UNIVERSIDAD DE GRANADA

iFilter Image

Procesamiento Digital de Señales

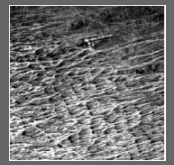
Realizado por Marino Fajardo, Sofía Real y Alejandro Coca

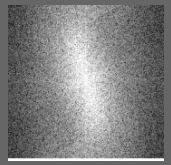
Introducción

- Propósito de la aplicación
- Contexto
- Descripción

Fundamentos del Procesamiento Digital de Señales en el ámbito de imágenes

- Transformada de Fourier





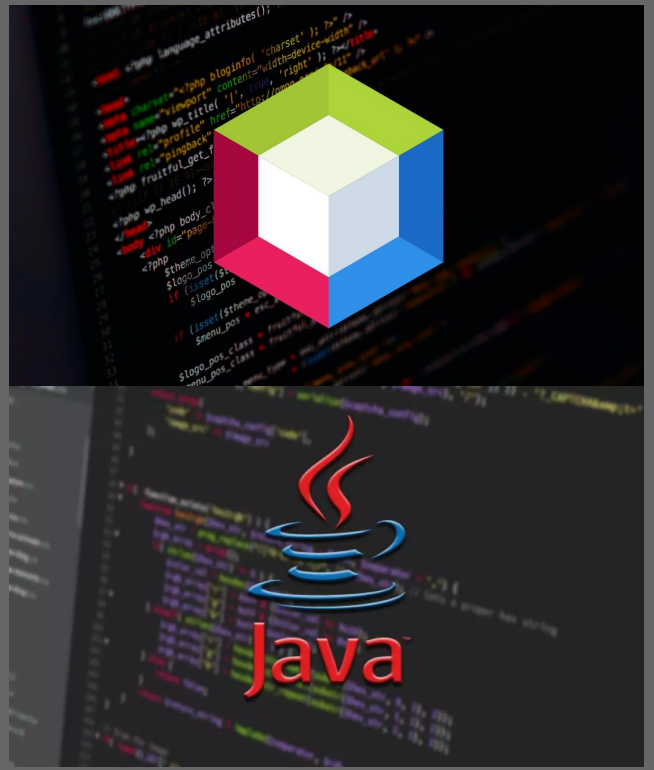
- Funciones convolve

- Filtros de Color

Diseño de la aplicación

- Entorno de desarrollo:
  - Netbeans

- Bibliotecas usadas:
  - java.awt.image



Algoritmos y funcionamiento de los filtros

- Filtros de Escalado de Píxeles (RescaleOp)
- Filtros de Convolución (ConvolveOp)
- Filtros de Transformación de Píxeles (ByteLookupTable)
- Filtros de Transformación de Imágenes (AffineTransform)
- Filtro de Ecualización de Histogramas
- Filtros de Substracción de Color
- Filtro de Posterizado

RescaleOp

- Nos permite realizar operaciones de escala en los píxeles de una imagen.

```
VentanaInterna vi = (VentanaInterna) (Escritorio.getSelectedFrame());
if(vi != null){
    if(imgFuente!=null){
        try{
            RescaleOp rop = new RescaleOp((this.SliderContraste.getValue())/10.f, 0.F, null);
            BufferedImage imgdest = rop.filter(imgFuente, null);
            vi.getLienzo().setImage(imgdest);
            vi.getLienzo().repaint();
        }catch(IllegalArgumentException e){
            System.err.println(e.getLocalizedMessage());
        }
    }

}
```

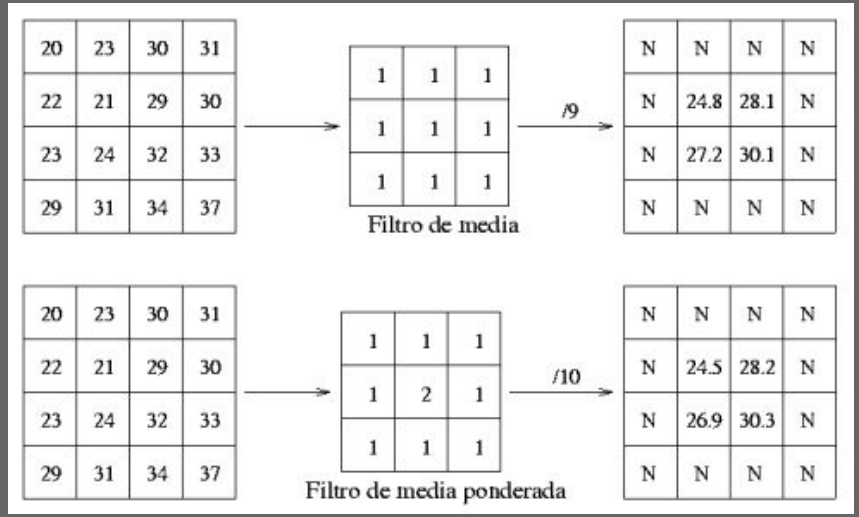
ConvolveOp

- Combinación de dos funciones para representar la forma en la que estas se superponen la una con la otra.
- Tipos de Filtro:
  - Paso Bajo
  - Paso Alto
  - Filtros Direccionales
  - Filtros de Detección de Bordes

```
ConvolveOp cop = new ConvolveOp(k,ConvolveOp.EDGE_NO_OP,null);
BufferedImage imgdest = cop.filter(img, null);
```

ConvolveOp: Paso Bajo

- Suavizar la imagen o eliminar ruido.



