PRÁCTICA 10

Procesamiento de imágenes

Parte 2

Descripción de la práctica

El objetivo de esta práctica es realizar una aplicación que amplíe la realizada en las prácticas 8 y 9, introduciendo nuevas funcionalidades relativas al procesamiento imágenes. Concretamente, deberá incluir las siguientes nuevas funcionalidades:

- Modificación del contraste
- Operador cuadrático
- Rotación y escalado de imágenes

El aspecto visual de la aplicación será el mostrado en la Figura 1. El menú incorporará en su opción "Imagen", además de lo incluido en la practica 9, los ítems "AffineTransformOp", "LookupOp", "BandCombineOp" y "ColorConvertOp". En la parte inferior, además de los ya incluido en la práctica 9, se incorporará un área con tres botones asociados a tres tipos de contraste (normal, iluminado y oscurecido), un botón para el operador cuadrático, otra área asociada a la rotación, con un deslizador para girar la imagen y tres botones para rotaciones fijas, y un área de escalado con dos botones (incrementar y reducir).

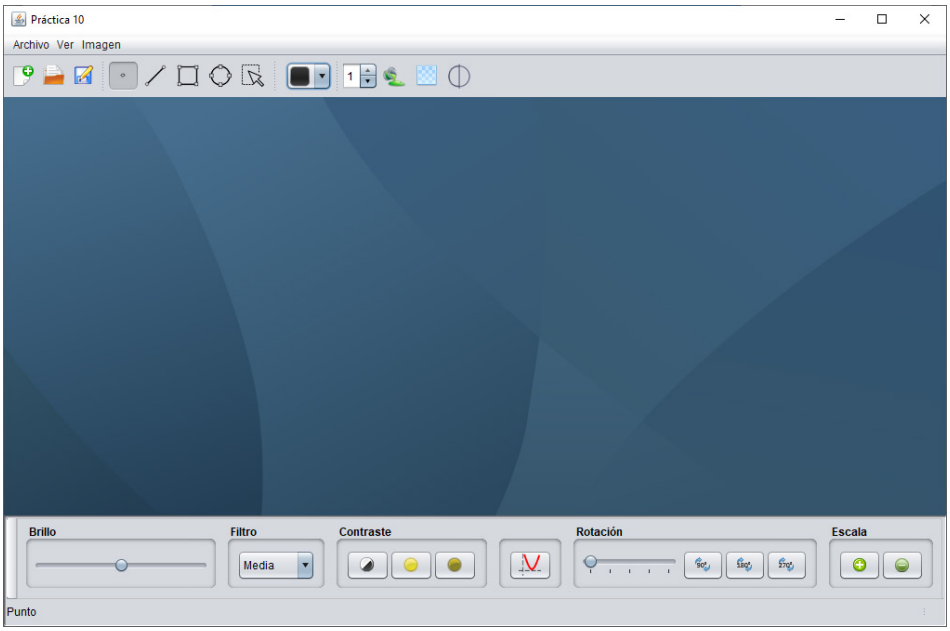

Figura 1: Aspecto de la aplicación

Pruebas iniciales

En una primera parte, probaremos los operadores "`AffineTransformOp`" y "`LookupOp`" usando parámetros fijos (i.e., sin interacción del usuario para definir sus valores). Para ello, incluiremos las correspondientes opciones en el menú "Imagen" cuya selección implicará la aplicación de la correspondiente operación en la imagen seleccionada.

En estos casos, se probará el código mostrado en las transparencias de teoría (que incluiremos en el manejador del evento "acción" asociado al menú), teniendo en cuenta que dicha operación se aplicará sobre la imagen mostrada en la ventana activa. Al igual que vimos en la

práctica 9, esto supondrá que, al código visto en teoría, habrá que añadirle las sentencias para el acceso y actualización de la imagen; por ejemplo, para el caso de la operación "AffineTransformOp" y el escalado:

```
private void menuAffineTransformOpActionPerformed(ActionEvent evt) {
  VentanaInterna vi = (VentanaInterna) (escritorio.getSelectedFrame());
  if (vi != null) {
    BufferedImage img = vi.getLienzo().getImage();
    if(img!=null){
      try{
        AffineTransform at = AffineTransform.getScaleInstance(1.5,1.5);
        AffineTransformOp atop = new AffineTransformOp(at,null);
        BufferedImage imgdest = atop.filter(img, null);
        vi.getLienzo().setImage(imgdest);
        vi.getLienzo().repaint();
      } catch(IllegalArgumentException e){
         System.err.println(e.getLocalizedMessage());
      }
    }
  }
}
```

