Administración y Organización de Computadores

Curso 2019-2020

<u>Práctica</u>: *Detección de matrículas*

El principal objetivo de esta práctica es completar una aplicación, escrita en *C++* sobre Linux, a través de la implementación en ensamblador de las diferentes operaciones que debe proporcionar el programa. La integración de los dos lenguajes se llevará a cabo mediante las facilidades de ensamblado en línea proporcionadas por el compilador *gcc*. La programación en ensamblador se realizará considerando la arquitectura de un procesador Intel o compatible de 64 bits.

A continuación, se detallan diferentes aspectos a tener en cuenta para el desarrollo de esta práctica.

<u>Descripción de la aplicación</u>

El código que deberá ser completado estará incluido en una aplicación *C++*, disponible como proyecto de Qt. La siguiente figura muestra una captura de pantalla en un instante de ejecución de la aplicación:

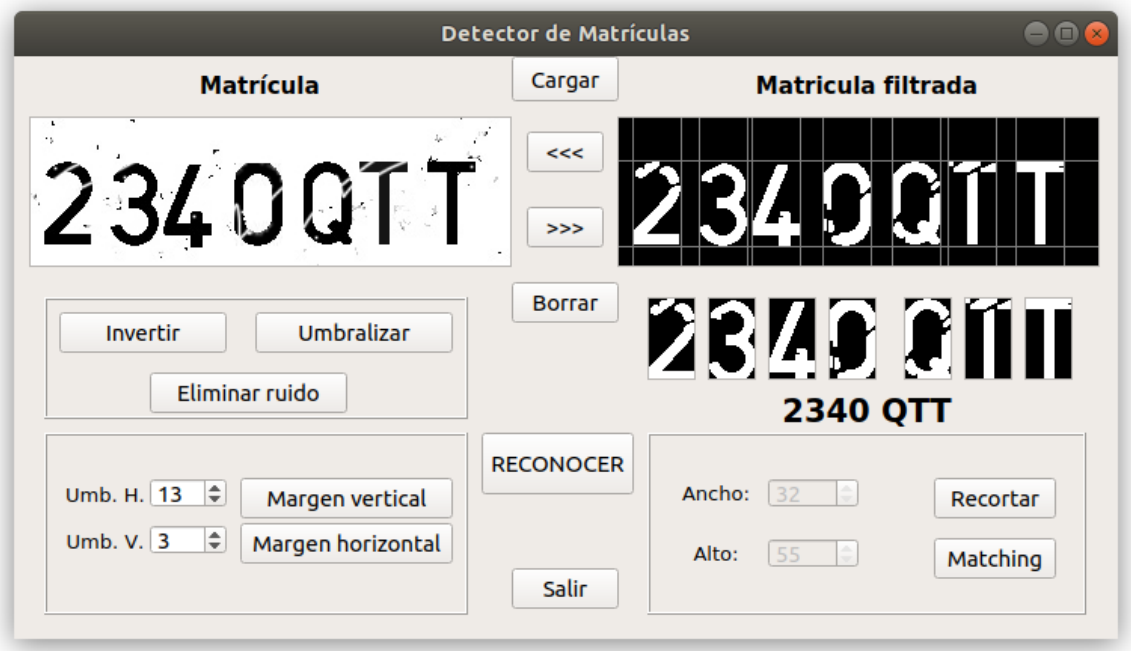


Figura 1: interfaz principal de la aplicación

¿Qué es Qt?

Qt es un *framework* de desarrollo de aplicaciones que, entre otras aportaciones, proporciona herramientas y librerías de clases para la creación de interfaces de usuario en entornos de escritorio.