ConvolveOp: Paso Alto

- Eliminar las zonas de bajas frecuencias.

-1	-1	-1
-1	8	-1
-1	-1	-1

DIV 9

0	1	0
1	-4	1
0	1	0

DIV=1

0	-1	0
-1	5	-1
0	-1	0

DIV=1

ConvolveOp: Direccionales

- Detección de estructuras que resalta el contraste entre los píxeles a ambos lados de la estructura.

1	1	1
1	-2	1
-1	-1	-1

DIV=1

-1	1	1
-1	-2	1
-1	1	1

DIV=1

ConvolveOp: Detección de Bordes

- Detección de bordes de Sobel:

-1.0	0	1.0
-2.0	0	2.0
-1.0	0	1.0

-1.0	-2.0	-1.0
0	0	0
1	2	1

ByteLookupTable

- Es una estructura de datos que se usa para asignar valores de entrada a valores de salida específicos.
- Podemos crear operaciones para aumentar el contraste (sfuncion), iluminar una imagen (logarítmica) u oscurecer una imagen (potencia).

```
int type = LookupTableProducer.TYPE_SFUNCION;
LookupTable lt = LookupTableProducer.createLookupTable(type);
LookupOp lop = new LookupOp(lt, null);
```

ByteLookupTable: Función Cuadrática

- Aplica una transformación cuadrática, que aumenta o disminuye el contraste de la imagen, resaltando o atenuando la diferencia entre los valores de los píxeles.

```
public ByteLookupTable cuadratica(double m){
    double Max;
    if(m>=128.0){
        Max = (double)((1.0/100)*(Math.pow(0.0-m, 2.0)));
    }else{
        Max = (double)((1.0/100)*(Math.pow(255.0-m, 2.0)));
    }
    double K = 255.0/Max;
    byte lt[] = new byte[256];
    lt[0]=0;
    for (int l=1; l<256; l++){
        lt[l] = (byte)(K*((1.0/100)*(Math.pow(m-(float)l,2.0))));
    }
    ByteLookupTable slt = new ByteLookupTable(0,lt);
    return slt;
}
```

ByteLookupTable: Función Trapezoidal

- Dependiendo de los valores de a y b se pueden lograr diferentes efectos de contraste y realce sobre una imagen.

```
public ByteLookupTable trapezoidal(double a, double b)
    double K = 255.0;
    byte lt[] = new byte[256];
    lt[0]=0;
    for (int l=1; l<256; l++){
        if(l<=0){
            lt[l] = (byte)(K*0);
        }else if((0<l) && (l<a)){
            lt[l] = (byte)(K*(l/a));
        }else if((a<=l)&&(l<=b)){
            lt[l] = (byte)(K*1);
        }else if((b<l)&&(l<255)){
            lt[l] = (byte)(K*((255.0-l)/(255.0-b)));
        }else if(l>=255){
            lt[l] = (byte)(K*0);
        }
    }
    ByteLookupTable slt = new ByteLookupTable(0,lt);
    return slt;
}
```

AffineTransform

- Nos permite hacer transformaciones en una imagen del tipo escala y rotación.

```
double r = Math.toRadians(90);
Point c = new Point(img.getWidth()/2, img.getHeight()/2);
AffineTransform at = AffineTransform.getRotateInstance(r,c.x,c.y);
AffineTransformOp atop;
atop = new AffineTransformOp(at,AffineTransformOp.TYPE_BILINEAR);
BufferedImage imgdest = atop.filter(img, null);
```

```
AffineTransform at = AffineTransform.getScaleInstance(1.25, 1.25);
AffineTransformOp atop;
atop = new AffineTransformOp(at,AffineTransformOp.TYPE_BILINEAR);
BufferedImage imgdest = atop.filter(img, null);
```

Ecualización de Histogramas

- Mejora el contraste de una imagen al redistribuir los niveles de intensidad.

```
// Obtener el color del píxel en coordenadas (x, y)
int rgb = image.getRGB(x, y);

// Obtener los componentes RGB del color
int red = (rgb >> 16) & 0xFF;
int green = (rgb >> 8) & 0xFF;
int blue = rgb & 0xFF;

// Calcular los nuevos componentes RGB aplicando la ecualizació
int newRed = Math.round(cdf[red] * maxValue);
int newGreen = Math.round(cdf[green] * maxValue);
int newBlue = Math.round(cdf[blue] * maxValue);

// Combinar los nuevos componentes RGB en un solo valor entero
int newRGB = (newRed << 16) | (newGreen << 8) | newBlue;

// Establecer el nuevo color en el resultado
result.setRGB(x, y, newRGB);
```

Substracción de Color

- Lleva todos los colores de una imagen a escala de grises conservando únicamente los colores deseados (Rojo, Azul, Verde).

