int mode = CV_RETR_LIST,
                int method = CV_CHAIN_APPROX_SIMPLE,
                    CvPoint offset = cvPoint(0,0))
```

Parámetros:

- **image** – Imagen de entrada, debe ser monocanal y de 8 bits.
- **storage** – Objeto de almacenamiento de memoria donde se guardan los contornos obtenidos.
- **first_contour** – Puntero a la secuencia que señala los contornos.
- **header_size** – Tamaño del contorno.
- **mode** – Modo de recuperación de contornos:
  - CV_RETR_EXTERNAL – Devuelve únicamente los contornos externos.
  - CV_RETR_LIST – Devuelve todos los contornos sin establecer ninguna relación jerárquica.
  - CV_RETR_CCOMP – Devuelve todos los contornos organizados en una jerarquía de dos niveles.
  - CV_RETR_LIST – Devuelve todos los contornos y reconstruye la jerarquía completa de los contornos anidados.
- **method** – Método de aproximación de los contornos:
  - CV_CHAIN_APPROX_NONE – Almacena absolutamente todos los puntos de los contornos.
  - CV_CHAIN_APPROX_SIMPLE – Realiza una compresión horizontal, vertical y segmentos diagonales y aporta solo sus puntos finales.
  - CV_CHAIN_APPROX_TC89_L1 – Aplica uno de los tipos del algoritmo de aproximación de Teh-Chin.
- **offset** – Offset opcional para que todos los puntos sean desplazados.

cvDrawContours

Dibuja los contornos perimetrales o rellenos.

```
int cvDrawContours(CvArr* img,
                    CvSeq* contour,
                   CvScalar externalColor,
                   CvScalar holeColor,
```

```
                                    int maxLevel,
                                    int thickness = 1,
                                    int lineType = 8)
```

Parámetros:

- **img** – Imagen destino.
- **contour** – Todos los contornos de entrada.
- **externalColor** – Color usado para el perímetro externo de los contornos.
- **holeColor** – Color usado para el interior de los contornos.
- **maxLevel** – Máximo nivel para los contornos dibujados.
- **Thickness** – Grosor de las líneas con las que se dibujan los contornos.
- **lineType** – Conectividad de las líneas utilizadas.

cvContourArea

Calcula el área de un contorno.

```
double cvContourArea(const CvArr* contour,
                     CvSlice slice = CV_WHOLE_SEQ,
                    int oriented = 0)
```

Parámetros:

- **contour** – Vector de puntos bidimensional de entrada.
- **slice** – Modo de separación de la imagen que posee los contornos.
- **oriented** – Orientación del área calculada, ya sea horizontal o vertical.

cvMinAreaRect2

Encuentra el mínimo rectángulo rotado que encierra el grupo de puntos aportados a la entrada.

```
CvBox2D cvMinAreaRect2(const CvArr* points,
                       CvMemStorage* storage = NULL)
```

Parámetros:

- **points** – Vector de entrada formado por puntos bidimensionales.
- **storage** – Objeto de almacenamiento opcional para los datos obtenidos.

cvNamedWindow

Crea una ventana.

```
int cvNamedWindow(const char* name,
```

```
                                    int flags = CV_WINDOW_AUTOSIZE)
```

Parámetros:

- **name** – Nombre de la ventana que será utilizado como identificador de la misma.
- **flags** – *Flags* de la ventana:
  - CV_WINDOW_NORMAL – Permite redimensionar la ventana.
  - CV_WINDOW_AUTOSIZE – La ventana se ajusta automáticamente a la imagen mostrada.
  - CV_WINDOW_OPENGL - La ventana se crea con soporte *OpenGL*.

cvShowImage

Muestra una imagen en una ventana concreta.

```
void cvShowImage(const char* name,
                 const CvArr* image)
```

Parámetros:

- **name** – Nombre de la ventana que será utilizado como contenedor de la imagen.
- **image** – Imagen que será mostrada.

cvDestroyAllWindows

Destruye todas las ventanas HighGUI.

```
void cvDestroyAllWindow()
```

Parámetros:

- Sin parámetros.

cvWaitKey

Muestra una imagen en una ventana concreta.

```
void cvWaitKey(int delay = 0)
```

Parámetros:

- **delay** – Retardo en milisegundos. 0 es un valor especial que representa infinito.

cvCreateCameraCapture

Crea un objeto capturador de vídeo a partir de una cámara.

```
CvCapture* cvCreateCameraCapture(int index)
```

Parámetros:

- **index** – Índice denominador de la cámara.

cvCreateFileCapture

Crea un objeto capturador de vídeo a partir de un archivo.

```
CvCapture* cvCreateFileCapture(const char* filename)
```

Parámetros:

- **filename** – Ruta del archivo de vídeo.

cvReleaseCapture

Cierra un archivo de vídeo o un dispositivo de adquisición.

```
void cvReleaseCapture(CvCapture** capture)
```

Parámetros:

- **capture** – Puntero que señala al objeto capturador.

cvGrabFrame

Obtiene el siguiente *frame* del archivo de video o del dispositivo capturador.

```
int cvGrabFrame(CvCapture* capture)
```

Parámetros:

- **capture** – Objeto capturador de vídeo.

cvRetrieveFrame

Decodifica y devuelve un puntero al *frame* de vídeo obtenido con *cvGrabFrame*.

```
IplImage* cvRetrieveFrame(CvCapture* capture)
```

Parámetros:

- **capture** – Objeto capturador de vídeo.

cvQueryFrame

Combina las funciones de cvGrabFrame y cvRetrieveFrame en una sola.

```
IplImage* cvQueryFrame(CvCapture* capture)
```

Parámetros:

- **capture** – Objeto capturador de vídeo.

cvGetCaptureProperty/cvSetCaptureProperty

Devuelve/Asigna una propiedad especifica del/al capturador de vídeo.

```
double cvGetCaptureProperty(CvCapture* capture, int property_id)

int cvSetCaptureProperty(CvCapture* capture,
                         int property_id,
                         double value)
```

Parámetros:

- **capture** – Objeto capturador de vídeo.
- **property_id** – Identificador de la propiedad:
  - CV_CAP_PROP_POS_MSEC – Posición temporal del vídeo o tiempo de adquisición del objeto capturador.
  - CV_CAP_PROP_POS_FRAMES – Índice del *frame* a ser capturado.
  - CV_CAP_PROP_POS_AVI_RATIO – Posición relativa del archivo de video: 0 – principio, 1 – final.
  - CV_CAP_PROP_FRAME_WIDTH – Ancho del *frame* en el flujo de video.
  - CV_CAP_PROP_FRAME_HEIGHT - Alto del *frame* en el flujo de video.
  - CV_CAP_PROP_FPS – Ratio de *frames* por segundo.
  - CV_CAP_PROP_FOURCC – Código del *codec* utilizado.
  - CV_CAP_PROP_FRAME_COUNT – Numero de *frames* en el archivo de video.
  - CV_CAP_PROP_FORMAT – Formato de los objetos devueltos por cvRetrieveFrame.
  - CV_CAP_PROP_MODE – Valor de *backend* especifico indicado por el actual modo de adquisición.
  - CV_CAP_PROP_BRIGHTNESS – Brillo de la imagen (solo para cámaras).
  - CV_CAP_PROP_CONTRAST - Contraste de la imagen (solo para cámaras).
  - CV_CAP_PROP_SATURATION - Saturación de la imagen (solo para

cámaras).
    - CV_CAP_PROP_HUE - HUE de la imagen (solo para cámaras).
    - CV_CAP_PROP_GAIN - Ganancia de la imagen (solo para cámaras).
    - CV_CAP_PROP_EXPOSURE - Obturación (solo para cámaras).
- **value** – Valor que se desea asignar a la propiedad.

cvFindChessboardCorners

Encuentra la posición de las esquinas internas del tablero de ajedrez.

