Procesamiento y Análisis de Imágenes: Tarea 1

Alejandro Oróstica*

Universidad de Santiago de Chile
alejandro.orostica@usach.cl

October 9, 2019

I. Introducción

El filtro de difusión anisotrópica es una técnica que tiene por finalidad disminuir el ruido de una imagen sin tener eliminar partes importantes del contenido de la imagen, que es, por lo general lineas, bordes u otros detalles que son clave para la interpretación de la imagen.

II. Desarrollo

El filtro de difusión anisotrópico propuesto por Perona y Malik consiste en tomar un punto (x,y) de una imagen y realizar una resta entre dicho punto y sus puntos vecinos (norte, sur, este y oeste) y cada diferencia multiplicarlo por un coeficiente de difusión obtenido por una ecuación, la cual posee las siguientes variantes

$$h(x) = e^{-}(\frac{x}{k})^2 \quad (1)$$

$$h(x) = \frac{1}{1 + (\frac{x}{K})^2} \quad (2)$$

Una vez obtenidos estos datos se deben sumar los cuatro resultados y multiplicarlos por unλ donde $0 \leq \lambda \leq 0.25$ el resultado del producto se debe sumar al pixel punto original dando así salida al punto (x,y) de la imagen de salida. Este proceso se realiza una n cantidad de veces

Dicho lo anterior, el programa se abordó de la siguiente manera. Para lograr obtener las diferencias entre los puntos cuya ubicación corresponde a esquinas u orilllas se implementó un método que reflejaba los bordes de una imagen

```
def reflectEdges(imagen):
    dimx, dimy = imagen.shape
    output = np.zeros((dimx +2, dimy + 2))
    for x in range(output.shape[0]):
        for y in range(output.shape[1]):
            if (x == 0 and y == 0): ## Empezamos reflejando esquinas
                output[x,y] = imagen[x+1,y+1]
            elif (x == 0 and y == output.shape[1]-1):
                output[x,y] = imagen[x+1, imagen.shape[1]-2]
            elif (x == output.shape[0]-1 and y == 0):
                output[x,y] = imagen[imagen.shape[0] -2, y+1]
            elif ( x == output.shape[0]-1 and y == output.shape[1] -1 ):
                output[x,y] = imagen[imagen.shape[0]-2,imagen.shape[1]-2 ]
            elif ( x == 0 ): # Se reflejan las orillas
                output[x,y] = imagen[x+1, y-1]
            elif (x == output.shape[0]-1):
                output[x,y] = imagen[x-3, y-1]
            elif (y == 0):
                output[x,y] = imagen[x-1, y+1]
            elif(y == output.shape[1]-1):
                output[x,y] = imagen[x-1,y-3]
            else: # Se copia el resto de la imagen
                output[x,y] = imagen[x-1,y-1]

    return output
```

Figure 1: *Función para reflejar bordes*

Una vez obtenida una imagen cuyos bordes fueron reflejados se procede a realizar en otra función el filtro de difusión anisotropico propuesto por Perona y Malik.
Dicha función recibirá por parámetros una imagen, la variante de la fórmula a utilizar, una constante K que se recomienda que tome valores entre 20 y 100 y el valor de λ

*

```
def anisotropicDiffusion(img,option,k,lambdaCoeff):
    dimx, dimy = img.shape
    extendedImage = reflectEdges(img)
    output = np.zeros((dimx,dimy))
    for x in range(extendedImage.shape[0]):
        for y in range(extendedImage.shape[1]):
            if(x != 0 and x !=extendedImage.shape[0]-1 and y != 0 and  y !=extendedImage.shape[1]-1 ):
                northDiff = extendedImage[x-1,y] - extendedImage[x,y]
                southDiff = extendedImage[x+1,y] - extendedImage[x,y]
                westDiff = extendedImage[x,y-1] - extendedImage[x,y]
                eastDiff = extendedImage[x,y+1] - extendedImage[x,y]
                if(option == 1):
                    northCoeff = diffusionCoefficient1(northDiff,k)
                    southCoeff = diffusionCoefficient1(southDiff,k)
                    westCoeff = diffusionCoefficient1(westDiff,k)
                    eastCoeff = diffusionCoefficient1(eastDiff,k)
                else:
                    northCoeff = diffusionCoefficient2(northDiff,k)
                    southCoeff = diffusionCoefficient2(southDiff,k)
                    westCoeff = diffusionCoefficient2(westDiff,k)
                    eastCoeff = diffusionCoefficient2(eastDiff,k)
                north = northCoeff*northDiff
                south = southCoeff*southDiff
                west = westCoeff*westDiff
                east = eastCoeff*eastDiff
                output[x-1,y-1] = extendedImage[x,y] + lambdaCoeff*(north + south + west + east)
    return output
```