Puesta en marcha del proyecto

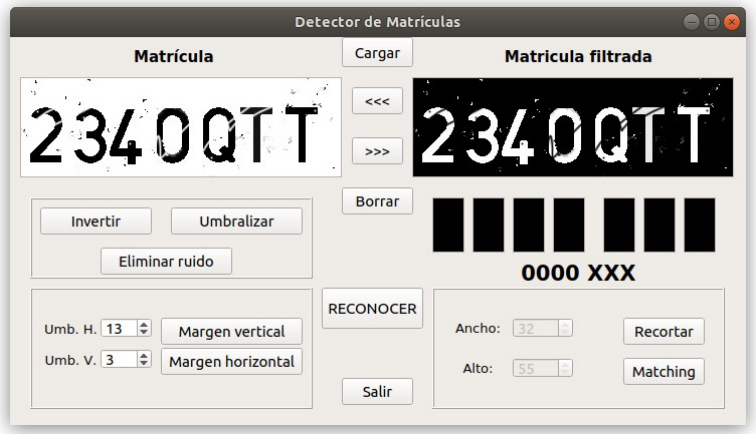
El fichero que contiene la descripción del proyecto (*pracaoc.pro*) se encuentra disponible en la carpeta principal de la aplicación. Dicho fichero puede ser utilizado para importar el proyecto desde diferentes entornos de desarrollo. No obstante, se recomienda trabajar con Qt Creator (paquete *qtcreator*). Es necesario instalar además los paquetes *qt5-default* y *qttools5-dev-tools*.

Funcionamiento de la aplicación

La interfaz principal (figura 1) está compuesta por varias ventanas de imagen y una serie de botones a través de los cuales es posible ejecutar las diferentes operaciones que proporciona la aplicación. Las dos ventanas de imagen situadas en la parte superior se corresponden con la imagen de la matrícula original (ventana de la izquierda) y con la imagen una vez filtrada (ventana de la derecha). La primera es usada como punto de partida del proceso de detección, mientras que la segunda muestra el resultado tras la aplicación de distintas fases del proceso. Estas imágenes tienen inicialmente un tamaño de 320x100 pixels y 255 niveles de gris originalmente aunque en fases posteriores se desechan todos los niveles excepto el 0 (negro) y 255 (blanco). El conjunto de 7 imágenes situadas debajo de la matrícula filtrada tienen un tamaño de 32x55 píxeles en blanco (255) y negro (0) y son usadas para la última fase del proceso de detección. Se describen a continuación las opciones incluidas en la aplicación:

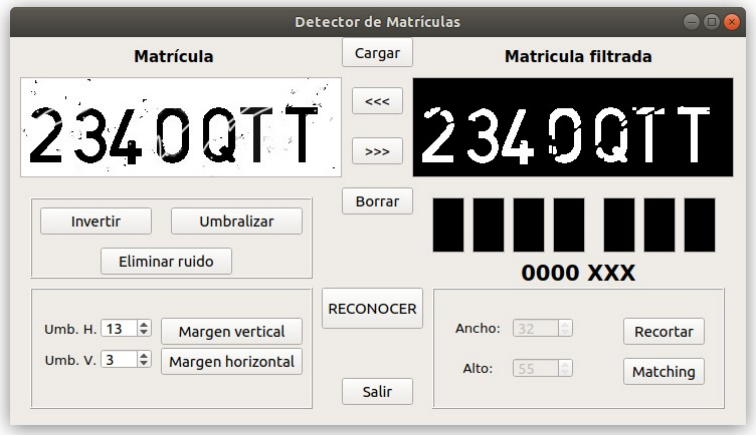
- **Cargar**: carga, en la imagen origen, la imagen del fichero indicado. La imagen debe tener un tamaño mínimo de 320x100. Aunque pueden cargarse imágenes mayores, sólo se visualizará un recorte 320x100 píxeles como máximo. Es posible cargar imágenes en color, en tal caso esta opción se encarga de transformar la imagen a escala de grises.
- >>>: copia la imagen origen en el contenedor destino.
- <<<: copia una imagen procesada en el contenedor origen.
- **Borrar**: borra el contenido de las imágenes destino poniendo a 0 (negro) cada uno de sus píxeles. Además inicializa ciertos datos, fruto de procesos anteriores.
- **RECONOCER**: realiza todos los pasos para la detección de matrículas de forma automática. Empieza con la inversión de la imagen y termina con el proceso de *Matching*.
- **Salir**: cierra la aplicación.