Obsérvese que en el código anterior se ha optado por pasarle `null` como segundo parámetro al método `filter` , con lo cual se crea una nueva imagen salida; esto implica que sea necesario actualizar la imagen del lienzo de forma explícita. Por otro lado, en la creación del operador no lo indicamos ningún método de interpolación específico (al usar `null`, emplea el método de "vecino más cercano" por defecto); si queremos mejorar el resultado usando otra interpolación, podría indicarse mediante las variables estáticas que ofrece la clase `AffineTransformOp` (por ejemplo, `AffineTransformOp .TYPE_BILINEAR`. para interpolación bilineal[1]).

Para el caso de la operación "LookupOp", el código será similar al anterior pero cambiando el operador; por ejemplo, para aplicar una operación de negativo (véanse transparencias) el código sería:

```
byte funcionT[] = new byte[256];
for (int x=0; x<256; x++)
  funcionT[x] = (byte)(255-x);  // Negativo
LookupTable tabla = new ByteLookupTable(0, funcionT);
LookupOp lop = new LookupOp(tabla, null);
```

En relación al operador `LookupOp`, hay que tener en cuenta que para imágenes tipo "BGR" (p.e., `TYPE_INT_BGR`), si no indicamos imagen de salida en la llamada a `filter` (i.e, dejamos a `null` el segundo parámetro), el operador genera una imagen de salida que intercambia[2] los canales B y R. Para abordar este problema, podríamos optar por convertir la imagen fuente a tipo `TYPE_INT_ARGB` para asegurar compatibilidad[3] o pasar en la llamada a `filter` una imagen destino compatible con la fuente (como caso particular, se podría pasar la misma imagen fuente, que en ese caso sería modificada).

Variación del contraste

En este apartado modificaremos el contraste de la imagen aplicando el operador `LookupOp`. Para ello, incluiremos tres botones correspondientes a tres posibles situaciones:

- Contraste "normal", para imágenes en las que la luminosidad esté equilibrada. En este caso, se usan funciones tipo S

[1] En este caso, además, emplearía transparencia en la imagen resultado (esto es relevante, por ejemplo, en el caso de rotaciones)

[2] *Bug* no documentado

[3] Por ejemplo, el siguiente código haría la conversión usando un método de la clase `sm.image.ImageTool`:
`img = ImageTools.convertImageType(img, BufferedImage.TYPE_INT_ARGB);`

- Contraste con iluminación, para imágenes oscuras. En este caso, se usan funciones tipo logaritmo (si es muy oscura), funciones raíz (con cuyo parámetro podemos determinar el grado de iluminación) o correcciones gamma (con gamma mayor que 1)
- Contraste con oscurecimiento, para imágenes sobre-iluminadas. En este caso, se usan funciones potencia (con cuyo parámetro podemos determinar el grado de oscurecimiento) o correcciones gamma (con gamma entre 0 y 1).

Cada una de las operaciones anteriores está asociada a una determinada función (véanse transparencias de teoría). En la clase `LookupTableProducer` del paquete `sm.image` se definen métodos estáticos para crear objetos `LookupTable` correspondientes a funciones estándar (negativo, función-S, potencia, raíz, corrección gamma, etc.); por ejemplo, para crear la función S pasándole los correspondientes parámetros usaríamos el siguiente código[4]:

```
LookupTable lt = LookupTableProducer.sFuction(128.0,3.0);
```

Por otro lado, esta clase define un método `createLookupTable` que devuelve objetos `LookupTable` correspondientes a las funciones clásicas anteriores pero usando parámetros predefinidos. Por ejemplo, el siguiente código crearía un objeto `LookupTable` por defecto asociado a la función-S:

```
LookupTable lt;
lt=LookupTableProducer.createLookupTable(LookupTableProducer.TYPE_SFUNCION);
```