```
int cvFindChessboardCorners(const void* image,
                            CvSize pattern_size,
                            CvPoint2D32f* corners,
                            int* corner_count = NULL,
                  int flags = CV_CALIB_CB_ADAPTATIVE_THRESH
                            + CV_CALIB_CB_NORMALIZE_IMAGE)
```

Parámetros:

- **image** – Imagen fuente del tablero de ajedrez.
- **pattern_size** – Número de esquinas del tablero.
- **corners** – Array de salida de las esquinas detectadas.
- **corner_count** - Cuenta de las esquinas detectadas.
- **flags** – Operaciones variadas:
    - CV_CALIB_CB_ADAPTATIVE_THRESH – Uso de umbralizado adaptativo para convertir la imagen a binaria.
    - CV_CALIB_CB_NORMALIZE_IMAGE – Normalización de la imagen gamma mediante la ecualización del histograma tras el umbralizado.
    - CV_CALIB_CB_FILTER_QUADS – Utilización de un criterio adicional para filtrar los falsos positivos.
    - CV_CALIB_CB_FAST_CHECK – Búsqueda rápida de equinas en el tablero con el fin de descartar imágenes no válidas.

cvFindCornerSubPix

Afina la localización de esquinas.

```
void cvFindCornerSubPix(const CvArr* image,
                        CvPoint2D32f* corners,
                        int* count,
                        CvSize win,
                        CvSize zero_zone,
                        CvTermCriteria criteria)
```

Parámetros:

- **image** – Imagen fuente.
- **corners** – Coordenadas iniciales de las esquinas y coordenadas afinadas provistas a la salida.
- **count** – Cuenta de las esquinas detectadas.
- **win** – Mitad de la longitud lateral de la ventana buscada.
- **zero_zone** – Mitad de la longitud de la región muerta en el medio de la zona de búsqueda.
- **criteria** – Criterio para la terminación del proceso iterativo de refinamiento en la localización de las esquinas.

cvDrawChessboardCorners

Dibuja las esquinas internas detectadas del tablero de ajedrez.

```
void cvDrawChessboardCorners(CvArr* image,
                             CvSize pattern_size,
                             CvPoint2D32f* corners,
                             int* count,
                             int pattern_was_found)
```

Parámetros:

- **image** – Imagen destino. Debe ser una imagen de 8 bits.
- **pattern**_size – Número de esquinas del tablero.
- **corners** – Array de las esquinas detectadas.
- **count** – Cuenta de las esquinas detectadas.
- **patter_was_found** – Parámetro que indica si se ha encontrado el tablero completo o no.

cvCalibrateCamera2

Encuentra los parámetros intrínsecos y extrínsecos de la cámara a partir de varias visualizaciones del patrón de calibración.

```
double cvCalibrateCamera2(const CvMat* object_points,
                          const CvMat* image_points,
                          const CvMat* point_counts,
                          CvSize image_size,
                          CvMat* camera_matrix,
                          Cv_Mat* distortion_coeffs,
                          Cv_Mat* rotation_vectors = NULL,
                         Cv_Mat* translation_vectors = NULL,
                          int flags = 0,
CvTermCriteria term_crit = cvTermCriteria(CV_TERMCRIT_ITER +
                                          CV_TERMCRIT_EPS,
                                          30,
                                          DBL_EPSILON))
```

Parámetros:

- **`object_points`** – Concatenación de los puntos hallados de las imágenes usadas como patrón.
- **`image_points`** – Concatenación de la proyección de los puntos de las imágenes usadas como patrón.
- **`point_counts`** – Vector de enteros que contiene el número de visualizaciones de los patrones de calibración.
- **`image_size`** – Tamaño de la imagen.
- camera_matrix – Matriz intrínseca de salida de la cámara. Tamaño 3x3 con valores flotantes.
- **`distortion_coeffs`** – Matriz de salida con los coeficientes de distorsión de la cámara.
- **`rotation_vectors`** – Vector de salida con los vectores de rotación estimados para cada vista usada como patrón.
- **`translation_vectors`** - Vector de salida con los vectores de translación estimados para cada vista usada como patrón.
- **`flags`** – Diferentes *flags* que pueden ser:
  - CV_CALIB_USE_INTRINSIC_GUESS – Utiliza una estimación de los valores intrínsecos de la cámara.
  - CV_CALIB_FIX_PRINCIPAL_POINT – Fija el punto principal durante toda la optimización.
  - CV_CALIB_FIX_ASPECT_RATIO – Fija el ratio de aspecto en la matriz intrínseca de la cámara durante toda la optimización.
  - CV_CALIB_ZERO_TANGENT_DIST – Los coeficientes de distorsión tangenciales se fijan en cero.
- **`term_crit`** – Criterio de terminación para el algoritmo iterativo de optimización.

7.1.2. Funciones implementadas a partir de nativas de OpenCV

MuestraImagen

Muestra la imagen introducida con el título especificado.

```
void MuestraImagen(IplImage* imagen, const char* titulo)
```

Parámetros:

- **`imagen`** – Imagen que se desea mostrar.
- **`titulo`** – Titulo de la ventana HighGUI en la que se va a mostrar.

Funciones nativas utilizadas:

- `cvNamedWindow`

- `cvShowImage`
- `cvWaitKey`

BinarizaImagen

Convierte una imagen monocanal en una imagen binaria mediante el método elegido y devuelve el puntero a la imagen resultado.

```
IplImage* BinarizaImagen(IplImage* imagen, int modo)
```

Parámetros:

- **`imagen`** – Imagen que se desea binarizar.
- **`modo`** – Método por el que se llevará a cabo la binarización:
  - CV_THRESH_OTSU – Binarización mediante el método de Otsu.
  - OTHER – Binarización mediante umbral fijo.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvThreshold`

ImagenUnos

Convierte una imagen binaria de valores (0, 2^n) en una imagen de valores (0, 1) y devuelve el puntero a la imagen resultado.

```
IplImage* ImagenUnos(IplImage* imagen)
```

Parámetros:

- **`imagen`** – Imagen que se desea convertir.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvGetReal2D`
- `cvSetReal2D`

LimpiaBordes

Elimina los elementos conectados a los bordes de una imagen de grises y devuelve el puntero a la imagen resultado.

```
IplImage* LimpiaBordes(IplImage* imagen)
```

Parámetros:

- **`imagen`** – Imagen de entrada.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvGetReal2D`
- `cvSetReal2D`
- `cvNot`
- `cvMul`
- `cvReleaseImage`

Funciones derivadas utilizadas:

- `BinarizaImagen`
- `Reconstruccion`
- `ImagenUnos`

LimpiaPlaca

Elimina todos los elementos de la imagen que no puedan ser considerados dígitos de la matrícula según los límites de área, respecto a la media, introducidos y devuelve el puntero a la imagen resultado.

```
IplImage* LimpiaPlaca(IplImage* imagen,
                      double coef_inferior,
                      double coef_superior)
```

Parámetros:

- **`imagen`** – Imagen que se desea limpiar.
- **`coef_inferior`** – Umbral inferior respecto a la media del área de los contornos considerados dígitos.
- **`coef_superior`** – Umbral superior respecto a la media del área de los contornos considerados dígitos.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvCreateMemStorage`
- `cvCloneImage`
- `cvFindContours`
- `cvContourArea`
- `cvDrawContours`
- `cvMin`
- `cvReleaseImage`
- `cvReleaseMemStorage`

RotaImagen

Rota la imagen con el fin corregir la posición relativa de la matrícula y devuelve el puntero a la imagen rotada.