Figure 2: *Implementación del algoritmo*

El software desarrollado pedirá al usuario por pantalla el número de iteraciones, el valor de λ, el nombre de la imagen y la variante de la ecuación a utilizar

```
Ingrese el nombre de la imagen junto a su formato : car.tiff
Ingrese el valor del coeficiente lambda: 0.25
Ingrese el valor de la constante K: 10
Ingrese el número de iteraciones: 1
Ingrese 1 para utilizar la variante exponencial o 2 para la variante fraccionaria: 1
```

Figure 3: *Inicio del Programa*

Una vez ingresado los parametros mostrará la imagen resultante comparándola con la original

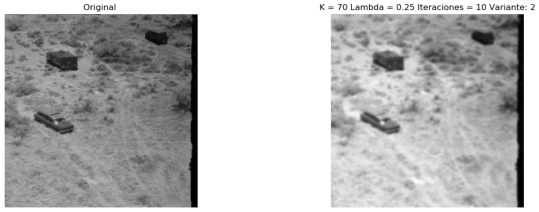

Figure 4: *Salida del Programa*

III. Evaluación de Resultados

En esta sección se evaluarán los resultados obtenidos a través del proceso de aplicar el filtro a imágenes variando sus parámetros

i. Parámetro K

La constante K usada en las ecuaciones mencionadas en el capítulo 2 tiene relación con que tan difuminada quedará la imagen de salida a continuación se muestran los resultados llevados a cabo variando dicho parámetro

Figure 5: *K = 40*

Figure 6: *K= 100*

Figure 7: *K= 50*

ii. Parámetro λ

El parámetro λ tiene un efecto similar a la constante K pero es más notorio cuando se varía mucho su valor

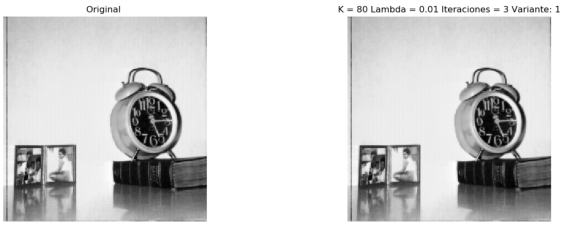

Figure 8: $\lambda = 0.01$

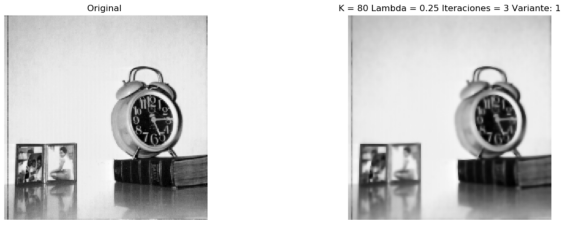

Figure 9: $\lambda = 0.25$

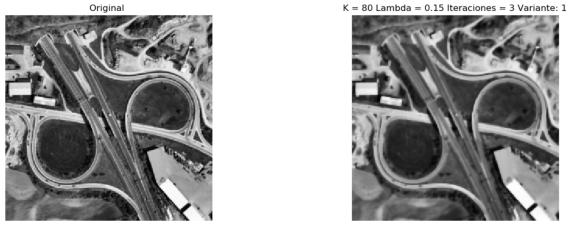

Figure 10: $\lambda = 0.15$

iii. Número de iteraciones

Con un mayor número de iteraciones se logra obtener una imagen más lisa, también muy similar a los otros dos parámetros también logra una imagen más borrosa

Figure 11: *10 iteraciones*

Figure 12: *50 iteraciones*

IV. Conclusiones

A modo de conclusión se puede decir que los objetivos se cumplieron pero no del todo, ya que a pesar de haber implementado el filtro siguiendo el paper de Perona y Malik, no se comprobó que este lo ejecutase de manera correcta. Siguiendo esa misma linea el algoritmo en sí no es eficiente ya que toma largos tiempos de espera para muchas iteraciones en imagenes de tamaño medio-grande (apróximadamente 9 segundos por iteración para una imagen de 512x512). Los datos recabados acerca de los parámetros no son del todo concluyentes ya que no se probaron en todo tipo de imagenes para ver como influían en foto con distintas características

Respecto a las dificultades de este trabajo se utilizó bastante tiempo en conocer bien el algoritmo y entenderlo, ya es que es la primera vez que se debe buscar en trabajos de investigaciones

References

[1] Perona, P., Malik, J. (1990a). Scale-space and edge detection using anisotropic diffusion. IEEE, .