- **Invertir** : el nivel de gris de cada píxel de la imagen origen es invertido (255 – nivel de gris) y almacenado en la imagen destino.

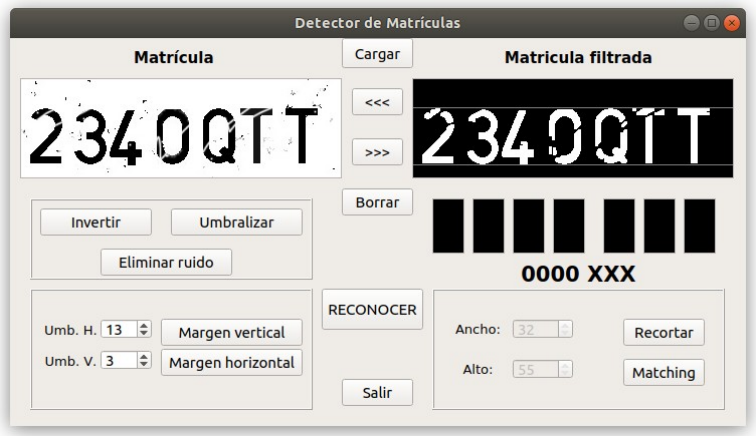


- **Umbralizar**: Transforma la imagen en escala de grises en una imagen en blanco y negro para lo cual aplica a cada punto de la imagen origen la comparación con el nivel de gris medio (128). Los píxeles con valores superiores pasan a valer 255 (blanco) mientras que el resto serán 0 (negro).

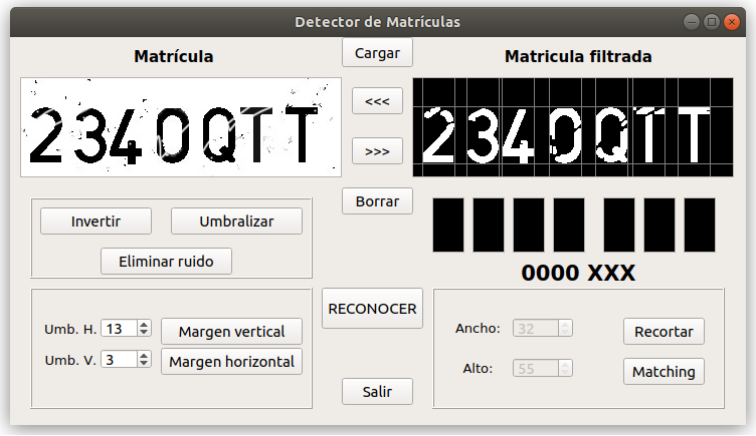
- **Eliminar ruido**: elimina el posible ruido en la imagen utilizando dos fases de procesamiento, una vez que la imagen ha sido invertida y umbralizada. En la primera fase, por cada píxel, comprueba si todos los puntos de su entorno (3x3) son blancos. En tal caso pone a 255 el píxel en el destino y, en caso contrario, lo pone a 0. En la segunda fase, vuelve a analizar el entorno de cada píxel de la imagen generada por la fase anterior. Si existe algún píxel blanco en su entorno, pone a 255 el píxel en la imagen destino y, en caso contrario, lo pone a 0.