```
IplImage* RotaImagen(IplImage* imagen_rotar,
                     float* angulo,
                     CvPoint2D32f* centro)
```

Parámetros:

- **imagen_rotar** – Imagen que se desea rotar.
- **angulo** – Ángulo en grados que se va a utilizar para rotar la imagen.
- **centro** – Punto de la imagen que servirá como eje de la rotación.

Funciones nativas utilizadas:

- cvCreateImage
- cvCreateMat
- cv2DRotationMatrix
- cvWarpAffine
- cvReleaseMat

ROIs

Selecciona las regiones que se consideran matrícula o de interés y discrimina el resto, devolviendo un puntero a la imagen que contiene solo la región valida y, a través de referencia, el punto central y el ángulo de la misma con respecto a la horizontal.

```
IplImage* ROIs(IplImage* imagen_rotar,
               float* angulo,
               CvPoint2D32f* centro)
```

Parámetros:

- **imagen_rotar** – Imagen de entrada. Debe de ser binaria.
- **angulo** – Ángulo en grados respecto a la horizontal de la región considerada matrícula. Creciente en sentido antihorario.
- **centro** – Punto central de la región considerada matrícula.

Funciones nativas utilizadas:

- cvCreateImage
- cvCreateMemStorage
- cvFindContours
- cvMinAreaRect2
- cvDrawContours
- cvReleaseMemStorage

Funciones derivadas utilizadas:

- EsMatricula

EsMatricula

Realiza la comprobación de las dimensiones y aspecto de la región y certifica si se ajusta a las esperadas devolviendo *true* en tal caso, o *false* en caso contrario.

```
bool EsMatricula(CvBox2D* rectangulo)
```

Parámetros:

- **rectangulo** – Rectángulo de área mínima que engloba a la región que se desea evaluar.

CorrectorMatricula

Corrector lingüístico que elimina todos los caracteres de la cadena que no sean dígitos o letras en mayúscula, incluido los espacios.

```
char* CorrectorMatricula(char* matricula)
```

Parámetros:

- **matricula** – Cadena de caracteres que se desea corregir.

ValidadorMatricula

Evalúa el número de caracteres de la matrícula y la encamina hacia la comprobación de su validez, según las codificaciones contempladas en los últimos estándares de la unión europea.

```
bool ValidadorMatricula(char* matricula, char* procedencia)
```

Parámetros:

- **matricula** – Matricula que se desea comprobar.
- **procedencia** – Posible procedencia del vehículo según la codificación de su matrícula.

Funciones derivadas utilizadas:

- `CuatroDigitos`
- `CincoDigitos`
- `SeisDigitos`
- `SieteDigitos`
- `OchoDigitos`
- `NueveDigitos`

EnmascaraMatricula

Convierte la cadena de caracteres aportada en una máscara para la posterior comprobación de su validez, según la codificación europea.

*char** EnmascaraMatricula(*char** **matricula**)

Parámetros:

- **matricula** – Cadena de caracteres que se desea enmascarar.

CuatroDigitos

Evalúa la validez de una matrícula de cuatro dígitos según la codificación europea y, en caso positivo, devuelve la posible procedencia de dicha matrícula.

bool CuatroDigitos(*char** **matricula**, *char** **procedencia**)

Parámetros:

- **matricula** – Matricula que se desea comprobar.
- **procedencia** – Posible procedencia del vehículo según la codificación de su matrícula.

Funciones derivadas utilizadas:

- EnmascaraMatricula

CincoDigitos

Evalúa la validez de una matrícula de cinco dígitos según la codificación europea y, en caso positivo, devuelve la posible procedencia de dicha matrícula.

bool CincoDigitos(*char** **matricula**, *char** **procedencia**)

Parámetros:

- **matricula** – Matricula que se desea comprobar.
- **procedencia** – Posible procedencia del vehículo según la codificación de su matrícula.

Funciones derivadas utilizadas:

- EnmascaraMatricula

SeisDigitos

Evalúa la validez de una matrícula de seis dígitos según la codificación europea y, en caso positivo, devuelve la posible procedencia de dicha matrícula.

bool SeisDigitos(*char** **matricula**, *char** **procedencia**)

Parámetros:

- **matricula** – Matricula que se desea comprobar.
- **procedencia** – Posible procedencia del vehículo según la codificación de su matrícula.

Funciones derivadas utilizadas:

- `EnmascaraMatricula`

SieteDigitos

Evalúa la validez de una matrícula de siete dígitos según la codificación europea y, en caso positivo, devuelve la posible procedencia de dicha matrícula.

```
bool SieteDigitos(char* matricula, char* procedencia)
```

Parámetros:

- **matricula** – Matricula que se desea comprobar.
- **procedencia** – Posible procedencia del vehículo según la codificación de su matrícula.

Funciones derivadas utilizadas:

- `EnmascaraMatricula`

OchoDigitos

Evalúa la validez de una matrícula de ocho dígitos según la codificación europea y, en caso positivo, devuelve la posible procedencia de dicha matrícula.

```
bool OchoDigitos(char* matricula, char* procedencia)
```

Parámetros:

- **matricula** – Matricula que se desea comprobar.
- **procedencia** – Posible procedencia del vehículo según la codificación de su matrícula.

Funciones derivadas utilizadas:

- `EnmascaraMatricula`

NueveDigitos

Evalúa la validez de una matrícula de nueve dígitos según la codificación europea y, en caso positivo, devuelve la posible procedencia de dicha matrícula.

```
bool NueveDigitos(char* matricula, char* procedencia)
```

Parámetros:

- **matricula** – Matricula que se desea comprobar.
- **procedencia** – Posible procedencia del vehículo según la codificación de su matrícula.

Funciones derivadas utilizadas:

- EnmascaraMatricula

EsIgual

Evalúa si dos imágenes binarias son iguales.

```
bool EsIgual(IplImage* imagen_1, IplImage* imagen_2)
```

Parámetros:

- **imagen_1** – Primera imagen de entrada.
- **imagen_2** – Segunda imagen de entrada.

Funciones nativas utilizadas:

- cvCreateMat
- cvCmp
- cvSum
- cvReleaseMat

Dilatacion

Realiza una dilatación morfológica de la imagen aportada a la entrada, mediante el elemento estructurante especificado.

```
IplImage* Dilatacion(IplImage* imagen,
                     int elstr_ancho,
                     int elstr_alto,
                     int anclaje_x,
                     int anclaje_y,
                     int elstr_tipo,
                     int* elstr_valores)
```

Parámetros:

- **imagen** – Imagen a la que se desea aplicar la dilatación.
- **elstr_ancho** – Ancho del elemento estructurante.
- **elstr_alto** – Alto del elemento estructurante.
- **anclaje_x** – Punto de anclaje *x* del elemento estructurante.
- **anclaje_y** – Punto de anclaje *y* del elemento estructurante.
- **elstr_tipo** – Tipo de elemento estructurante:
  - CV_SHAPE_RECT – Elemento estructurante rectangular.

    - CV_ SHAPE_ELLIPSE – Elemento estructurante elíptico.
    - CV_SHAPE_CROSS – Elemento estructurante en cruz.
    - CV_SHAPE_CUSTOM – Elemento estructurante customizado.
- **elstr_valores** – Array de enteros de (**elstr_ancho** * **elstr_alto**) elementos que especifica la forma customizada del elemento estructurante, cuando **elstr_tipo** = CV_SHAPE_CUSTOM.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvCreateStructuringElementEx`
- `cvDilate`
- `cvReleaseStructuringElement`

Erosion

Realiza una erosión morfológica de la imagen aportada a la entrada, mediante el elemento estructurante especificado.

