Administración y Organización de Computadores

Curso 2015-2016

<u>Práctica</u>: *Procesamiento de imágenes en ensamblador*

El principal objetivo de esta práctica es completar una aplicación de procesamiento de imágenes, 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 32 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:

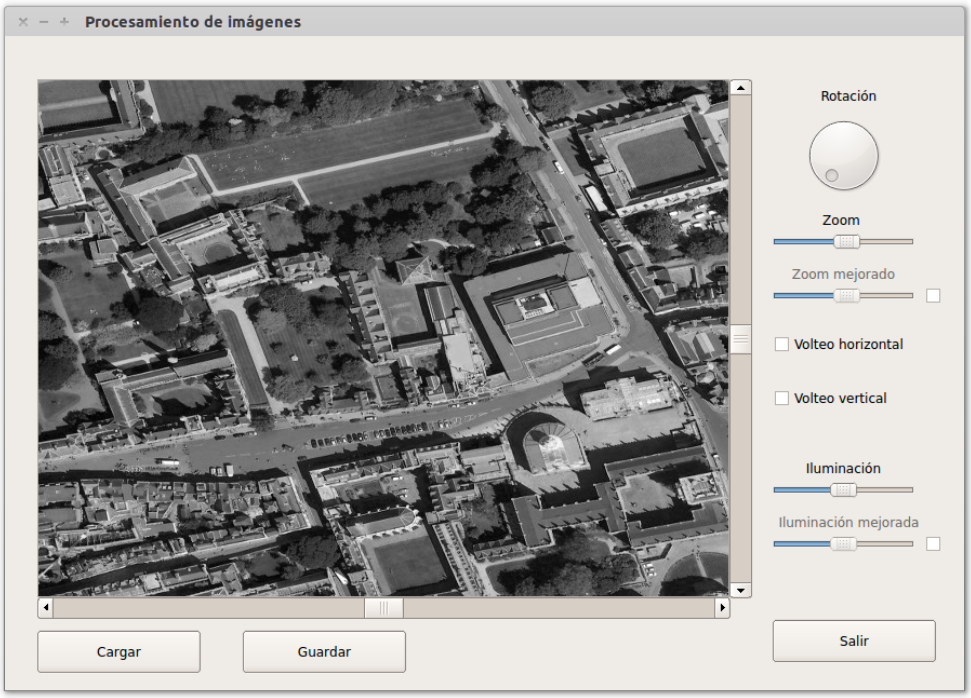

Figura 1: interfaz principal de la aplicación

¿Qué es Qt?

Qt es un marco 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 *libqt4-dev* y *qt4-dev-tools*, en sistemas de 32 bits, o *libqt4-dev:i386* y *qt4-dev-tools:i386* en sistemas de 64 bits.

Funcionamiento de la aplicación

La interfaz principal (figura 1) está compuesta por una ventana de imagen y una serie de elementos gráficos a través de los cuales es posible ejecutar las diferentes operaciones que proporciona la aplicación. La ventana de imagen tiene un tamaño de 640x480 píxeles (640 columnas y 480 filas) y se corresponde con la zona visible de la imagen procesada. Es posible moverse por dicha imagen utilizando las barras de desplazamiento horizontal y vertical que se encuentran situadas junto a la ventana. Cada píxel de la imagen está representado por un nivel de gris (0-negro, 255-blanco).

Se describen a continuación las opciones incluidas en la aplicación:

- ***Cargar***: carga la imagen del fichero indicado. Si la imagen del fichero tiene una resolución superior a la soportada por el programa (640x480), ésta es recortada. Además, es posible cargar imágenes en color. En tal caso, esta opción se encarga de transformar la imagen a escala de grises.

- ***Guardar***: guarda, en el fichero indicado, la zona visible de la imagen procesada.

- ***Rotación***: gira la imagen según el ángulo (α) indicado por el usuario a través del dial situado debajo del nombre de la opción. Suponiendo un píxel de la imagen transformada (*imgD*) situado en una posición (*fD*, *cD*), dicho píxel tomará el valor del píxel de la imagen origen (*imgO*) que se encuentra en la posición (*fO*, *cO*), teniendo en cuenta la siguiente relación entre ambas posiciones:

$$fO = sin(\alpha)*cD + cos(\alpha)*fD$$
$$cO = cos(\alpha)*cD - sin(\alpha)*fD$$

Figura 2: ejecución de la opción Rotación.

