

Laboratorio 5

RESTAURACIÓN DE IMÁGENES RESULTANTES DEL LABORATORIO 01

Abiel Porras Garro - 2020209597¹, Elías Castro Montero - 2020098930²

¹Ingeniería en Computación, Instituto Tecnológico de Costa Rica, San José, Costa Rica

²Ingeniería en Computación, Instituto Tecnológico de Costa Rica, San José, Costa Rica

I. RESTAURACIÓN DE PÍXELES BORRADOS CON INTERPOLACIÓN

Para realizar los ejemplos de este laboratorio se utilizarán imágenes distintas a las nuestras que habíamos creado en el laboratorio 1 debido a que estas contenían marcas de agua en varios lugares y pensamos que podrían afectar los resultados de las siguientes etapas. Sin embargo, las nuevas imágenes del paisaje original cuentan con los píxeles borrados en patrón de ajedrez y fueron realizadas con el mismo programa realizado en dicho laboratorio.



Fig. 1: Imagen original



(a) lab1_imagen1

(b) lab1_imagen2

Fig. 2: Imágenes con píxeles faltantes

Para la obtención de los píxeles faltantes se utilizó la herramienta realizada en el laboratorio 4, donde se hace

uso de interpolación para obtener la información ausente. La interpolación realiza una búsqueda de los valores de los 4 píxeles colindantes con el píxel que se desea obtener su información. Solo se escogen los píxeles que tienen un valor igual al primer píxel de la imagen debido a que todos los demás del patrón de ajedrez deben tener el mismo valor, esto también se puede realizar escogiendo aquellos píxeles cuya suma de su valor en el eje x y eje y sea par.

Una vez el algoritmo de interpolación encuentra el valor de los 4 píxeles colindantes realiza un promedio de los valores obtenidos y se le asigna al píxel actual. Cuando se ha realizado esto con todos los píxeles faltantes de la imagen se obtiene una nueva representación muy parecida a la original.



Fig. 3: Imágenes reconstruidas con interpolación N=4

Como se puede apreciar, aunque en los píxeles faltantes pueda haber un valor distinto, el de los píxeles colindantes no cambia, por lo que mediante la interpolación se va a generar casi el mismo resultado.

II. MEJORA DE NITIDEZ MEDIANTE OPERADOR LAPLACIANO

Para realizar la mejora de nitidez se utiliza la siguiente fórmula

$$g(x, y) = f(x, y) + c\nabla^2 f(x, y)$$

donde:

$f(x, y)$ es la imagen de entrada

$g(x, y)$ es la imagen resultante

c es la constante de amplificación

∇^2 es el operador Laplaciano

En nuestro caso, esta fórmula se representa mediante la función `improve_sharpness` que recibe la imagen

obtenida en el paso anterior (el resultado de aplicar interpolación), y la constante c para amplificar los bordes.

```
def improve_sharpness(img, c):
    ddepth = cv2.CV_16S
    kernel_size = 3
    dst = cv2.Laplacian(img, ddepth,
        ksize=kernel_size)
    abs_dst = cv2.convertScaleAbs(dst)
    final = img + c * abs_dst
    return final
```

Para calcular el operador Laplaciano de la imagen, se utiliza la función `cv2.Laplacian(img, ddepth, kernel_size)` de OpenCV, esta función implementa el operador Laplaciano discreto, definida como $\text{Laplace}(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$ [1].

Además, esta función recibe 3 parámetros, la imagen de entrada, un valor `ddepth`, que corresponde a la profundidad de la imagen de destino, también interpretado como el tipo de dato de la imagen de salida, nosotros elegimos `cv2.CV_16S` para evitar overflow, ya que es lo que recomienda la documentación de OpenCV, y por último, `kernel_size` que es el tamaño del kernel para el operador Sobel que se aplica internamente, debido a que OpenCV implementa el operador Laplaciano ejecutando el operador Sobel [1].