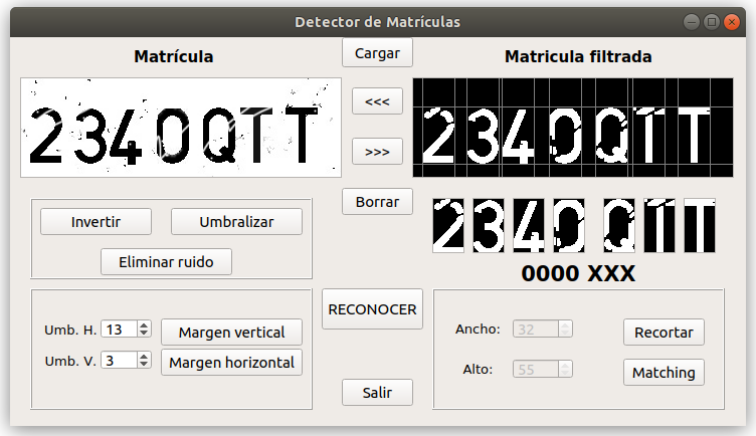
- **Margen vertical**: Partiendo de la imagen derecha, una vez aplicadas las operaciones anteriores, determina los limites en fila superior e inferior a partir de los cuales hay caracteres alfanuméricos. Para realizar esta operación, se aplica la integral proyectiva a cada fila de la imagen (suma de píxeles blancos de la fila). Busca las dos primeras filas desde arriba y desde abajo que cumplan que la suma de sus puntos sea mayor que un valor umbral determinado en la aplicación.
  Una vez determinadas las filas superior e inferior, la aplicación las marca en la imagen de proceso.



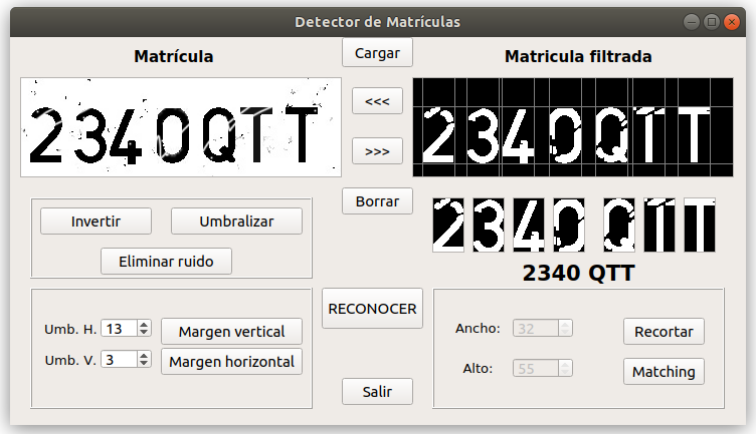
- **Margen horizontal**: Al igual que la operación anterior, realiza la integral proyectiva de cada columna de la imagen procesada de tal forma que determina en que bandas de columna existen dígitos y en cuales no. Lo hace en dos fases, primero calcula que columnas contienen puntos blancos por encima del umbral fijado para posteriormente fijar el punto medio de cada zona. De esta forma se obtiene la columna central aproximada de cada dígito en la imagen.
  El proceso termina marcando en la imagen filtrada los limites aproximados entre los que se encuentra cada uno de los caracteres alfanuméricos (columna_central-16, columna_central+16).



- **Recortar**: Con los dos procesos anteriores se puede obtener la ventana de imagen donde aparece cada uno de los caracteres de una matrícula. A partir de las posiciones de estas ventanas, teniendo en cuenta que cada carácter ocupa aproximadamente 32x55 píxeles, este proceso se encarga de recortar de la imagen procesada cada una de las imágenes individuales de caracteres y de copiarlas a las ventanas de imagen correspondientes.



- ***Matching***: Este es el último proceso y el más complejo. Finaliza cuando se obtiene una cadena de caracteres que identifican a los de la imagen.
  Coteja cada una de las imágenes obtenidas del proceso anterior con las almacenadas en la aplicación (0..9, B..Z). En concreto, son 31 imágenes de tamaño 32X55 en blanco y negro que contienen cada uno de los caracteres necesarios para formar matrículas de coche. Teniendo en cuenta que los 4 primeros dígitos deben de ser numéricos y los 3 últimos alfabéticos, es posible minimizar el número de comparaciones y aumentar la posibilidad de acierto.
  El proceso se realiza para cada una de las imágenes obtenidas del proceso anterior (*recortar)* y consiste en comparar cada imagen con un conjunto posible. La comparación con cada imagen del conjunto da como resultado un valor que representa la similitud entre las dos imágenes. El carácter asociado con la imagen del conjunto para la que el valor de similitud sea mayor, se considera el carácter finalmente identificado.