- ***Zoom***: amplía o reduce la imagen teniendo en cuenta el factor de escala indicado por el usuario a través de la barra de desplazamiento situada debajo del nombre de la opción. Para un factor de escala *s*, cada píxel de la imagen destino situado en una determinada posición (*fD*, *cD*), toma el valor del píxel de la imagen origen situado en la posición (*fO*, *cO*) que se obtiene de la siguiente relación:

$$fO = fD/s \qquad cO = cD/s$$

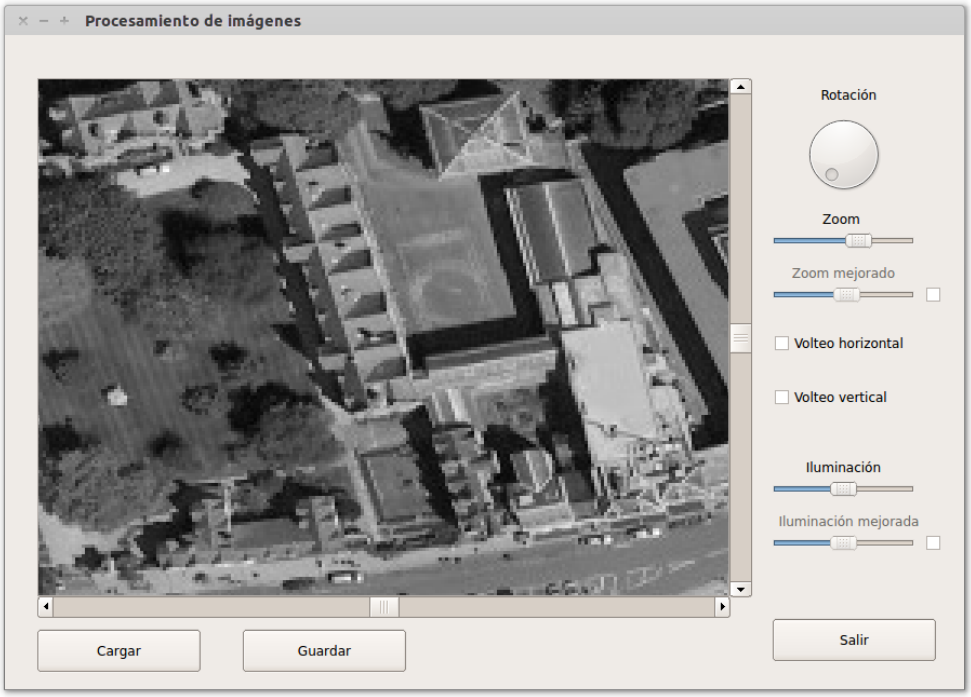

Figura 3: ejecución de la opción *Zoom*.

- ***Zoom mejorado***: al igual que la opción anterior, esta opción permite ampliar o reducir la imagen a partir del factor de escala indicado por el usuario. En este caso, sin embargo, los efectos de pixelado que se producen en la ampliación de una imagen por la opción anterior se reducen aplicando un proceso de interpolación bilineal. A través de

este proceso, el valor de un píxel en la imagen destino se obtiene aplicando una media ponderada de los 4 píxeles más cercanos de su entorno. Esto proporciona un resultado mucho más suave que el que se obtiene por el proceso anterior.

Para aplicar esta operación es necesario activar la casilla de verificación situada junto a la opción.