Después de varias pruebas, se decidió utilizar tamaño 3 para el kernel, debido a que con tamaño 1 los bordes no se marcaban lo suficiente para que se viesen claramente, y con tamaño 5 los bordes se ven muy sucios, a continuación, en la figura 4 se aprecian las diferencias de utilizar el operador Laplaciano con kernel de tamaño 1, 3 y 5, utilizando $c = -0.2$, valor escogido gracias a un análisis previo que será descrito en la sección III.

Además, en conjunto con la función anterior, se utiliza `cv2.convertScaleAbs(dst)` de OpenCV, para regresar el tipo de dato de la imagen de salida a `uint8` [1].

Por último, gracias a que las imágenes son implementadas mediante `arrays` de numpy (biblioteca para computación científica en Python), se pueden utilizar operadores aritméticos fácilmente, por ende, basta con calcular `img + c * abs_dst` para obtener el resultado final.

III. MEJOR RESULTADO OBTENIDO

Después de distintas pruebas modificando el valor de c pudimos notar cuales valores son los más adecuados para esta constante. Para empezar, pensamos que el valor de c debe estar entre los valores -0.5 y 0.5 , debido a que después de estos la imagen adquiere suciedad de píxeles en vez de nitidez. Además, si el valor es negativo, la amplificación bajaría la luminosidad de los bordes y por el contrario si el valor es positivo, se amplificará la luminosidad. Esto es muy útil dependiendo de la situación de la imagen, ya que si está en general tiene colores muy claros es mejor remarcar los bordes con un color más oscuro para que sea más vistoso el efecto.

El mejor caso obtenido por nosotros fue con el valor $c = -0.2$ debido a que el resultado de los bordes es bastante notorio y asimismo da una sensación de mejor acabado. Por otro lado, las zonas afectadas y la cantidad de píxeles que se



(a) Operador Laplaciano con kernel de tamaño 1

(b) Operador Laplaciano con kernel de tamaño 3



(c) Operador Laplaciano con kernel de tamaño 5

Fig. 4: Uso del Operador Laplaciano con diferentes tamaños de kernel

modifican son las adecuadas, en virtud de que no se siente suciedad en la imagen. Finalmente, pensamos que es una imagen que se presta mucho más a adquirir nitidez mediante tonalidades oscuras debido a su paleta de colores, ya que la imagen final tiene un mejor resultado final con esta tonalidad.



Fig. 5: Imagen con mejor resultado en nitidez

Como modo de comparación, la figura 6 muestra la diferencia entre utilizar valores de c positivos y negativos, en este caso $c = 0.2$ y $c = -0.2$.



Fig. 6: Uso de valores positivos y negativos de c

IV. CONCLUSIONES

- A pesar de que los datos recuperados por la interpolación no son exactos, son muy cercanos al original. Sin embargo, para lograr este resultado los datos faltantes siempre deben ser igual a un píxel, debido a que si la cantidad de píxeles perdidos unidos empieza a aumentar ya no habría buenos datos con los que deducir el faltante y por lo tanto se volvería un trabajo de adivinar los datos ausentes con información que puede ser muy distinta a la original.
- La utilización de la documentación de bibliotecas como en este caso OpenCV, permite una mejor comprensión de las herramientas y personalización de los resultados obtenidos, en este caso, gracias a que se revisó la información existente respecto a la forma en la que OpenCV implementa el operador Laplaciano, se supo que se utiliza el operador Sobel, el cual genera resultados distintos en función de su tamaño de kernel, por lo que a la hora de realizar distintas pruebas, identificamos el valor adecuado desde nuestro punto de vista; el desconocimiento de esta información, podría hacer que no aprovechemos al máximo el potencial que nos ofrecen estas herramientas.
- Los valores de c positivos (oscuros) realzan mucho mejor los bordes que los valores de c negativos (claros), debido a que se hace mucha mejor diferenciación a la vista. Esto también depende de la imagen, pero en nuestras pruebas las imágenes al ser de paisajes contaban con tonalidades claras, por lo que los bordes con más nitidez se obtienen a la hora de resaltar sus bordes con colores oscuros a diferencia de los colores claros, en donde no se aprecian de la misma forma.

REFERENCES

- [1] OpenCV, “OpenCV: Laplace Operator.” [Online]. Available: https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html