Componentes de la aplicación

El código fuente de la aplicación está formado por 5 ficheros: *main.cpp*, *pracaoc.cpp*, *pracaoc.h*, *imageprocess.cpp* e *imageprocess.h*. Además, existe un formulario de Qt (*mainForm.ui*) y otros ficheros necesarios para la gestión del proyecto. El contenido de cada fichero fuente es el siguiente:

- *main.cpp*: contiene el procedimiento principal que permite lanzar la aplicación, así como crear y mostrar la ventana principal que actúa como interfaz entre el usuario y la aplicación.
- *pracaoc.h*: fichero que contiene la definición de la clase principal de la aplicación (*pracAOC*). Esta clase contiene los elementos principales de gestión de la aplicación. Entre los atributos, se encuentran las definiciones de las interfaces de usuario incluidas en el programa, así como de las variables que permiten almacenar la información de las imágenes origen y procesada utilizadas en las opciones de procesamiento de la aplicación.
- *pracaoc.cpp*: Incluye la implementación de los métodos de la clase *pracAOC*. En su mayoría, estos métodos se encargan de responder a los distintos eventos de la interfaz de usuario incluida en el programa y de llamar a las funciones de procesamiento de imagen que correspondan en cada caso (disponibles en *imageprocess.cpp* e *imageprocess.h*).
- *imageprocess.h*: contiene la definición de las funciones implementadas en el fichero *imageprocess.cpp*.
- *imageprocess.cpp*: implementación de las funciones de procesamiento de imagen que se ejecutan a través de las distintas opciones de la aplicación. La mayoría de estas funciones contienen una implementación vacía. El objetivo de esta práctica es completarlas para que el funcionamiento de la aplicación sea el descrito anteriormente.

Extensiones principales en x86-64

- En relación a la representación de datos en memoria, en 64 bits, los punteros y los datos de tipo *long* ocupan 64 bits. El resto de tipos mantiene el mismo tamaño que en la línea de procesadores de 32 bits (int: 4 bytes, short: 2 bytes, …).
- Todas las instrucciones de 32 bits pueden utilizarse ahora con operandos de 64 bits. En ensamblador, el sufijo de instrucción para operandos de 64 bits es “q”.
- Los registros de 32 bits se extienden a 64. Para hacer referencia a ellos desde un programa en lenguaje ensamblador, hay que sustituir la letra inicial “e” por “r” (%rax, %rbx, …). Los registros de 32 bits de la IA32 se corresponden con los 32 bits de menor peso de estos nuevos registros.

- El byte bajo de los registros %rsi, %rdi, %rsp y %rbp es accesible. Desde el lenguaje ensamblador, el acceso a estos registros se realiza a través del nombre del registro de 16 bits finalizado con la letra “l” (%sil, %dil, …).
- Aparecen 8 nuevos registros de propósito general de 64 bits (%r8, %r9, …, %r15). Es posible acceder a los 4, 2 o al último byte de estos registros incluyendo en su nombre el sufijo d, w o b (%r8d – 4 bytes, %r8w – 2 bytes, %r8b – 1 byte).