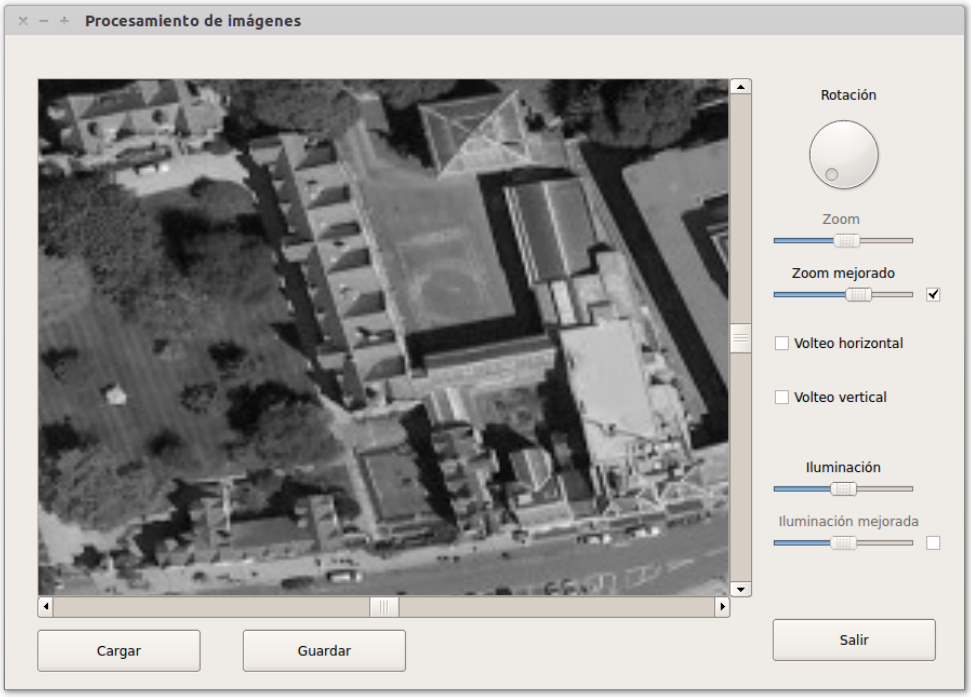

Figura 4: ejecución de la opción *Zoom mejorado*.

- ***Volteo horizontal***: voltea la imagen en horizontal. Cada píxel de la imagen resultante toma el valor del píxel situado en la columna opuesta en la imagen origen:

$$imgD[f, c]=imgO[f, 640-c]$$

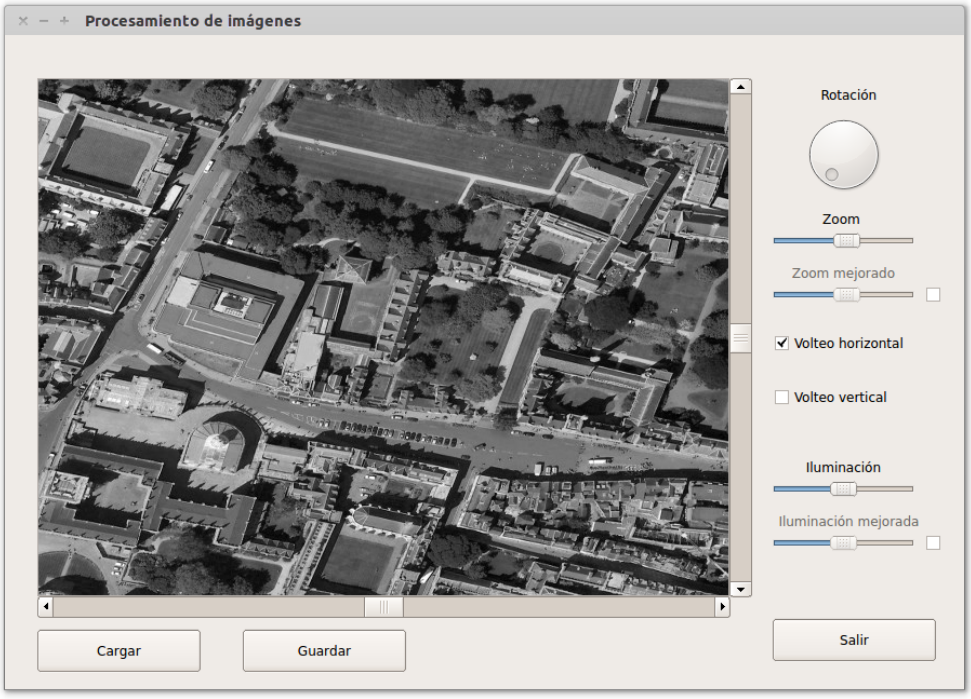

Figura 5: ejecución de la opción *Volteo horizontal*.

