Boletín de ejercicios de Visión Artificial

Grado de Ing. Informática

Curso 17/18

Las implementaciones propuestas se realizarán en Matlab (o herramientas similares: Octave, Python+NumPy/SciPy/Matplotlib, etc. si se prefiere).

Exceptuando las operaciones básicas de entrada/salida/visualización de imagen (imread, imwrite, imshow...) ordenación no se pueden utilizar funciones ya existentes de las librerías de tratamiento de imágenes ni otras relacionadas como convoluciones etc. Funciones no relacionadas con el análisis de imagen, como las de ordenación, pueden ser usadas también.

1. Histogramas

Implementar el algoritmo realce de contraste "window-level contrast enhancement", especificando el nivel de gris central y el tamaño de ventana.

```
function outputImage = histEnhance(inputImage, cenValue, winSize)
```

Implementar el algoritmo de compresión/estiramiento de histograma, que permita introducir los nuevos límites inferior y superior.

```
function outputImage = histAdapt(inputImage, minValue, maxValue)
```

2. Filtrado espacial: Suavizado y Realce

Implementar una función que permita realizar un filtrado espacial sobre una imagen con un kernel arbitrario que se pasará por parámetro.

```
function outputImage = convolve(inputImage,kernel)
```

donde la imagen de salida tiene el mismo tamaño que la imagen de entrada.

Implementar una función que calcule un kernel Gaussiano horizontal $1 \times N$, a partir de σ que será pasado como parámetro, y calculando N como $N = 2\lceil 3\sigma \rceil + 1$.

```
function kernel = gaussKernel1D(sigma)
```

Implementar una función que permita realizar un suavizado Gaussiano bidimensional usando un filtro $N \times N$ de parámetro σ, donde N se calcula igual que en la función anterior.

```
function outputImage = gaussianFilter2D(inputImage, sigma)
```

nótese que, al ser el filtro Gaussiano separable, podremos obtener este suavizado convolucionando la imagen, primero, con un kernel Gaussiano unidimensional $1 \times N$ y, luego, convolucionando el resultado con el kernel transpuesto $N \times 1$.

Implementar el filtro de orden de medianas. La función permitirá establecer el tamaño del filtro.

```
function outputImage = medianFilter2D(inputImage, filterSize)
```

Implementar el algoritmo High Boost que permita especificar, además del factor de amplificación A, el método de suavizado utilizado y su parámetro. Las opciones serán filtrado Gaussiano (con parámetro σ) y filtrado de medianas (con tamaño de ventana como parámetro).

```
function outputImage = highBoost(inputImage, A, method, parameter)
```

donde `method = 'gaussian' | 'median'`.

3. Operadores morfológicos

Implementar los operadores morfológicos de erosión, dilatación, apertura y cierre para imágenes **binarias**. Ambas funciones deben permitir especificar el tamaño del elemento estructurante y su forma (cuadrada, cruz, línea horizontal o vertical).

```
function outputImage = erode(inputImage, strElType, strElSize)
function outputImage = dilate(inputImage, strElType, strElSize)
function outputImage = opening(inputImage, strElType, strElSize)
function outputImage = closing(inputImage, strElType, strElSize)
```

donde `strElType = 'square' | 'cross' | 'linev' | 'lineh'`.

Implementar un algoritmo basado en operadores morfológicos que permita calcular las transformaciones *Top Hat* blanca y negra

```
function outputImage = tophatFilter(inputImage, strElType, mode)
```

donde `mode='white' | 'black'`

El filtro *Top Hat* blanco se define como la diferencia entre la imagen original y una apertura mientras que el negro es la diferencia entre un cierre y la imagen original.

4. Bordes y Esquinas

4.1. Operadores de primera derivada

Implementar una función que permita obtener las componentes de gradiente G_x y G_y de una imagen, pudiendo elegir entre los operadores de `Roberts`, `CentralDiff` (Diferencias centrales de Prewitt/Sobel sin promedio), `Prewitt` y `Sobel`.

```
function [gx, gy] = derivatives(inputImage, operator)
```

donde `operator='Roberts' | 'CentralDiff' | 'Prewitt' | Sobel'`

4.2. Canny

Implementar el detector de bordes de `Canny` mediante una función que permita especificar el valor de σ en el suavizado Gaussiano y los umbrales del proceso de histéresis

```
function outputImage = edgeCanny(inputImage, sigma, tlow, thigh)
```

4.3. Opcional – Detección de esquinas

Implementar el detector de esquinas de `Harris` que utilice Gaussianas tanto para la diferenciación como para la integración. La función permitirá establecer la escala de diferenciación σ_D, la escala de integración σ_I, y el valor del umbral para esquinas (t)

```
function outputImage = cornerHarris(inputImage, sigmaD, sigmaI, t)
```