Ejemplo de implementación en ensamblador x86-64 de una función de C/C++

Dentro del fichero *imageprocess.cpp*, se ha incluido la implementación de la primera función (*copiar*) a modo de ejemplo. Dicha implementación se detalla a continuación. La función *copiar* es la única función de *imageprocess.cpp* que se encuentra implementada. Como ya se ha comentado anteriormente, la implementación de las restantes es objeto de esta práctica.

```
void imageprocess::copiar(uchar * imgO, uchar * imgD, int tam)
{

   asm volatile(
         "movq %0, %%rsi;"
         "movq %1, %%rdi;"
         "movsxd %2, %%rcx;"
            “BCopiar:
                        “mov (%%rsi), %%al;”
                        “mov %%al, (%%rdi);”
                        “inc %%rsi;”
                        “inc %%rdi;”
         "loop BCopiar;"
         :
         : "m" (imgO), "m" (imgD), "m"(tam)
         : "%rsi", "%rdi", "%rax", "%rcx", "memory"

   );

}
```

La función *copiar* es invocada en la ejecución de las opciones “>>>” y “<<<” para copiar la imagen origen en la destino y viceversa. Esta función incluye como parámetros un array que contiene la imagen a copiar (*imgO*) y un segundo array que especifica la imagen donde se debe realizar la copia (*imgD*). Ambos parámetros son tratados como operandos de entrada del bloque de código en ensamblador (la lista de operandos de salida está vacía). El motivo para que esto sea así es que en ningún caso se van a modificar estos parámetros, puesto que su contenido es la dirección de comienzo de los bloques de memoria donde se encuentran las dos imágenes. Lo que sí se va a modificar es el contenido del bloque de memoria apuntado por *imgD*, pero esto no afecta a la dirección almacenada en dicho parámetro.

Tras la definición de los operandos, se incluye la lista de registros utilizados dentro del código. Además de los registros indicados, dado que la memoria es modificada, la lista incluye también la palabra “memory”.

Una vez aclaradas estas definiciones, analicemos a continuación paso a paso el código ensamblador incluido en el procedimiento:

- Las dos primeras instrucciones se encargan de copiar las direcciones iniciales de memoria de las dos imágenes, indicadas por los dos operandos (%0=imgO, %1=imgD), en dos registros, *%rsi* y *%rdi*, para su posterior direccionamiento. Así, a través del registro *%rsi* podremos acceder a cada uno de los píxeles de *imgO* y, mediante el registro *%rdi*, tendremos acceso a los píxeles de *imgD*.
- La copia de cada píxel de *imgO* en la imagen *imgD* se realiza mediante un bucle con tantas iteraciones como indica el tamaño de las imágenes (parámetro *tam*). Este bucle se lleva a cabo mediante la instrucción *loop*, por lo que, lo siguiente que hace el

procedimiento es inicializar el registro *%rcx* con el total de iteraciones. Dado que la variable *tam* ocupa 32 bits y el registro *%rcx* tiene un tamaño de 64 bits, la transferencia se lleva a cabo con la instrucción *movsxd*, que permite transferir, extendiendo el signo, el contenido de un operando fuente de 4 bytes a un registro destino de mayor tamaño.

- Una vez inicializado *%rcx*, comienza el bucle de copia. Cada iteración consiste en la copia del píxel actual de *imgO* – (*%%esi*) – en el píxel correspondiente de *imgD* – (*%%edi*) –. Dicha copia se lleva a cabo a través del registro *%al*, puesto que no es posible indicar los dos elementos de memoria como operandos de la instrucción *mov*. Tras esta operación y antes de pasar a la siguiente iteración del bucle, los índices *esi* y *edi* son incrementados para que apunten a los siguientes elementos de *imgO* e *imgD*.

Aunque esta es la forma más fácil de implementar este proceso, no es sin embargo la forma más eficiente. A continuación se propone una forma alternativa usando en este caso las instrucciones especificas para gestionar vectores:

```
void imageprocess::copiar(uchar * imgO, uchar * imgD, int tam)
{

   asm volatile(
         "movq %0, %%rsi;"
         "movq %1, %%rdi;"
         "movsxd %2, %%rcx;"
         "rep movsb;"
         :
         : "m" (imgO), "m" (imgD), "m"(tam)
         : "%rcx", "%rsi", "%rdi", "memory"

   );

}
```

Como se puede ver, el bucle se sustituye por la instrucción *rep movsb*. Esta instrucción copiará valores entre los datos apuntados por *%rsi* y *%rdi* tantas veces como indique *%rcx*.

