



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
Ingeniería en Inteligencia Artificial



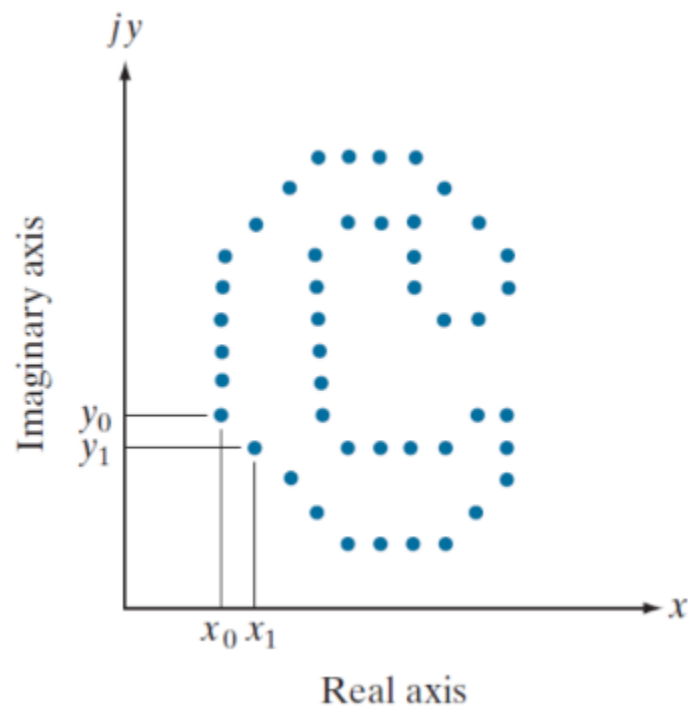
Practica 11: Descriptores de imagen

Nombre del alumno: Torres López Marco Antonio

Nombre del profesor: Saul de la O Torres

Grupo: 5BM1

Unidad de aprendizaje: Visión artificial



Introducción

Los descriptores visuales describen las características visuales de los contenidos dispuestos en imágenes o en vídeos. Describen características elementales tales como la forma, el color, la textura o el movimiento, entre otros.


Propiedades de los descriptores:

1. Unicidad: cada objeto debe tener una única representación.
2. Invariancia frente a transformaciones geométricas, como traslaciones, rotaciones, cambios de escala y reflexiones.
3. Sensibilidad o capacidad para diferenciar objetos casi iguales.
4. Abstracción del detalle o capacidad para representar los rasgos característicos básicos de los objetos y abstraer los detalles.

Existen distintos tipos de descriptores, uno de ellos son los descriptores de Fourier.

Los descriptores de Fourier son coeficientes del espectro de Fourier que se extraen del borde de los objetos para describir su forma. Un píxel del borde tiene al menos un píxel vecino del fondo.

Descriptores de Fourier


$$\text{DFT} \rightarrow a(u) = \sum_{k=0}^{N-1} s(k) e^{-j2\pi uk/N}$$

$$\text{IDFT} \rightarrow s(k) = \frac{1}{N} \sum_{u=0}^{N-1} a(u) e^{j2\pi uk/N}$$

Desarrollo

Paso 1.

Cargaremos nuestra imagen a procesar

```
1 String PATH = "c:/Users/MARCO/Documents/NetBeansProjects/";
2 String filePath = PATH + "circle.png";
3 // # 1. Cargamos la imagen
4 Mat newImage = Imgcodecs.imread(filePath);
```

Paso 2.

Convertiremos nuestra imagen a niveles de grises para poder empezar a procesar la imagen

```
1 // # 2. Convertimos a escala de grises
2 Mat gris = new Mat();
3 Imgproc.cvtColor(newImage.clone(), gris, Imgproc.COLOR_BGR2GRAY);
```

Paso 3.

Aplicaremos suavizado gaussiano para reducir el ruido de la imagen

```
1 // # 3. Aplicar suavizado Gaussiano
2 Mat gauss = aberrarGaussiano(gris, 11);
3 mostrarImagen("suavisado", gauss);
4 System.out.println(gauss.height() + " " + gauss.width() + " " + gauss.type());
```

Paso 4.