- ***Volteo vertical***: voltea la imagen en vertical. Cada píxel de la imagen resultante toma el valor del píxel situado en la fila opuesta en la imagen origen:

$$imgD[f, c]=imgO[480-f, c]$$

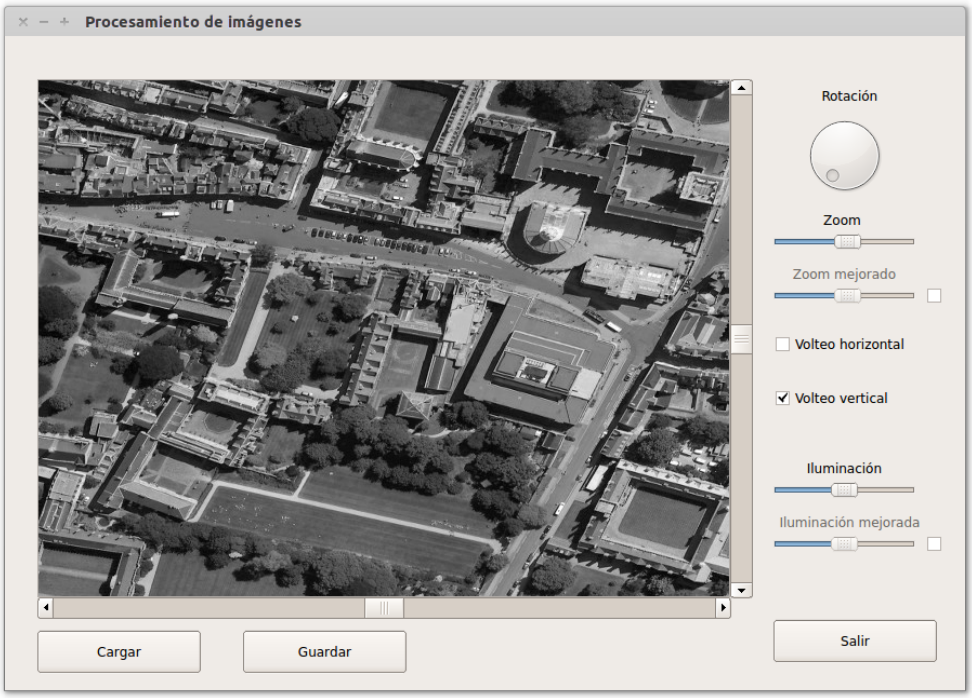

Figura 6: ejecución de la opción *Volteo vertical*.

- ***Iluminación***: transforma los niveles de gris de la imagen para iluminarla u oscurecerla según indique el usuario a través de la barra de desplazamiento situada debajo del nombre de la opción. Para iluminar la imagen, el usuario debe mover la barra de desplazamiento hacia la derecha. Para oscurecerla, hacia la izquierda.
  La transformación que se aplica en ambos casos consiste en saturar a blanco o a negro los niveles de gris situados por encima o por debajo de un nivel de gris indicado por el usuario. Así, para iluminar la imagen, suponiendo que el valor de saturación indicado por el usuario es *gW*, cada nivel de gris *g* de la imagen se transforma en un nivel de gris *gn* a partir de la siguiente relación:

$$\begin{aligned} Sig \geq gW \rightarrow gn = 255 \\ Sig < gW \rightarrow gn = \frac{g \times 255}{gW} \end{aligned} \qquad (1)$$

  Asimismo, el proceso para oscurecer la imagen aplica la siguiente transformación sobre cada nivel de gris *g* partiendo de un nivel de saturación *gB*:

$$\begin{aligned} Sig \leq gB \rightarrow gn = 0 \\ Sig > gB \rightarrow gn = \frac{(g - gB) \times 255}{255 - gB} \end{aligned} \qquad (2)$$