Especificación de los objetivos de la práctica

El principal objetivo de esta práctica es completar el código de la aplicación descrita para que su funcionamiento sea el que se detalla en la sección "*Funcionamiento de la aplicación"*, incluida es esta documentación. Para ello, se deberán implementar, en lenguaje ensamblador, los procedimientos "vacíos" del módulo "imageprocess.cpp". Estos procedimientos son invocados por las distintas opciones del programa. Para cada uno de ellos, se proporciona la estructura inicial del bloque ensamblador, en la que se ha incluido la definición de operandos que afectan a la implementación. La lista de registros utilizados incluye únicamente la palabra "memory", ya que en todos los casos la memoria es modificada. La inclusión de registros dentro de esta lista dependerá de la implementación que se desarrolle en cada caso, por lo que, será necesario completarla para cada uno de los procedimientos.

Se describe a continuación la funcionalidad de cada uno de los procedimientos a completar. Para todos ellos, los parámetros *imgO* e *imgD* contienen, respectivamente, la dirección de los bloques de memoria que almacenan el contenido de las imágenes origen y destino. Para una descripción más detallada de estas funciones, se recomienda revisar la sección "Funcionamiento de la aplicación".

- ***void imageprocess::borrar****(uchar * imgD,* ***int*** *w,* ***int*** *h)*

  Borra todos los píxeles de *imgD*, asignándoles un valor 0 (negro) a cada uno de ellos. El tamaño de la matriz a procesar viene determinada por *w* (ancho de imagen) y *h* (alto de imagen).

- ***void imageprocess::invertir****(uchar * imgO, uchar * imgD,* ***int*** *w,* ***int*** *h)*

  Invierte el nivel de gris de cada píxel de *imgO* y devuelve el resultado en *imgD*. Para ello asigna a cada píxel de la imagen destino 255 menos el valor del píxel correspondiente de la imagen origen. Al igual que en el procedimiento anterior, *w* y *h* determinan las dimensiones de las imágenes.

- ***void imageprocess::umbralizar****(uchar * imgD,* ***int*** *w,* ***int*** *h)*

  Transforma los píxeles de *imgD* poniendo a 255 aquellos que tengan valor superior a 128 y a 0 los restantes.

- ***void imageprocess::eliminarRuido_F1****(uchar *****imgO****, uchar *****imgD****, int* ***w****, int* ***h****)*

  Realiza la primera fase de la opción "*Eliminar ruido*". Para ello, por cada píxel de *imgO* comprueba si todos los píxeles de su entorno de 3x3 tienen valor 255. En tal caso asigna el valor 255 en el píxel correspondiente de *imgD* y, en caso contrario, el valor 0.

- ***void imageprocess::eliminarRuido_F2****(uchar *****imgO****, uchar *****imgD****, int* ***w****, int* ***h****)*

  Partiendo del resultado del procedimiento anterior, realiza la segunda fase de la opción "*Eliminar ruido*". Concretamente, por cada píxel de *imgO* comprueba si algún píxel de su entorno de 3x3 tienen valor 255. En tal caso asigna el valor 255 en el píxel correspondiente de *imgD* y, en caso contrario, el valor 0.

- ***int imageprocess::detectarV_min****(uchar *imgD,* ***int*** *U)*

  Busca la fila inicial de la imagen a partir de la cual se sitúan los caracteres de la matrícula. Para ello, comenzando por la fila 0, cuenta el número de píxeles de la fila con valor igual a 255. Si el número de píxeles es mayor que *U* retorna la posición de la fila actual. En caso contrario, repite el proceso para la siguiente fila.

- ***int imageprocess::detectarV_max****(uchar *imgD,* ***int*** *U)*

  Busca la última fila de imagen donde están situados los caracteres de la matrícula. Realiza el mismo proceso que el procedimiento anterior, pero comenzando por la última fila de imagen (99). Cuando el número de píxeles blancos de la fila actual es superior al valor indicado en *U*, retorna dicha posición de fila. En caso contrario, repite el proceso para la fila anterior.