```
IplImage* Erosion(IplImage* imagen,
                  int elstr_ancho,
                  int elstr_alto,
                  int anclaje_x,
                  int anclaje_y,
                  int elstr_tipo,
                  int* elstr_valores)
```

Parámetros:

- **imagen** – Imagen a la que se desea aplicar la erosión.
- **elstr_ancho** – Ancho del elemento estructurante.
- **elstr_alto** – Alto del elemento estructurante.
- **anclaje_x** – Punto de anclaje *x* del elemento estructurante.
- **anclaje_y** – Punto de anclaje *y* del elemento estructurante.
- **elstr_tipo** – Tipo de elemento estructurante:
    - CV_SHAPE_RECT – Elemento estructurante rectangular.
    - CV_ SHAPE_ELLIPSE – Elemento estructurante elíptico.
    - CV_SHAPE_CROSS – Elemento estructurante en cruz.
    - CV_SHAPE_CUSTOM – Elemento estructurante customizado.
- **elstr_valores** – Array de enteros de (**elstr_ancho** * **elstr_alto**) elementos que especifica la forma customizada del elemento estructurante, cuando **elstr_tipo** = CV_SHAPE_CUSTOM.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvCreateStructuringElementEx`

- `cvErode`
- `cvReleaseStructuringElement`

Apertura

Realiza una apertura morfológica de la imagen aportada a la entrada, mediante el elemento estructurante especificado.

```
IplImage* Apertura(IplImage* imagen,
                   int elstr_ancho,
                   int elstr_alto,
                   int anclaje_x,
                   int anclaje_y,
                   int elstr_tipo,
                   int* elstr_valores)
```

Parámetros:

- **`imagen`** – Imagen a la que se desea aplicar la apertura.
- **`elstr_ancho`** – Ancho del elemento estructurante.
- **`elstr_alto`** – Alto del elemento estructurante.
- **`anclaje_x`** – Punto de anclaje *x* del elemento estructurante.
- **`anclaje_y`** – Punto de anclaje *y* del elemento estructurante.
- **`elstr_tipo`** – Tipo de elemento estructurante:
  - CV_SHAPE_RECT – Elemento estructurante rectangular.
  - CV_ SHAPE_ELLIPSE – Elemento estructurante elíptico.
  - CV_SHAPE_CROSS – Elemento estructurante en cruz.
  - CV_SHAPE_CUSTOM – Elemento estructurante customizado.
- **`elstr_valores`** – Array de enteros de (**`elstr_ancho`** * **`elstr_alto`**) elementos que especifica la forma customizada del elemento estructurante, cuando **`elstr_tipo`** = CV_SHAPE_CUSTOM.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvCreateStructuringElementEx`
- `cvMorphologyEx`
- `cvReleaseStructuringElement`

Cierre

Realiza un cierre morfológico de la imagen aportada a la entrada, mediante el elemento estructurante especificado.

```
IplImage* Cierre(IplImage* imagen,
                 int elstr_ancho,
```

```
                                    int elstr_alto,
                                    int anclaje_x,
                                   int anclaje_y,
                                   int elstr_tipo,
                                   int* elstr_valores)
```

Parámetros:

- **imagen** – Imagen a la que se desea aplicar el cierre.
- **elstr_ancho** – Ancho del elemento estructurante.
- **elstr_alto** – Alto del elemento estructurante.
- **anclaje_x** – Punto de anclaje *x* del elemento estructurante.
- **anclaje_y** – Punto de anclaje *y* del elemento estructurante.
- **elstr_tipo** – Tipo de elemento estructurante:
  - CV_SHAPE_RECT – Elemento estructurante rectangular.
  - CV_ SHAPE_ELLIPSE – Elemento estructurante elíptico.
  - CV_SHAPE_CROSS – Elemento estructurante en cruz.
  - CV_SHAPE_CUSTOM – Elemento estructurante customizado.
- **elstr_valores** – Array de enteros de (**elstr_ancho** * **elstr_alto**) elementos que especifica la forma customizada del elemento estructurante, cuando **elstr_tipo** = CV_SHAPE_CUSTOM.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvCreateStructuringElementEx`
- `cvMorphologyEx`
- `cvReleaseStructuringElement`

TopHat

Realiza un *Top-Hat* morfológico de la imagen aportada a la entrada, mediante el elemento estructurante especificado.

```
IplImage* TopHat(IplImage* imagen,
                 int elstr_ancho,
                 int elstr_alto,
                 int anclaje_x,
                int anclaje_y,
                int elstr_tipo,
                int* elstr_valores)
```

Parámetros:

- **imagen** – Imagen a la que se desea aplicar el *Top-Hat*.
- **elstr_ancho** – Ancho del elemento estructurante.
- **elstr_alto** – Alto del elemento estructurante.
- **anclaje_x** – Punto de anclaje *x* del elemento estructurante.

- **anclaje_y** – Punto de anclaje *y* del elemento estructurante.
- **elstr_tipo** – Tipo de elemento estructurante:
  - CV_SHAPE_RECT – Elemento estructurante rectangular.
  - CV_ SHAPE_ELLIPSE – Elemento estructurante elíptico.
  - CV_SHAPE_CROSS – Elemento estructurante en cruz.
  - CV_SHAPE_CUSTOM – Elemento estructurante customizado.
- **elstr_valores** – Array de enteros de (**elstr_ancho** * **elstr_alto**) elementos que especifica la forma customizada del elemento estructurante, cuando **elstr_tipo** = CV_SHAPE_CUSTOM.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvCreateStructuringElementEx`
- `cvMorphologyEx`
- `cvReleaseStructuringElement`

BottomHat

Realiza un *Bottom-Hat* morfológico de la imagen aportada a la entrada, mediante el elemento estructurante especificado.

```
IplImage* BottomHat(IplImage* imagen,
                    int elstr_ancho,
                    int elstr_alto,
                    int anclaje_x,
                    int anclaje_y,
                    int elstr_tipo,
                    int* elstr_valores)
```

Parámetros:

- **imagen** – Imagen a la que se desea aplicar el *Bottom-Hat*.
- **elstr_ancho** – Ancho del elemento estructurante.
- **elstr_alto** – Alto del elemento estructurante.
- **anclaje_x** – Punto de anclaje *x* del elemento estructurante.
- **anclaje_y** – Punto de anclaje *y* del elemento estructurante.
- **elstr_tipo** – Tipo de elemento estructurante:
  - CV_SHAPE_RECT – Elemento estructurante rectangular.
  - CV_ SHAPE_ELLIPSE – Elemento estructurante elíptico.
  - CV_SHAPE_CROSS – Elemento estructurante en cruz.
  - CV_SHAPE_CUSTOM – Elemento estructurante customizado.
- **elstr_valores** – Array de enteros de (**elstr_ancho** * **elstr_alto**) elementos que especifica la forma customizada del elemento estructurante, cuando **elstr_tipo** = CV_SHAPE_CUSTOM.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvCreateStructuringElementEx`
- `cvMorphologyEx`
- `cvReleaseStructuringElement`

DilatacionGeo

Realiza un dilatación mediante morfología geodésica de la imagen aportada a la entrada, condicionada por la máscara indicada.