  Para no tener que realizar estas operaciones por cada píxel de la imagen, el proceso se realiza en dos fases. En una primera fase, se genera una tabla conocida como tabla LUT (tabla de consulta - *Look Up Table*). La tabla LUT contiene 256 bytes, uno por cada nivel de gris. Cada índice de la tabla se corresponde con un nivel de gris de la imagen origen. El contenido de cada elemento se refiere al nivel de gris asociado en la imagen destino. Con esta estructura, la tabla permite relacionar cada nivel de gris de la imagen origen con el correspondiente en la imagen destino:

$$\mathrm{LUT}[g]=gn$$

En la segunda fase, se genera la imagen resultante utilizando la tabla anterior. Así, por cada píxel *imgO*[*f*,*c*] de la imagen origen, se obtiene el píxel correspondiente *imgD*[*f*,*c*] de la imagen destino accediendo a la entrada de la tabla LUT que corresponda:

$$g = imgO[f,c]$$
$$imgD[f,c] = LUT[g]$$

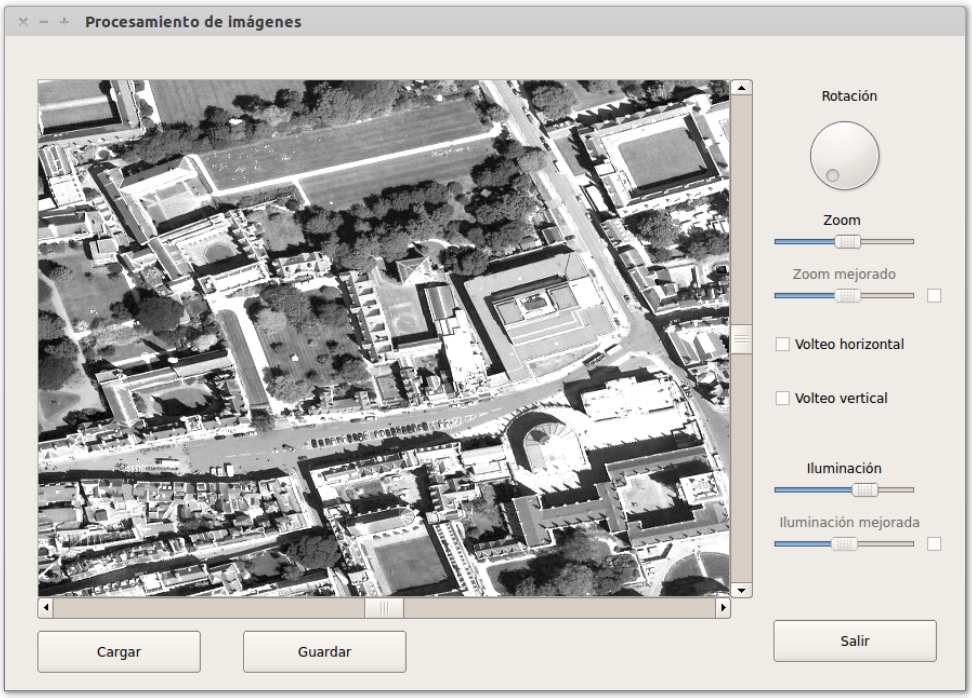

Figura 7: ejecución de la opción Iluminación.

- ***Iluminación mejorada***: permite iluminar u oscurecer la imagen aplicando las funciones seno y coseno, respectivamente, según el valor seleccionado por el usuario a través de la barra de desplazamiento correspondiente.
  Al igual que en la opción anterior, para iluminar la imagen, el usuario debe mover la barra de desplazamiento hacia la derecha. El valor seleccionado (*gW*) permite transformar los niveles de gris de la imagen de la siguiente forma:

$$\begin{aligned} Sig \geq gW \rightarrow gn = g \\ Sig < gW \rightarrow gn = gW \times \sin\left(\frac{\pi \times g}{2 \times gW}\right) \end{aligned} \qquad (3)$$

  De manera similar, la transformación aplicada para oscurecer la imagen a partir de un valor *gB* seleccionado por el usuario se obtiene como:

$$\begin{aligned} Si\ g \geq gB \rightarrow gn = g \\ Si\ g < gB \rightarrow gn = gB \times \left(1 - \cos\left(\frac{\pi \times g}{2 \times gB}\right)\right) \end{aligned} \qquad (4)$$

  Como en la operación anterior, la transformación sobre los píxeles de la imagen se lleva a cabo construyendo una tabla LUT a partir de las relaciones anteriores y aplicando dicha tabla sobre cada píxel de la imagen.

