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 sinusoidal
- 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 sinusoidal, 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).

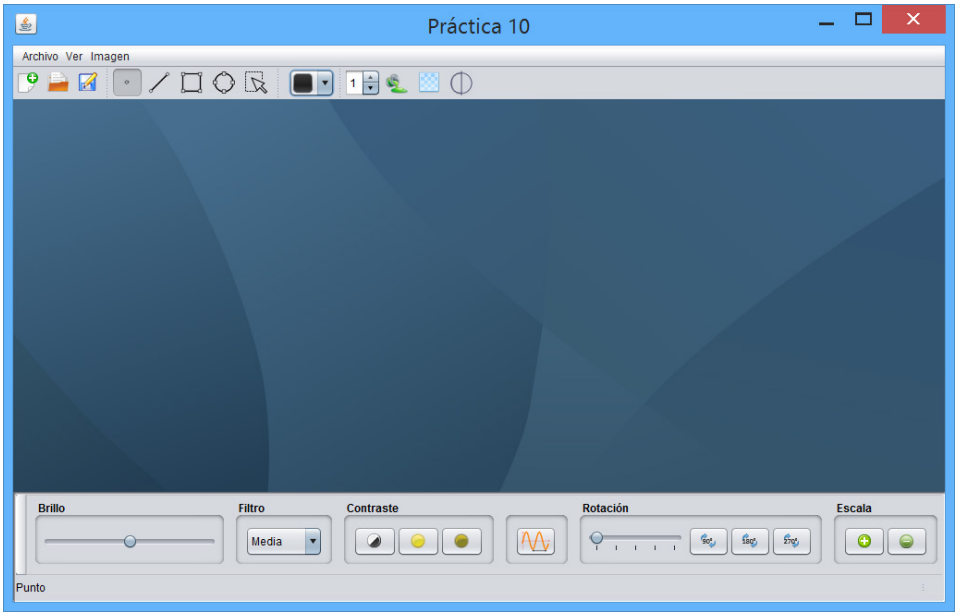


Figura 1: Aspecto de la aplicación

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 (véase ejemplo de práctica 9).

A la hora de usar el operador *LookupOp*, la clase *LookupTableProducer* del paquete *sm.image* contiene 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[1]:

```
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);
```

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
- 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).

Todas las funciones anteriores se encuentran implementadas en la clase *LookupTableProducer* del paquete *sm.image*. Para estos ejemplos, bastaría usar los parámetros por defecto que ofrece el método *createLookupTable*. Por ejemplo, para el caso del contraste normal[4]:

---

[1] 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;
}
```

[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);

[4] 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.

```
private void bContrasteActionPerformed(ActionEvent evt) {
  VentanaInterna vi = (VentanaInterna) (escritorio.getSelectedFrame());
  if (vi != null) {
    BufferedImage imgSource = vi.getLienzo().getImage();
    if(imgSource!=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( imgSource , imgSource);
        vi.repaint();
      } catch(Exception e){
          System.err.println(e.getLocalizedMessage());
      }
    }
  }
}
```

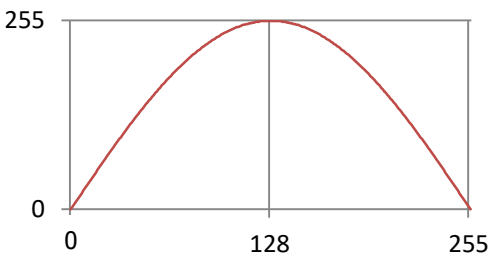
Nótese que en ejemplo anterior se usa como imagen destino la propia imagen fuente.

Función seno

A continuación, aplicaremos la función $f(x) = |\sin(w \cdot x)|$, con w la velocidad angular y donde $|\cdot|$ representa el valor absoluto. La función anterior no está implementada en el paquete `sm.image`, por lo que tendrá que ser programada por el estudiante:

```
public LookupTable seno(double w){
  double K = 255.0; // Cte de normalización
  // Código implementado f(x)=|sin(wx)|
  // TODO
}
```

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, pero sabemos que el máximo valor devuelto por la función seno es 1.0 cuando $x = \pi/2$, por lo que tendremos que $K = 255.0/|\sin(\pi/2)| = 255.0$.



Para probar la función anterior, incorporar una nuevo botón asociado a esta operación y testearla para $w = 180.0/255.0$ (en este caso, la función será como la mostrada en la figura). Si aumentamos el valor de w tendremos una función seno (positiva) de mayor frecuencia (y habrá más de un ciclo en nuestra gráfica); si disminuimos, la frecuencia será menor (y no llegará a completarse un ciclo)

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[5,6]:

```
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);
```

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).

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.

[5] 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).

[6] Para el caso de 90º y 270º, el operador devuelve una imagen cuadrada (dejando transparente la zona sobrante); si quisiéramos que tuviera las dimensiones correctas, tendríamos que crear nosotros la imagen y pasarla como segundo parámetro en el método *filter*.