```
IplImage* DilatacionGeo(IplImage* imagen, IplImage* mascara)
```

Parámetros:

- **`imagen`** – Imagen a la que se desea aplicar la dilatación.
- **`mascara`** – Imagen que sirve de condicionante para la dilatación.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvDilate`
- `cvMin`
- `cvReleaseImage`

ErosionGeo

Realiza un erosión mediante morfología geodésica de la imagen aportada a la entrada, condicionada por la máscara indicada.

```
IplImage* ErosionGeo(IplImage* imagen, IplImage* mascara)
```

Parámetros:

- **`imagen`** – Imagen a la que se desea aplicar la erosión.
- **`mascara`** – Imagen que sirve de condicionante para la erosión.

Funciones nativas utilizadas:

- `cvCreateImage`
- `cvErode`
- `cvMax`
- `cvReleaseImage`

Reconstruccion

Realiza una reconstrucción mediante dilatación geodésica de la imagen aportada a la entrada a partir de la imagen marcadora que se indica.

*IplImage** Reconstruccion(*IplImage** **imagen**, *IplImage** **imagen_base**)

Parámetros:

- **imagen** – Imagen marcador a la que se desea aplicar la reconstrucción.
- **imagen_base** – Imagen máscara que sirve de base para la reconstrucción.

Funciones nativas utilizadas:

- cvCreateImage

Funciones derivadas utilizadas:

- DilatacionGeo
- BinarizaImagen
- EsIgual

ApperturasLineales

Realiza un serie de Top-Hat's mediante elementos estructurantes lineales con una variación angular de 30 grados entre ellos.

*IplImage** AperturasLineales(*IplImage** **imagen**)

Parámetros:

- **imagen** – Imagen a la que se desea aplicar las aperturas.

Funciones nativas utilizadas:

- cvCreateImage
- cvCreateMat
- cvLine
- cvMin
- cvReleaseImage
- cvReleaseMat

Funciones derivadas utilizadas:

- TopHat

CalibrarCAM

Realiza un proceso de autocalibración de la cámara a partir de los patrones previamente obtenidos y almacenados de un tablero de ajedrez.

void CalibrarCAM()

Parámetros:

- Sin parámetros

Funciones nativas utilizadas:

- `cvCreateMat`
- `cvLoadImage`
- `cvCreateImage`
- `cvFindChessboardCorners`
- `cvFindCornerSubPix`
- `cvDrawChessboardCorners`
- `cvCalibrateCamera2`
- `cvInitUndistortMap`
- `cvReleaseMat`

7.1.3. Funciones de Tesseract-OCR

Init

Inicializa y entrena el clasificador del OCR a partir del idioma y el archivo de datos de entrenamiento indicados.

```
int Init(const char* datapath, const char* language)
```

Parámetros:

- **`datapath`** – Ruta al archivo de datos de entrenamiento.
- **`language`** – idioma de los dígitos a analizar.

SetPageSegMode

Especifica el tipo de texto que se pretende reconocer con el OCR.

```
void SetPageSegMode(PagSegMode mode)
```

Parámetros:

- **`mode`** – Modo en el que se va a realizar el reconocimiento de la imagen.

SetImage

Carga la imagen sobre la que el OCR realizará el reconocimiento de dígitos.

```
void SetImage(const uchar* imagedata,
              int width,
              int height,
              int bytes_per_pixel,
```

```
int bytes_per_line)
```

Parámetros:

- **imagedata** – Valor de cada píxel de la imagen.
- **width** – Ancho de la imagen.
- **height** – Alto de la imagen.
- **bytes_per_pixel** – Numero de bytes que representa cada píxel de la imagen.
- **bytes_per_line** – Numero de bytes que representa cada linea de la imagen.

GetUTF8Text

El texto reconocido se devuelve como una cadena de caracteres con la codificación UTF8.

```
char* GetUTF8Text()
```

Parámetros:

- Sin parámetros.

Clear

Libera el resultado de reconocimiento y cualquier imagen que continúe alojada en memoria.

```
void Clear()
```

Parámetros:

- Sin parámetros.

End

Cierra completamente el objeto Tesseract y libera cualquier objeto que pueda continuar alojado en memoria.

```
void End()
```

Parámetros:

- Sin parámetros.

7.2. Anexo II: Funciones utilizadas para la comunicación con la BD

Continuando con la mecánica seguida en el anexo anterior, en éste se incluyen las funciones utilizadas en la clase que se encarga de la comunicación y tratamiento de datos obtenidos de la base de datos. Cabe separar dichas funciones en dos grupos, que serían:

- Funciones o scripts implementados en el cliente
  - Funciones nativas de cURL
  - Funciones implementadas a partir de las nativas de cURL
  - Funciones nativas de WiringPi
- Funciones o scripts implementados en el servidor
  - Funciones nativas SQL
  - Funciones nativas PHP
  - Scripts PHP implementados

Esta distinción lleva implícita una distinción de lenguaje, ya que las funciones alojadas en el cliente son implementadas en C (libcurl), mientras que para las alojadas en el servidor se han utilizado scripts PHP para ejecutar las órdenes SQL y el posterior tratamiento de los resultados provenientes de la base de datos.

7.2.1. Funciones implementadas en el cliente

Funciones nativas de cURL

curl_global_init

Activa e inicializa el medio que la librería necesita. Funciona como una extensión de la propia carga de la librería, con el fin de no cargar módulos que puedan no resultar necesarios.

```
CURLcode curl_global_init(long flags)
```

Parámetros:

- **`flags`** – Formado por un patrón de bits que indican a la librería qué características inicializar. Pueden ser:
  - CURL_GLOBAL_ALL – Inicializa todo lo posible.
  - CURL_GLOBAL_SSL – Inicializa *SSL*.
  - CURL_GLOBAL_WIN32 – Inicializa la librería de *sockets Win32*.

- CURL_GLOBAL_NOTHING – No inicializa nada extra.

curl_easy_init

Crea y devuelve el puntero a un objeto de tipo *CURL* que servirá de identificador de la comunicación "*easy*" y será utilizado como entrada en otras funciones de tipo "*easy*".

```
CURL* curl_easy_init()
```

Parámetros:

- Sin parámetros.

curl_easy_cleanup

Función opuesta a la inicialización de la comunicación "*easy*". Cierra todas las conexiones del identificador "*easy*" aportado que, llegado a este punto, debería haber cumplido todas sus funciones dentro de la implementación.

```
void curl_easy_cleanup(CURL* handle)
```

Parámetros:

- **handle** – Puntero al identificador de la sesión "*easy*".

curl_easy_setopt

Función utilizada para configurar el comportamiento de la librería y especificar la transferencia a realizar. Todas las opciones son modificadas mediante su código de opción, seguido de los parámetros adecuados.

```
CURLcode curl_easy_cleanup(CURL* handle,
                           CURLoption option,
                           parameters)
```

Parámetros:

- **handle** – Puntero al identificador de la sesión "*easy*".
- **option** – Código de opción representado por un entero de tipo *long*.
- **parameters** – Parámetro asociado a la opción que se desea modificar. Puede tratarse de un *long*, un *puntero a función*, un *puntero a objeto* o un *curl_off_t*, dependiendo de la opción.

curl_easy_perform

Función encargada de llevar a cabo la transferencia descrita por las opciones

aportadas. Por tanto, es importante haber configurado previamente la transferencia.

```
CURLcode curl_easy_perform(CURL* handle)
```

Parámetros:

- **handle** – Puntero al identificador de la sesión "*easy*".

curl_easy_strerror

Función que devuelve un *string* describiendo detalladamente el código de error *CURLcode* aportado como argumento de entrada.