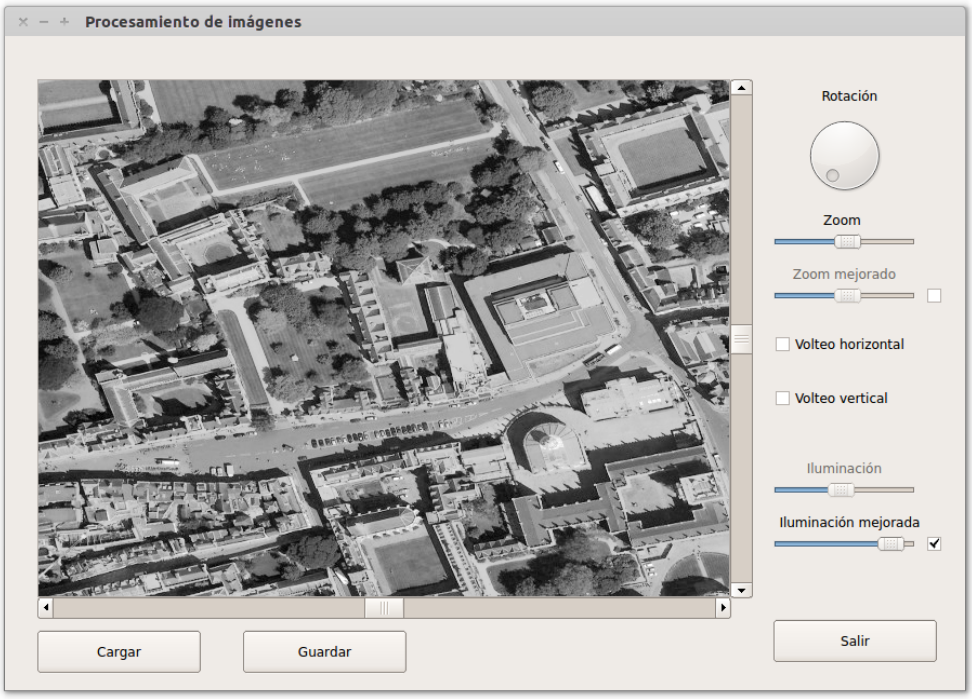

Figura 7: ejecución de la opción Iluminación mejorada.

Componentes de la aplicación

El código fuente de la aplicación está formado por 5 ficheros: *main.cpp*, *pracoc.cpp*, *pracoc.h*, *imageprocess.cpp* e *imageprocess.h*. Además, se incluyen un formulario de QT (*mainForm.ui*) y el fichero de descripción del proyecto (*pracaoc.pro*). 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 la definición de la interfaz de usuario incluida en el programa, así como de las variables que permiten almacenar la información de las imágenes origen y destino utilizadas en las opciones de procesamiento de la aplicación. Estas variables son *imgO, imgD, imgAux e imgDC*. Las tres primeras están definidas como arrays de tipo uchar (unsigned char) de 640x480 elementos. La última mantiene la misma definición de tipo, pero con un tamaño de 800x800. Cada uno de los elementos de estos arrays almacena un posible nivel de gris comprendido en el rango entre 0 y 255. La imagen dentro de cada uno de estos arrays se encuentra almacenada por filas. Esto implica que el acceso a un determinado píxel situado en una fila *f* y una columna *c* de imagen se realiza a través de la posición del array *f*640+c* para las 3 primeras imágenes y de la posición *f*800+c* en la última imagen.
- *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 incluidas 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. Estas funciones contienen una implementación vacía. El objetivo de esta práctica es completarlas para que el funcionamiento de la aplicación sea el descrito anteriormente.