Para los ejemplos que se proponen en este ejercicio, bastaría usar los parámetros por defecto que ofrece el método `createLookupTable`. Por ejemplo, para el caso del contraste normal[5]:

```
private void bContrasteActionPerformed(ActionEvent evt) {
  VentanaInterna vi = (VentanaInterna) (escritorio.getSelectedFrame());
  if (vi != null) {
    BufferedImage img = vi.getLienzo().getImage();
    if(img!=null){
      try{
        int type = LookupTableProducer.TYPE_SFUNCION;
        LookupTable lt = LookupTableProducer.createLookupTable(type);
        LookupOp lop = new LookupOp(lt, null);
        // Imagen origen y destino iguales
        lop.filter( img , img);
        vi.getLienzo().repaint();
      } catch(Exception e){
          System.err.println(e.getLocalizedMessage());
      }
    }
  }
}
```

Nótese que en ejemplo anterior se usa como imagen destino la propia imagen fuente.

---

[4] Por si resulta útil de cara a implementar otras funciones propias, el código en el paquete `sm.image` correspondiente a la función S es el siguiente:

```
public static LookupTable sFuction(double m, double e){
  double Max = (1.0/(1.0+Math.pow(m/255.0,e)));
  double K = 255.0/Max;
  byte lt[] = new byte[256];
  lt[0]=0;
  for (int l=1; l<256; l++){
    lt[l] = (byte)(K*(1.0/(1.0+Math.pow(m/(float)l,e))));
  }
  ByteLookupTable slt = new ByteLookupTable(0,lt);
  return slt;
}
```

[5] En caso de que se quisiera ofrecer al usuario la posibilidad de modificar los parámetros del contraste (p.e., mediante un deslizador similar al caso del brillo), habría que usar las funciones directamente (sin llamar al método `createLookupTable`) pasándole como parámetros los definidos por el usuario.

Función cuadrática

A continuación, aplicaremos la función $f(x) = \frac{1}{100}(x-m)^2$, con $0 \leq m \leq 255$ un parámetro que indica dónde se hace cero la función[6]. La función anterior no está implementada en el paquete `sm.image`, por lo que tendrá que ser programada por el estudiante:

```
public LookupTable cuadratica(double m){
  double K = ¿?; // Cte de normalización
  // Código implementado f(x)
  // TO·DO
}
```

Recordemos que debemos normalizar el resultado de la función para garantizar que la salida esté entre 0 y 255. En este caso, al contrario de los ejemplos vistos en clase, no tenemos una función monótona creciente, por lo que el valor máximo (en el intervalo [0,255]) lo alcanzará en el 0 o en el 255 dependiendo del parámetro m (concretamente, será en el 0 si $m \geq 128$ y en el 255 en caso contrario).

Para probar la función anterior, incorporar un nuevo botón asociado a esta operación y testearla para $m = 128$. Indicar, por último, que la constante $1/100$ determina cómo de "pronunciada" es la función: cuanto mayor sea el denominador, más suave será. Esta constante podría ser otro parámetro de la función, si bien se ha optado por fijarla para este ejercicio (se invita al estudiante a modificar ese valor para ver el efecto).

Rotación

En este apartado rotaremos la imagen ofreciendo dos posibilidades:

- Giro libre, donde el usuario podrá girar 360° la imagen usando un deslizador (con rango [0,360]).
- Rotaciones predeterminadas de 90°, 180° y 270°.

Para ello haremos uso del operador "`AffineTransformOp`"; en el primer caso, se usará como grado el definido por el usuario en el deslizador, en el segundo serán valores fijos. En ambos casos, la rotación tendrá que hacerse poniendo como eje de rotación el centro de la imagen[7,8]:

```
double r = Math.toRadians(180);
Point c = new Point(imgSource.getWidth()/2, imgSource.getHeight()/2);
AffineTransform at = AffineTransform.getRotateInstance(r,p.x,p.y);
AffineTransformOp atop;
atop = new AffineTransformOp(at,AffineTransformOp.TYPE_BILINEAR);
BufferedImage imgdest = atop.filter(imgSource, null);
```

En el caso del deslizador, al igual que ocurrió con el brillo en la práctica 9, tendremos que tener en cuenta que la imagen sobre la que se aplica la operación ha de ser la original, y no la obtenida temporalmente durante el proceso: cada nuevo valor del deslizador[9], implicará calcular la imagen (temporal) resultado de aplicar ese valor de brillo sobre la imagen original (la imagen resultado se irá mostrando en la ventana mientras el usuario mueve el deslizador). Para abordar

---

[6] Se aconseja usar herramientas de visualización de funciones (como, por ejemplo, Mahtway) para así entender gráficamente el comportamiento de la función y la influencia de los parámetros.