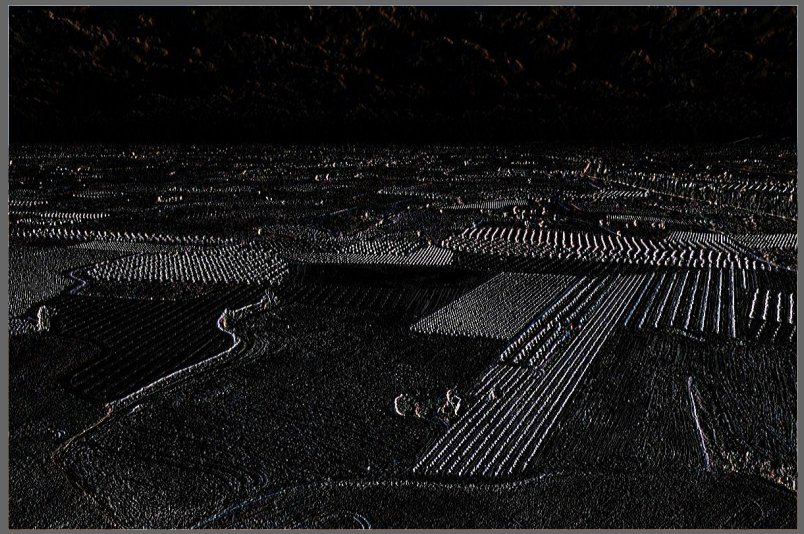
```
srcRaster.getPixel(x, y, pixelComp);
if(pixelComp[0]-pixelComp[1]-pixelComp[2] >= umbral){
    pixelCompDest[0] = pixelComp[0];
    pixelCompDest[1] = pixelComp[1];
    pixelCompDest[2] = pixelComp[2];
}else{
    pixelCompDest[0] = (pixelComp[0] + pixelComp[1] + pixelComp[2])/3;
    pixelCompDest[1] = (pixelComp[0] + pixelComp[1] + pixelComp[2])/3;
    pixelCompDest[2] = (pixelComp[0] + pixelComp[1] + pixelComp[2])/3;
}
destRaster.setPixel(x, y, pixelCompDest);
```

Posterizado

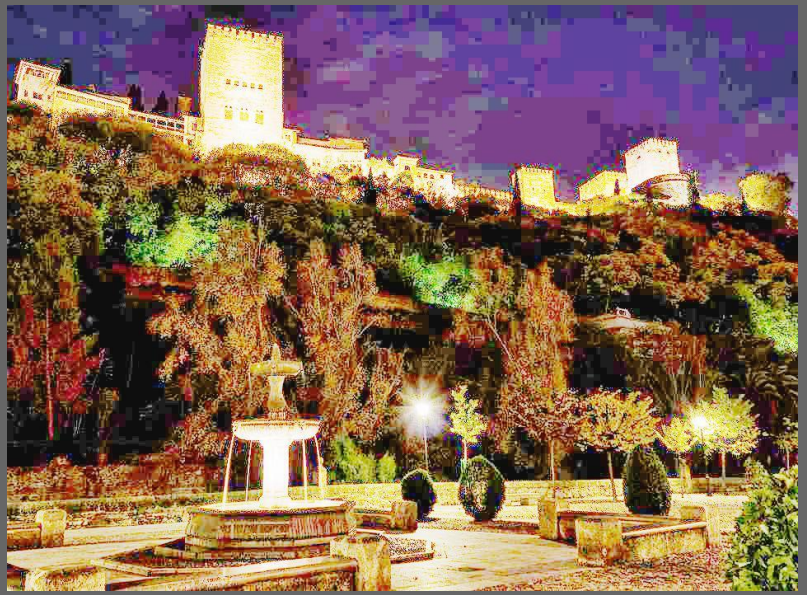
- Reduce el número de niveles de intensidad de una imagen, creando un efecto de imagen con colores sólidos y distintos.

```
float K = 256.f/niveles;
for (int x = 0; x < src.getWidth(); x++) {
    for (int y = 0; y < src.getHeight(); y++) {
        for (int band = 0; band < srcRaster.getNumBands(); band++){
            sample = srcRaster.getSample(x, y, band);
            sample = (int)(K * (int)(sample/K));
            destRaster.setSample(x, y, band, sample);
        }
    }
}
```

Ejemplos de Uso: Detección de Bordes



Ejemplos de Uso: Histograma



Ejemplos de Uso: Posterizado

Demostración de la aplicación

Preguntas

Muchas gracias por su atención