Como ejemplo de implementación de las funciones que deberán ser completadas, se muestra a continuación el código de una función que permite copiar una imagen origen en una imagen destino, ambas con resolución 640x480.

```
void copiar(uchar * imgO, uchar * imgD)
{
	asm volatile(
		"movl %0, %%esi \n\t"
		"movl %1, %%edi \n\t"
		"movl $640*480, %%ecx \n\t"
	"bcopia: \n\t"
		"movb (%%esi), %%al \n\t"
		"movb %%al, (%%edi) \n\t"
		"incl %%esi \n\t"
		"incl %%edi \n\t"
	"loop bcopia \n\t"
		:
		: "m" (imgO), "m" (imgD)
		: "%eax", "%ecx", "%esi", "%edi", "memory"

	);
}
```

La 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, *esi* y *edi*, para su posterior direccionamiento. Así, a través del registro *esi* podremos acceder a cada uno de los píxeles de *imgO* y, mediante el registro *edi*, 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 (640x480). Este bucle se lleva a cabo mediante la instrucción *loop*, por lo que, lo siguiente que hace el procedimiento es inicializar el registro *ecx* con el total de iteraciones.
- Una vez inicializado *ecx*, 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*. El incremento se realiza en una sola posición puesto que cada píxel ocupa 1 byte.

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, todos los procedimientos del módulo "imageprocess.cpp". Estos procedimientos están asociados con las 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::**rotar**(uchar * **imgO**, uchar * **imgD**, float **angle**)*: rota la imagen almacenada en *imgO* un ángulo expresado en radianes, especificado por el parámetro *angle,* y almacena el resultado en *imgD*. La imagen origen tiene un tamaño de 640x480 píxeles. La imagen destino cuenta con una resolución de 800x800.
  Se proponen dos alternativas para implementar este procedimiento. La primera de ellas realizaría los cálculos de transformación de coordenadas operando con valores expresados en punto flotante a través de instrucciones de la FPU. Para ello, se incluye en el procedimiento un bloque de código que permite truncar la parte decimal del resultado una vez realizado el cálculo y extraída la coordenada transformada a un dato de tipo entero. En el procedimiento se indica, entre comentarios, donde debe insertarse el nuevo código para que todo funcione correctamente.
  La segunda alternativa, realizaría la transformación de coordenadas mediante operaciones sobre valores enteros. Para ello, se han incluido las variables locales *sin1000* y *cos1000*, que contendrán, respectivamente, el seno y el coseno del ángulo de rotación multiplicado por 1000. Ambas variables han sido incluidas como operandos del bloque de código en ensamblador. Utilizando esta segunda opción, la transformación de coordendas se calcularía de la siguiente manera:

  $$fO = (sin1000*cD + cos1000*fD)/1000$$
  $$cO = (cos1000*cD - sin1000*fD)/1000$$

  Independientemente de la alternativa escogida, si las coordenadas obtenidas se encuentran situadas fuera de la imagen origen, se le asignará valor 0 (negro) al píxel correspondiente de la imagen destino.

- *void imageprocess::**zoom**(uchar * **imgO**, uchar * **imgD**, float **s**, int **dx**, int **dy**)*: amplía o reduce la imagen almacenada en *imgO* según el factor de escala especificado por *s* y almacena el resultado en *imgD*. Supone además un desplazamiento expresado por los parámetros *dx* y *dy* en la transformación de coordenadas, que afectaría al cálculo de la siguiente manera:

  $$fO = (fD+dy)/s \qquad cO = (cD+dx)/s$$

  Se considera que la imagen origen tiene un tamaño de 800x800 píxeles y la imagen destino de 640x480. Así, si la transformación de coordenadas da como resultado una posición situada fuera de la imagen origen, se le asignará valor 0 al píxel correspondiente de la imagen destino.

Al igual que para el procedimiento anterior, se proponen dos formas de resolver el procedimiento. La primera resolvería las expresiones anteriores utilizando instrucciones de la FPU y operando, por lo tanto, sobre valores en punto flotante. La segunda alternativa realizaría la transformación de coordenadas mediante instrucciones aritméticas sobre enteros. Para implementar esta segunda opción, se han definido las variables locales *sInt* y *ampliar*. Ambas se han incluido como operandos del bloque de código en ensamblador. La variable *ampliar* toma valor 1 cuando el factor de escala indica que la imagen debe ser ampliada y 0 en caso contrario. La variable *sInt* se corresponde con el valor por el que hay que escalar las coordenadas en formato entero. En el caso de la ampliación, el contenido de esta variable se aplicaría a la transformación de la misma manera que si se operara con valores en punto flotante:

$$fO = (fD+dy)/sInt \qquad cO = (cD+dx)/sInt$$

En el caso de la reducción la transformación a utilizar sería la siguiente:

$$fO = (fD+dy)*sInt \qquad cO = (cD+dx)*sInt$$

- *void imageprocess::**zoomMejorado**(uchar * **imgO**, uchar * **imgD**, float **s**, int **dx**, int **dy**)*: realiza la misma operación que el procedimiento anterior, pero aplica una interpolación bilineal en la obtención de cada píxel en el caso de la ampliación. El proceso de interpolación bilineal se describe en el pseudocódigo que se proporciona junto con esta documentación. La reducción de imagen se resuelve de la misma manera que en el procedimiento anterior, por lo que, se ha incluido una llamada a dicho procedimiento en ese caso. Así, la implementación del bloque de código en ensamblador sólo debe contemplar el caso de la ampliación. Se ha definido la variable local *sInt*, incluida como operando del bloque de código ensamblador, para que contenga el factor de escala con formato entero. Las operaciones a realizar para calcular la transformación de coordenadas pueden llevarse a cabo a partir de instrucciones aritméticas sobre valores enteros utilizando el contenido de *sInt*.
- *void imageprocess::**volteoHorizontal**(uchar * **imgO**, uchar * **imgD**)*: voltea, en horizontal, la imagen indicada por *imgO* y almacena el resultado en *imgD*. Ambas imágenes tienen un tamaño de 640x480 píxeles.
- *void imageprocess::**volteoVertical**(uchar * **imgO**, uchar * **imgD**)*: voltea, en vertical, la imagen indicada por *imgO* y almacena el resultado en *imgD*. Ambas imágenes tienen un tamaño de 640x480 píxeles.
- *void imageprocess::**iluminarLUT**(uchar * **tablaLUT**, uchar **gW**)*: construye una tabla LUT, que almacena en el parámetro *tablaLUT*, considerando la transformación de niveles de gris de la ecuación 1 con el valor de *gW* que se indica como segundo parámetro.
- *void imageprocess::**oscurecerLUT**(uchar * **tablaLUT**, uchar **gB**)*: construye una tabla LUT, que almacena en el parámetro *tablaLUT*, considerando la transformación de niveles de gris de la ecuación 2 con el valor de *gB* que se indica como segundo parámetro.
- *void imageprocess::**iluminarLUTMejorado**(uchar * **tablaLUT**, uchar **gW**)*: construye una tabla LUT, que almacena en el parámetro *tablaLUT*, considerando la transformación de niveles de gris de la ecuación 3 con el valor de *gW* que se indica como segundo parámetro. La implementación de este procedimiento conlleva usar la FPU para resolver operaciones en punto flotante.
- *void imageprocess::**oscurecerLUTMejorado**(uchar * **tablaLUT**, uchar **gB**)*: construye una tabla LUT, que almacena en el parámetro *tablaLUT*, considerando la transformación de niveles de gris de la ecuación 4 con el valor de *gB* que se indica como segundo parámetro. La implementación de este procedimiento conlleva usar la FPU para resolver operaciones en punto flotante.
- *void imageprocess::**aplicarTablaLUT**(uchar * tablaLUT, uchar * imgO, uchar * imgD)*: genera una imagen, que devuelve en *imgD*, aplicando la tabla LUT almacenada en el parámetro *tablaLUT* a la imagen indicada en *imgO*. Es invocado cuando el usuario ejecuta las opciones relacionadas con cambios de iluminación para completar la

transformación de la imagen tras haber obtenido la tabla LUT correspondiente por uno de los cuatro procedimientos anteriores. Tanto la imagen origen como la imagen destino cuentan con un tamaño de 640x480 píxeles.

La implementación de los procedimientos *zoomMejorado, iluminarLUTMejorado* y *oscurecerLUTMejorado* 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: La entrega se realizará en enero de 2016. La fecha concreta se comunicará una vez que se publique el calendario de exámenes de la convocatoria de enero.
Entrega: la entrega se realizará a través de la subida al aula virtual de la asignatura de un archivo .zip que contenga el proyecto completo.