[7] Tras aplicar el `AffineTransformOp`, la imagen resultado tendrá activo el canal alfa. Esto hay que tenerlo en cuenta, por ejemplo, en el operador `RescaleOp` (véase práctica 9 y su pie de página 5).

[8] Para que le efecto de la rotación fuese el esperado, tendríamos que concatenar la rotación con una traslación (el uso de únicamente la rotación "corta las esquinas"); el valor de dicha traslación dependerá del ángulo de rotación. Para el caso particular de 90° y 270°, el operador devuelve una imagen cuadrada que deja transparente la zona sobrante. Para esta práctica haremos las pruebas usando solo la transformación de rotación (sin la traslación), dejando para trabajos futuros la mejora que incluya la traslación.

[9] Notificado mediante el evento `stateChange`

este aspecto, seguiremos la propuesta de la práctica 9: usaremos la variable de tipo `BufferedImage` definida en la ventana principal y asociada a la imagen fuente original, de forma que cuando se inicie la operación (el deslizador gana el foco) se le asignará como valor la imagen que haya en el lienzo activo; posteriormente, durante el movimiento del deslizador, la operación se aplicará a la imagen fuente almacenada (y el resultado se irá mostrando en el lienzo); por último, cuando la operación acabe (el deslizador pierde el foco), anularemos la imagen fuente y pondremos a cero el valor del deslizador.

Indicar que, en este caso, y al contrario que hicimos para el brillo en la práctica 9, no es necesario crear una copia de la imagen original cuando empieza la operación; el motivo es que en la llamada al `filter` usamos la opción de `null` como segundo parámetro, por lo que internamente se creará una imagen nueva cada vez que apliquemos el operador (en el caso del brillo se usaba la misma imagen como fuente y destino, de ahí que fuese necesaria la copia inicial).

Escalado

En este apartado escalaremos la imagen mediante el operador "`AffineTransformOp`". Para ello se usarán dos botones: un para aumentar el tamaño de la imagen y otro para reducirlo. En este caso fijaremos el factor de escala (por ejemplo, 1.25 para aumentar y 0.75 para reducir).

El reto final…

Implementar un operador "LookupOp" que permita el siguiente efecto: aquellos valores[10] que superen un umbral T (por ejemplo, T=128) mantendrán su valor original, mientras que aquellos inferiores a dicho umbral deberán aumentar su luminosidad. Es decir, al aplicar este efecto "las zonas claras se mantendrán exactamente iguales" mientras que "las zonas oscuras se iluminarán".

Para solucionar este reto has de pensar en qué función habría que aplicar: hay varias opciones, así que intenta primero "esbozar" la gráfica sobre papel (si usas alguna herramienta de representación gráfica de funciones, mucho mejor) y luego trata de crear el `LookupTable` asociado. ¡Tú puedes! ;)

Para trabajar en casa…

Una vez realizada la práctica, se proponen las siguientes mejoras:

- Tras realizar la rotación, guardar la imagen. ¿Se guarda correctamente? Si no es así, posiblemente sea porque se esté almacenando en un formato inadecuado (como JPG): la imagen generada en una rotación tiene transparencia, por lo que hay que usar un formato que permita canal alfa (por ejemplo, PNG). Si se implementó la mejora propuesta en la práctica 8 (usar filtros en el diálogo guardar y obtener el formato a partir de la extensión del fichero), bastará elegir el formato adecuado; si no se hizo, ahora es un buen momento para hacerlo… ;)
- Incluir la operación "negativo".
- Incluir una opción "duplicar" que cree una nueva ventana interna con una copia de la imagen que había en la ventana activa.
- Incluir un deslizador para modificar de forma interactiva el parámetro "m" de la función cuadrática desarrollada en esta práctica.
- Definir nuevas operaciones "lookup" propias. Se aconseja utilizar herramientas de representación gráfica de funciones para analizar cuál sería el comportamiento esperado de las funciones propuestas (y cómo influyen los posibles parámetros, si los hubiera).

[10] En la misma línea de ejercicios anteriores, se aplicará la misma función en cada canal RGB.