Empezaremos a detectar contornos en la imagen con el operador Canny

```
1 // # 4. Detectamos los contornos con Canny
2 Mat canny = new Mat();
3 Imgproc.Canny(gauss, canny, 60, 180);
4 mostrarImagen("canny", canny);
```

Paso 5.

Teniendo los contornos comenzaremos a buscar los contornos con el método de FindCoutours

```
1 // # 5. Buscamos los contornos
2 List<MatOfPoint> contornos = new ArrayList<>();
3 Mat jerarquia = new Mat();
4 Imgproc.findContours(canny.clone(), contornos, jerarquia,
5     Imgproc.RETR_EXTERNAL,
6     Imgproc.CHAIN_APPROX_SIMPLE);
```

```

1
2 // # 6. Dibujar los contornos encontrados
3 Imgproc.drawContours(newImage, contornos, -1, new Scalar(0,0,255), 2);
4 System.out.println("Largo de la imagen: " + canny.width());
5 System.out.println("Ancho de la imagen: " + canny.height());
6 int longitud = canny.width()*canny.height();
7 TDFourier fourier = new TDFourier(longitud);
8 Complejo [] senial = new Complejo(longitud);
9
10 for (int i = 0; i < senial.length; i++)
11     senial[i] = new Complejo(0, 0);
12
13 for (MatOfPoint contorno : contornos) {
14     for (Point punto : contorno.toList()) {
15         int x = (int) punto.x;
16         int y = (int) punto.y;
17         senial[(x*canny.width()+y)] = new Complejo(canny.get(y, x)[0],0.0);
18     }
19 }
20
21 int P = longitud;
22 System.out.println("Cantidad de pixeles:" + P);
23 System.out.println("Numero de contornos encontrados: " + contornos.size());

```

Paso 6.

Dibujaremos los contornos dentro de los puntos detectados. Se agrego al código un ciclo For anidados donde se recorren todos los puntos dentro del valor de los contornos para convertirlos en números complejos para posteriormente obtener sus respectivos descriptores.

Paso 7.

Con los nuevos valores en números complejos se aplica la transformada directa de Fourier y la transformada inversa para obtener sus respectivos descriptores. Al final aplicamos Canny inversa para visualizar el resultado final.

Nota: se intercambiaron las coordenadas x,y del ciclo de CannyInv para que no salga invertida la imagen.

```

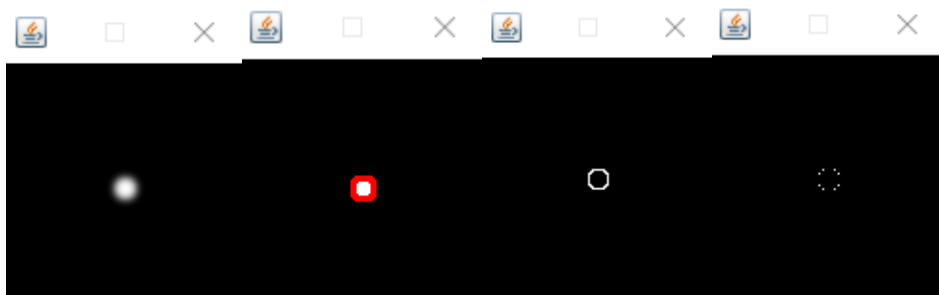
1
2 // directa
3 fourier.transformarDirecto(senial);
4 // inversa
5 fourier.transformarInverso(fourier.getAu(), P);
6 Complejo [] sk = fourier.getSk();
7 int contador = 0;
8 System.out.println(canny.height() + " " + canny.width() + " " + canny.type());
9 Mat cannyInv = new Mat(canny.height(), canny.width(), canny.type());
10 for(int y=0; y<canny.height(); y++) {
11     for(int x=0; x<canny.width(); x++) {
12         byte [] pixel = {
13             (byte)sk[contador].getReal()
14         };
15         cannyInv.put(x, y, pixel);
16         contador++;
17     }
18 }
19 mostrarImagen("cannyInv", cannyInv);
20 }

```

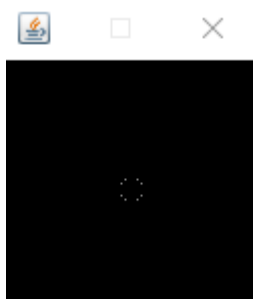
Resultados

Probaremos varias imágenes y con diferentes parámetros dentro de la división de P para observar la cantidad perdida de descriptores de la imagen.