```
const char* curl_easy_strerror(CURLcode errornum)
```

Parámetros:

- **errornum** – Código de error del que se precisa descripción.

Funciones implementadas a partir de las nativas de cURL

BusquedaBD

Función que ordena la ejecución del *script* de búsqueda, devolviendo 0 y una estructura con los datos de la fila que contiene el valor buscado, en caso positivo, o -1 en caso contrario.

```
char* BusquedaBD(char* tabla,
                 char* columna,
                 char* valor,
                 void* estructura)
```

Parámetros:

- **tabla** – Tabla de la base de datos sobre la que se quiere realizar la búsqueda.
- **columna** – Columna que debería poseer el valor buscado.
- **valor** – Valor objetivo.
- **estructura** – Estructura que será rellenada con los datos obtenidos de la tabla.

Funciones nativas utilizadas:

- `curl_easy_setopt`
- `curl_easy_perform`

InsertaEntradaBD

Función que ordena la ejecución del *script* de entrada, devolviendo 0 en caso de su correcta ejecución o -1 en caso contrario.

```
char* InsertaEntradaBD(void* estructura)
```

Parámetros:

- **estructura** – Estructura que posee los datos que se desean insertar en la base de datos.

Funciones nativas utilizadas:

- curl_easy_setopt
- curl_easy_perform
- curl_easy_strerror

ActualizaElementoBD

Función que ordena la ejecución del *script* de actualización, devolviendo 0 en caso de haber podido actualizar la celda deseada o -1 en caso contrario.

```
char* ActualizaElementoBD(char* tabla,
                          char* id,
                          char* columna,
                          char* valor)
```

Parámetros:

- **tabla** – Tabla de la base de datos sobre la que se quiere realizar la actualización.
- **id** – Fila en la que se encuentra la celda a modificar.
- **columna** – Columna en la que se encuentra la celda a modificar.
- **valor** – Valor a introducir.

Funciones nativas utilizadas:

- curl_easy_setopt
- curl_easy_perform
- curl_easy_strerror

EliminaEntradaBD

Función que ordena la ejecución del *script* de eliminación, devolviendo 0 en caso de su correcta ejecución o -1 en caso contrario.

```
char* EliminaEntradaBD(char* id)
```

Parámetros:

- **id** – Identificador de la fila que se desea eliminar.

Funciones nativas utilizadas:

- `curl_easy_setopt`
- `curl_easy_perform`
- `curl_easy_strerror`

RegistraLlegadaBD

Función que ordena la ejecución del *script* de registro de llegada, devolviendo 0 en caso de haber podido registrarla correctamente o -1 en caso contrario.

```
char* RegistraLlegadaBD(char* id, char* matricula)
```

Parámetros:

- **`id`** – Fila en la que se va a registrar la llegada.
- **`matricula`** – Matrícula del vehículo entrante.

Funciones nativas utilizadas:

- `curl_easy_setopt`
- `curl_easy_perform`
- `curl_easy_strerror`

RegistraSalidaBD

Función que ordena la ejecución del *script* de registro de salida, devolviendo 0 en caso de haber podido registrarla correctamente o -1 en caso contrario.

```
char* RegistraSalidaBD(char* id, char* matricula)
```

Parámetros:

- **`id`** – Fila en la que se va a registrar la salida.
- **`matricula`** – Matrícula del vehículo saliente.

Funciones nativas utilizadas:

- `curl_easy_setopt`
- `curl_easy_perform`
- `curl_easy_strerror`

Funciones nativas de WiringPi

wiringPiSetup

Inicializa el sistema y prepara el esquema de pines del dispositivo embebido

para su utilización.

```
int WiringPiSetup(void)
```

Parámetros:

- Sin parámetros.

pinMode

Otorga el carácter deseado (entrada, salida o pin PWM) al pin indicado. Solo el pin 1 puede ser PWM.

```
void pinMode(int pin, int mode)
```

Parámetros:

- **pin** - Número de pin al sobre el que se quiere actuar.
- **mode** - Carácter que se quiere aportar al pin seleccionado.

digitalWrite

Otorga el valor deseado (HIGH o LOW) al pin indicado. HIGH equivale al 1 lógico y LOW al 0 lógico.

```
void digitalWrite(int pin, int value)
```

Parámetros:

- **pin** - Número de pin al sobre el que se quiere actuar.
- **value** - Valor que se quiere aportar al pin seleccionado.

delay

Pausa la ejecución del programa por tanto tiempo como se indique con un máximo de 49 días.

```
void delay(unsigned int howLong)
```

Parámetros:

- **howLong** – Tiempo en milisegundos que se desea pausar la ejecución.

7.2.2. Scripts implementados en el servidor

Funciones nativas SQL

SELECT*

Recupera filas de la base de datos y habilita la selección de una o varias filas o columnas de una o varias tablas.

```
SELECT* FROM table_source WHERE search_condition
```

Parámetros:

- **table_source** – Tabla sobre la que realizamos la operación.
- **search_condition** – Condición aportada con el fin de restringir las filas/columnas recuperadas por la operación.

INSERT

Agrega una o varias filas a una tabla.

```
INSERT INTO (column_list) VALUES (values_list)
```

Parámetros:

- **column_list** – Lista de columnas que inicializamos en la tabla.
- **values_list** – Valores con los que inicializamos dichas columnas.

UPDATE

Modifica ciertos o todos los registros de una tabla.

```
UPDATE table_source SET column = value WHERE update_condition
```

Parámetros:

- **table_source** – Tabla sobre la que realizamos la operación.
- **column** – Columna a actualizar.
- **value** – Nuevo valor para el registro.
- **update_condition** – Condición aportada con el fin de restringir las filas modificadas por la operación.

DELETE

Elimina ciertos o todos los registros de una tabla.

```
DELETE FROM table_source WHERE delete_condition
```

Parámetros:

- **table_source** – Tabla sobre la que realizamos la operación.
- **delete_condition** – Condición aportada con el fin de restringir las filas eliminadas por la operación.

Funciones nativas PHP

echo

Imprime uno o más *strings*.

```
void echo(string $arg1)
```

Parámetros:

- **arg1** – Mensaje que se desea imprimir.

die

Ejecuta un mensaje de salida y finaliza la ejecución del *script* actual.

```
void die(string $status)
```

Parámetros:

- **status** – Mensaje que se imprime tras la salida.

empty

Determina cuando una variable está vacía. Devuelve FALSE si la variable existe, si no está vacía y su valor es distinto de cero o TRUE en otro caso.

```
bool empty(mixed $var)
```

Parámetros:

- **var** – Variable que se desea evaluar.

date

Formatea la fecha/hora local y devuelve un *string* del formato requerido con la hora actual o la especificada mediante el *timestamp*.

```
string date(string $format, int $timestamp = time())
```

Parámetros:

- **format** – Formato del string de salida. Existen ciertos formatos predefinidos:
  - DATE_ATOM - Ejemplo: 2005-08-15T15:52:01+00:00
  - DATE_COOKIE – Ejemplo: Monday, 15-Aug-2005 15:52:01 UTC
  - DATE_RFC822 - Ejemplo: Mon, 15 Aug 05 15:52:01 +0000
- **timestamp** – Parámetro opcional representado por un entero que consiste en una fecha en formato Unix. Por defecto, se utiliza la hora actual.

strtotime

Convierte una descripción de fecha/hora textual a una fecha *Unix*. Devuelve una marca de tiempo si tuvo éxito o FALSE en caso contrario.