- ***void imageprocess::detectarH_f1****(uchar *imgD,* ***uchar*** *VA, uchar U)*

  Este procedimiento realiza la primera fase del procesamiento de la opción "*Margen horizontal*". En concreto, por cada columna de *imgD*, cuenta el número de píxeles blancos y, si dicho número es superior al indicado en el parámetro *U*, almacena un 1 en la posición correspondiente de *VA*. En caso contrario, pondrá a 0 el elemento que corresponda de *VA*. Hay que considerar que la variable *VA* representa un array con tantos elementos como columnas tiene *imgD*.

- ***void imageprocess::detectarH_f2****(uchar *VA,* ***int*** **Vh)*

  Lleva a cabo la segunda fase del procesamiento de la opción "*Margen horizontal*". Parte del resultado obtenido por el procedimiento anterior, almacenado en *VA*. Se encarga de recorrer dicho array y localizar el primer índice cuyo elemento tiene valor 1 (*i*). A partir de esa posición, se cuenta el número de 1's consecutivos (*t*) y se calcula la posicion central de la zona detectada como (*i* + *t*/2). Dicha posición se almacena en *Vh* y se continúa con el recorrido de *VA* hasta localizar el siguiente 1, en cuyo caso se repite el proceso anterior, o hasta que se haya alcanzado el final del vector (320 elementos).

- ***void imageprocess::recortar****(uchar *imgO, uchar *imgD,****int*** *x,* ***int*** *y,* ***int*** *w,* ***int*** *h)*

  Copia una ventana de imagen de *imgO* a *imgD*. El tamaño de la ventana está determinado por *w* (ancho) y *h* (alto). La esquina superior izquierda de la ventana viene indicada por *x* (columna) e *y* (fila).
  Hay que considerar que el tamaño de *imgO* en cualquier caso es 320x100.

- ***int imageprocess::matching****(uchar *caracM,* ***uchar*** **patrones[31],* ***int*** *ini,* ***int*** *tam)*

  Este procedimiento recibe por parámetro una imagen (*caracM*) de tamaño 32x55, que contiene uno de los caracteres de matrícula obtenidos de la fase anterior, y un vector de imágenes (*patrones*) también de tamaño 32x55. Cada imagen de este vector contiene la representación estándar (patrón) de uno de los posibles caracteres de una matrícula (números y letras). El procedimiento compara la imagen almacenada en *caracM* con las imágenes disponibles en el vector *patrones* y devuelve el índice del vector que contiene la imagen más similar. Para evitar comparar imágenes que contienen números con imágenes de letras y viceversa, el procedimiento recibe por parámetro el índice inicial del vector (*ini*) que contiene la primera imagen con la que se debe comparar y el número de patrones posibles (*tam*).
  Para obtener un valor que cuantifique la similitud entre la imagen *caracM* y el patrón correspondiente, se llevará a cabo una comparación píxel a píxel de la siguiente forma: si los dos píxeles son iguales y tienen valor 255, la similitud se incrementa en 3; si ambos son iguales y su valor es 0, la similitud se incrementa en 1; si los dos píxeles son distintos, la similitud se decrementa en 1. Siguiendo este proceso, el patrón que obtenga el máximo valor de similitud, será el mejor candidato y el procedimiento retornará el índice de dicho patrón en el vector.

La implementación de los procedimientos "*eliminarRuido_F1*", "*eliminarRuido_F2*" y "*Matching*" será opcional. La nota máxima que podrá obtenerse sin la realización de dichos procedimientos será de Notable (8).

Nota: la práctica se realizará de manera individual.

Fecha de entrega de la práctica: 20 de enero de 2020 (hasta las 15:00)

Entrega**:** la entrega se realizará a través de la subida de un archivo .zip, que contenga el proyecto completo, al aula virtual de la asignatura.