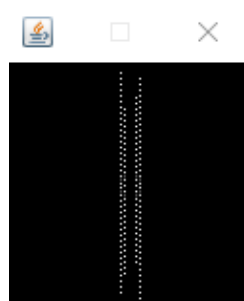
P con el total de longitud ($P=Longitud$):



P con la mitad de la longitud ($P=Longitud/2$):



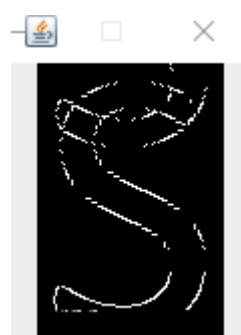
P con una tercera parte de la longitud ($P=Longitud/3$):



Probando con otras imágenes



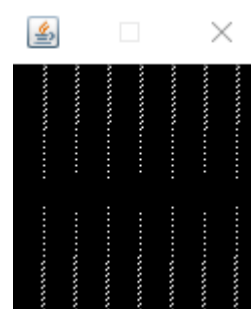
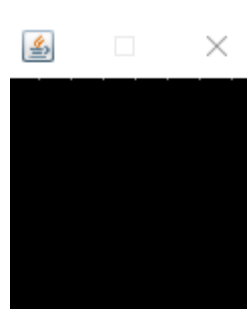
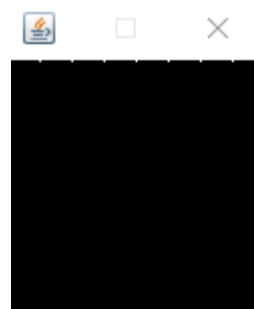
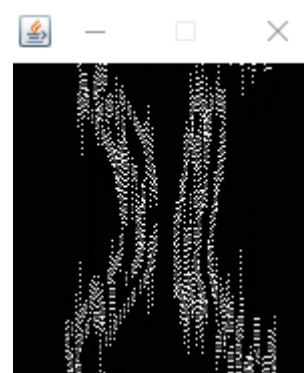
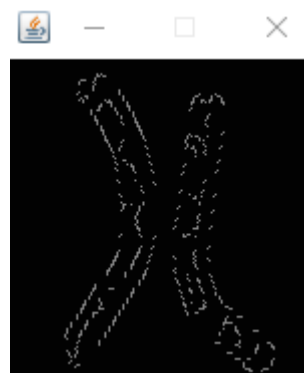
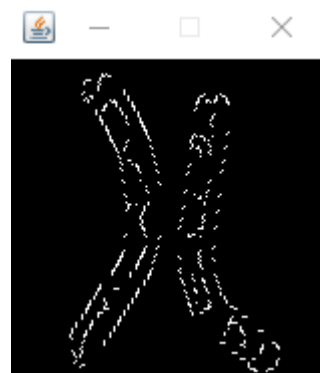
P=Longitud



P=Longitud/2



P=Longitud/3



Conclusión

Observamos que cuando la cantidad de P es igual a la cantidad de píxeles de la imagen obtiene un buen resultado dentro de los puntos y descriptores, para el caso de cuando se toma la mitad de los píxeles pierde alguno de los descriptores pero aun así su resultado es bueno, pero para cuando se toma una tercera parte de los píxeles es cuando gran parte de los descriptores se pierden y no conserva una buena información de lo caracterizado.

El uso descriptor de Fourier es importante ya que se enfoca en el reconocimiento de objetos y la determinación de parámetros de posicionamiento y escalado ha proporcionado resultados satisfactorios en tiempos cercanos al tiempo real.

Por otra parte, la estrategia de reducir el número de suscriptores de Fourier sin pérdida significativa de información permitirá optimizar los tiempos de ejecución.