```
int strtotime(string $time, int $now = time())
```

Parámetros:

- **time** – Cadena textual de fecha/hora que se desea convetir.
- **now** – La marca de tiempo que se usa como base para el cálculo de las fechas relativas.

strlen

Obtiene y devuelve la longitud de un string.

```
int strlen(string $string)
```

Parámetros:

- **string** – *String* de cual se desea medir su longitud.

utf8_encode/utf8_decode

Convierte una cadena con los caracteres codificados en *ISO-8859-1* a *UTF-8* o viceversa.

```
string utf8_encode(string $data)

string utf8_decode(string $data)
```

Parámetros:

- **data** – Cadena a la que se le desea aplicar el cambio.

mysql_connect

Abre una conexión al servidor MySQL y devuelve el identificador del enlace en caso de éxito o FALSE en caso de error.

```
resource mysql_connect(string $server,
                       string $username,
                       string $password,
                       bool $new_link = false,
                       int $client_flags = 0)
```

Parámetros:

- **server** – URL del servidor MySQL. También puede especificar un número de puerto.
- **username** – Nombre del usuario que solicita acceso.
- **password** – Contraseña asociada al usuario.
- **new_link** – Establece que el enlace sea reutilizable o se cree uno nuevo con cada llamada al servidor.
- **client_flags** – *Flags* que definen el comportamiento del enlace:
  - MYSQL_CLIENT_SSL – Usar encriptación SSL.
  - MYSQL_CLIENT_COMPRESS – Usar el protocolo de encriptación.
  - MYSQL_CLIENT_IGNORE_SPACE – Permite espacios después de los nombres de función.
  - MYSQL_CLIENT_INTERACTIVE – Utiliza el *timeout* interactivo en vez de la espera tradicional.

mysql_close

Cierra la conexión no persistente al servidor MySQL que está asociada con el identificador de enlace especificado. Devuelve TRUE en caso de éxito o FALSE en caso de error.

```
bool mysql_close(resource $link_identifier = NULL)
```

Parámetros:

- **link_identifier** – Identificador de la conexión con el servidor MySQL. Si no se especifica, se asume el último enlace abierto.

mysql_select_db

Selecciona una base de datos MySQL alojada en un servidor concreto y devuelve TRUE en caso de éxito o FALSE en caso de error.

```
bool mysql_select_db(string $database_name,
```

```
                                    resource $link_identifier = NULL)
```

Parámetros:

- **database_name** – Nombre de la base de datos que va a ser seleccionada.
- **link_identifier** – Conexión con el servidor que aloja la base de datos que se desea seleccionar. Si no se especifica, se asume el último enlace abierto.

mysql_query

Realiza una petición MySQL a una base de datos concreta, alojada en un servidor concreto y devuelve lo propio (según la consulta) en caso de éxito o FALSE en caso de error.

```
mixed mysql_query(string $query,
                  resource $link_identifier = NULL)
```

Parámetros:

- **query** – Petición SQL que se desea realizar.
- **link_identifier** – Conexión con el servidor que aloja la base de datos que a la que se desea trasladar la petición. Si no se especifica, se asume el último enlace abierto.

mysql_fetch_row

Obtiene y devuelve una fila de resultados como un *array* numérico.

```
array mysql_fetch_row(resource $result)
```

Parámetros:

- **result** – Resultado que está siendo evaluado, obtenido de una petición.

mysql_affected_rows

Obtiene y devuelve el número de filas afectadas por la anterior operación MySQL.

```
int mysql_affected_rows(resource $link_identifier = NULL)
```

Parámetros:

- **link_identifier** – Conexión con el servidor que aloja la base de datos que a la que se ha realizado la petición. Si no se especifica, se asume el último enlace abierto.

busqueda.php

Conecta a la base de datos *Control_Acceso* y, mediante un SELECT, busca en la tabla indicada, la fila cuya columna también indicada alberga el valor que se le solicita. Si encuentra dicho valor en la tabla, devuelve la fila completa y en caso de que el script falle en cualquier momento de su ejecución devuelve -1.

```
busqueda.php?tabla = tabla
              &patron_busqueda = columna
              &valor_busqueda = valor
```

Parámetros:

- **`tabla`** – Tabla donde se realiza la búsqueda.
- **`columna`** – Columna que se va a recorrer.
- **`valor`** – Valor que se busca en la columna.

Funciones PHP utilizadas:

- `mysql_connect`
- `mysql_select_db`
- `mysql_query`
- `mysql_fetch_row`
- `empty`
- `die`
- `echo`
- `utf8_encode`
- `mysql_close`

Peticiones SQL realizadas:

- `SELECT`

inserta.php

Conecta a la base de datos *Control_Acceso* y, mediante varias peticiones, inserta una nueva entrada que contiene los valores necesarios para su definición inicial. Si encuentra que la matrícula que se quiere introducir ya está en la tabla, algún valor de los introducidos no es aceptable o el proceso falla en cualquiera de sus pasos, devuelve -1. Por otro lado, si consigue llevar a cabo el registro e inicialización de la nueva entrada, devuelve la *ID* de la nueva entrada.

```
inserta.php?matricula = matricula
            &procedencia = procedencia
            &fecha_esperada = fecha
            &hora_esperada = hora
```

Parámetros:

- **matricula** – Matricula objeto de la nueva entrada.
- **procedencia** – País de procedencia del vehículo.
- **fecha** – Fecha en la que se espera la llegada del vehículo.
- **hora** – Hora a la que se espera la llegada del vehículo.

Funciones PHP utilizadas:

- mysql_connect
- mysql_select_db
- mysql_query
- mysql_fetch_row
- empty
- die
- echo
- date
- strtotime
- strlen
- utf8_decode
- mysql_affected_rows
- mysql_close

Peticiones SQL realizadas:

- SELECT
- INSERT
- DELETE

actualiza.php

Conecta a la base de datos *Control_Acceso* y, mediante un UPDATE, trata de actualizar el valor de la celda situada en la tabla, columna e *ID* indicadas en la entrada, con el valor aportado. Si la celda indicada no existe, el dato no es válido para la celda indicada o se aborta el proceso en cualquier punto de la ejecución del *script*, éste devuelve -1. Por otro lado, si consigue llevar a cabo la actualización de la celda indicada, devolverá 0.

```
actualiza.php?tabla = tabla
               &id = id
               &columna = columna
               &valor = valor
```

Parámetros:

- **tabla** – Tabla que se pretende modificar.
- **id** – Id de la entrada que contiene la celda.
- **columna** – Columna que contiene la celda.
- **valor** – Valor que se pretende usar para la actualización.

Funciones PHP utilizadas:

- `mysql_connect`
- `mysql_select_db`
- `mysql_query`
- `mysql_affected_rows`
- `die`
- `echo`
- `mysql_close`

Peticiones SQL realizadas:

- `UPDATE`

elimina.php

Conecta a la base de datos *Control_Acceso* y, mediante varias peticiones, elimina la entrada que posee la *ID* introducida. Tras eliminarla, si no se trata de la última entrada indicada, reordena las tablas para que las *ID's* continúen siendo correlativas. Si la entrada que intenta eliminar no existe o el proceso falla en cualquiera de sus pasos, devuelve -1. Por otro lado, si consigue llevar a cabo la eliminación de la entrada, devuelve 0.

`elimina.php?id = **id**`

Parámetros:

- **`id`** – Identificador de la entrada que se quiere eliminar.

Funciones PHP utilizadas:

- `mysql_connect`
- `mysql_select_db`
- `mysql_query`
- `mysql_fetch_row`
- `mysql_affected_rows`
- `die`
- `echo`
- `mysql_close`

Peticiones SQL realizadas:

- `DELETE`
- `SELECT`
- `UPDATE`

registrallegada.php

Conecta a la base de datos *Control_Acceso* y, mediante varias peticiones, trata

de registrar la llegada del vehículo asociado a la ID introducida. El script comprueba la situación de la entrada asociada a la ID y en el caso de ser *'Esperando'* actualiza su hora de llegada en la tabla "*Llegadas*" y su hora esperada de salida (una hora y media despues), en la tabla "*Salidas*". Tras llevar a cabo la actualización, cambia la situación del vehículo por *'En_Base'* en ambas tablas. Si la situación de la entrada no es *'Esperando'* o el proceso falla en cualquier paso de su ejecución, devuelve -1. Por otro lado, si consigue llevar a cabo el registro de la llegada, devuelve 0.

`registrallegada.php?id = **id**`

Parámetros:

- **`id`** – Identificador de la matricula que se quiere registrar.

Funciones PHP utilizadas:

- `mysql_connect`
- `mysql_select_db`
- `mysql_query`
- `mysql_fetch_row`
- `mysql_affected_rows`
- `date`
- `strtotime`
- `die`
- `echo`
- `mysql_close`

Peticiones SQL realizadas:

- `SELECT`
- `UPDATE`

registrasalida.php

Conecta a la base de datos *Control_Acceso* y, mediante varias peticiones, trata de registrar la salida del vehículo asociado a la ID introducida. El script comprueba la situación de la entrada asociada a la ID y en el caso de ser *'En_Base'* actualiza su hora de salida en la tabla "*Salidas*". Tras llevar a cabo la actualización, cambia la situación del vehículo por *'Servido'*, tanto en "*Salidas*" como en "*Llegadas*". Si la situación de la entrada no es *'En_Base'* o el proceso falla en cualquier paso de su ejecución, devuelve -1. Por otro lado, si consigue llevar a cabo el registro de la llegada, devuelve 0.

`registrasalida.php?id = **id**`

Parámetros:

- **`id`** – Identificador de la matricula que se quiere registrar.

Funciones PHP utilizadas:

- `mysql_connect`

- mysql_select_db
- mysql_query
- mysql_fetch_row
- mysql_affected_rows
- date
- die
- echo
- mysql_close

Peticiones SQL realizadas:

- SELECT
- UPDATE

7.3. Anexo III: Presupuesto

Dentro de este anexo se incluye el presupuesto aproximado de la instalación y puesta en marcha del sistema según el escenario propuesto, incluyendo una parte variable en función de la distancia real del entorno:

Concepto	Descripción	PVP (€)	Precio
Cámara	Cámara (F) IR para Rpi	36,00	
	2 IR Led de 1W para cámara Rpi	15,90	
			51,90
Procesador	Raspberry Pi 2 Model B	41,95	
	Carcasa de acero inoxidable color rojo RAL 3002 con aberturas para cámara	80,00	
	Soporte de acoplo de acero inoxidable color rojo RAL 3002 con aberturas para cables	30,00	
	Cableado eléctrico (3x2,5mm)	1,48 / m	
	Cableado de datos (2x1,5mm)	1,00 / m	
	Instalación y conexionado de procesador	250,00	
			410,95 + 2,48 / m1
Barrera	FORSA: BARRERA NIGHT&DAY-5 S/ASTA	2300,00	
	Instalación losa cemento	200,00	
	Cableado eléctrico (3x2,5mm)	1,48 / m	
	Cableado de datos (2x1,5mm)	1,00 / m	
	Instalación cableado	250,00	
	Instalación barrera	200,00	
	Conexionado barrera-procesador	250,00	
	Revisión instalación, cableado y barrera	250,00	
			3450 + 2,48 / m2
Interfono	FERMAX: KIT PORTERO CITY BUS 1/L	370,00	
	Cableado eléctrico (6x1,5mm)	3,50(m)	
	Instalación cableado	100	
	Soporte colocación interfono	100	
	Conexionado interfono-procesasor	200	
	Revisión instalación	250	
			1020 + 3,50 / m3
Software	Licencia SW gestión y control de acceso	1500	
	Configuración del SW	500	
			2000,00
	SUBTOTAL		6932,85+2,48(m1+m2)+3,5(m3)
	21% IVA		1455,89+0,52(m1+m2)+0,73(m3)
	TOTAL		8388,74+3(m1+m2)+4,23(m3)

8. Bibliografía

En éste apartado se incluyen las fuentes de todas las referencias hechas durante la presente memoria:

[1] "Visión Artificial", CIP ETI Tudela. http://www.etitudela.com/celula/downloads/visionartificial.pdf

[2] "Sistemas de Percepción y Visión por Computador", Alberto Ruiz García. Universidad de Murcia. (http://dis.um.es/profesores/alberto/material/percep.pdf)

[3] "Digital Image Processing", R. C. González, R. E. Woods. Prentice Hall (3º Edición)

[4] "Óptica tradicional y moderna". La ciencia para todos., D. Malacara. (1997) (http://bibliotecadigital.ilce.edu.mx/sites/ciencia/volumen2/ciencia3/084/htm/sec_9.htm)

[5] "Rasgos descriptores para el reconocimiento de objetos", J. H. Sossa-Azuela. Ra-Ma (2011)

[6] "Procesamiento digital de imágenes", R. C. González, P. Wintz, Addison-Wesley. (http://dmi.uib.es/~catalina/docencia/PDS/tema2.pdf)

[7] "Técnicas y algoritmos básicos de visión artificial", A. G. Marcos. Universidad de la Rioja

[8] "Introducción a la morfología matemática de conjuntos", J. L. Díaz de León-Santiago y C. Yáñez-Márquez.

[9] "An Introduction to Morphological Image Processing", E. R. Dougherty, SPIE Press, Bellingham, WA, (1992).

[10] "OCR (Optical Character Recognition)", J. Cano, J. C. Pérez.

[11] "Introducción a las librerias OpenCV", S. de la Llana, S. Molina, S. Sánchez. http://web-sisop.disca.upv.es/imd/cursosAnteriors/2k3-2k4/copiaTreballs/serdelal/trabajoIMD.xml)

[12] "An overview of the Tesseract OCR engine", R. Smith (http://tesseract-ocr.googlecode.com/svn/trunk/doc/tesseracticdar2007.pdf)

[13] "Bases de datos en MySQL", L. A. Casillas, M. Gibert, Ó. Pérez. Universitat Oberta de Catalunya. http://ocw.uoc.edu/computer-science-technology-and-multimedia/bases-de-datos/bases-de-

datos/P06_M2109_02151.pdf)

[14] "Manual de PHP", M. Achour, F. Betz, A. Dovgal (http://docs.php.net/manual/es/)

[15] "Documentación de la API C de libcurl", (http://curl.haxx.se/libcurl/c/)

[16] "Single-board computer", Wikipedia (http://en.wikipedia.org/wiki/Single-board_computer)

[17] "Raspberry-Pi", Wikipedia (http://es.wikipedia.org/wiki/Raspberry_Pi)

[18] "Wiring-Pi", G. Drogon. (https://projects.drogon.net/raspberry-pi/wiringpi/)

[19] "Image Analysis and Mathematical Morphology, volume I". J. Serra Academic Press, London, 1982.

[20] "Image Analysis and Mathematical Morphology: Theoretical Advances, volume II". J. Serra. Academic Press, London, 1988.