Eliminación de Ruido del Tipo Salt and Pepper en un vídeo

Integrantes:

- Kimberly Calderón Prado - 2017088598
- Jose Antonio Ortega González - 2017101758
- Andrey Sibaja Garro - 2017101898

1. Fast Median Filter Approximation (FMFA)

Este filtro es una optimización al filtro de la mediana utilizado para eliminar el tipo de ruido de sal y pimienta, en este se toman las columnas de un *kernel* o ventana de vecinos. La mejora que se realiza consiste en la acumulación de los resultados de forma que solo se deba calcular el ordenamiento de una columna, ya que se conservan los datos de la segunda y tercera columna del ciclo anterior, pasando a ser la nueva primer y segunda columna respectivamente [1].

Pseudocódigo

Entradas:

```
I : imagen a ser filtrada
H : altura de la imagen
W : ancho de la imagen
```

Salidas:

```
I_salida : imagen filtrada
```

Algoritmo:

```
for i = 2 to H-1
     col1 = median(I(i-1, 1), I(i, 1), I(i+1, 1))
     col2 = median(I(i-1, 2), I(i, 2), I(i+1, 2))

     for j = 3 to W-1
          col3 = median(I(i-1, j), I(i, j), I(i+1, j))
          I_salida(i, j) = median([col1; col2; col3])

          col1 = col2
          col2 = col3
```

Para la primera iteración del algoritmo se puede interpretar de la siguiente manera:

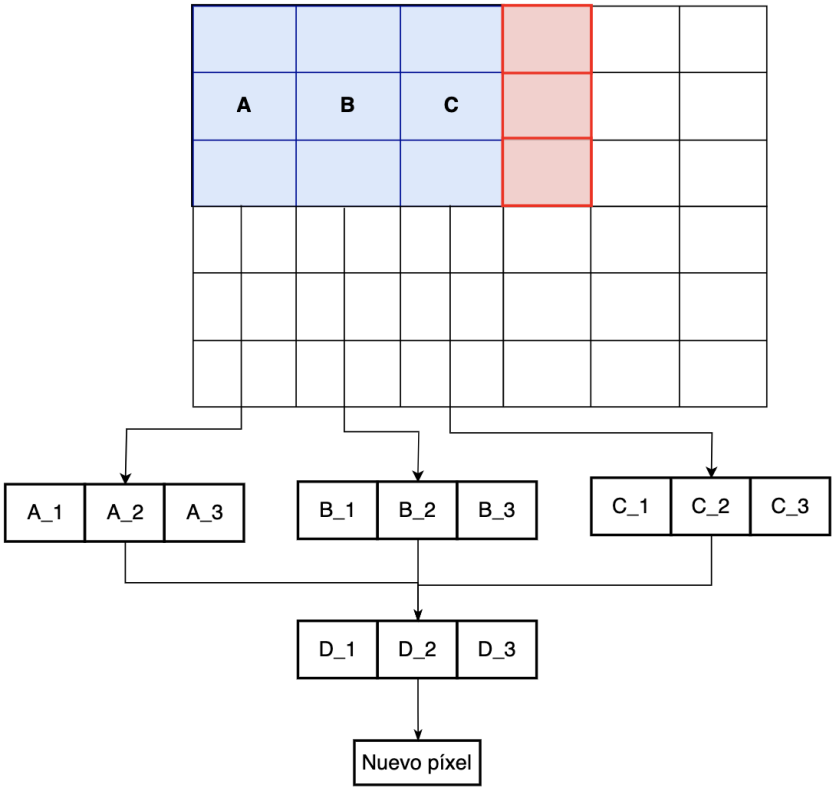

Figura 1. Fast median approximation en primera iteración.

Como se observa en la figura 1, la columna roja representa el único cálculo de columna que se debe realizar en la siguiente iteración ya que los valores para la columna B y C son reutilizados.

2. *High-Performance Modified Decision Based Median Filter Algorithm (HPMDBMF)*

Este algoritmo toma matrices de 3x3 de la imagen para realizar una búsqueda de valores de 0 y 255, si no existe ninguno entonces el píxel se mantiene, esto le da eficiencia al algoritmo al no tener que procesar píxeles que no se encuentran con ruido. En caso de que todos los valores sean 0 y/o 255 se amplía la matriz a una 5x5 para realizar el cálculo, esto es lo que da al algoritmo eficiencia a la hora de obtener la imagen filtrada y obtener un índice de similaridad mayor con respecto a la imagen original.

Pseudocódigo

Mid-Value-Decision Median

Función: mvdm(row)

Entradas:

```
row : vector de 3 valores para determinar el medio
```

Salidas:

```
result : valor medio de la entrada
```

Algoritmo:

```
Y = sort(row)

p_1 = primer valor de Y
p_2 = segundo valor de Y
p_3 = tercer valor de Y

Si p_2 == 0
     result = p_3
sino Si p_2 == 255
     result = p_1
sino
     result = p_2
```

Traverse matrix

Función: traverse(matrix)

Entradas:

```
matrix : matriz a buscar valores diferentes de 0 y 255
```

Salidas:

```
pixel : valor del pixel, [0, 1]
```

Algoritmo:

```
pixel = 0

Sea m y n las dimensiones de matrix

for i=1 hasta m
     for j=1 hasta n
          Si pixel > 0 y pixel < 255
               break
```

HPDBMF

Función: hpdbmf(row)

Entradas:

```
img : imagen a aplicar filtro
```

Salidas:

```
Y : resultado imagen filtrada
```

Algoritmo:

```
Sea m y n las dimensiones de img.

for i = 3 hasta m-2
  for j = 3 hasta n-2
    pixel = img(i,j)
    Si pixel > 0 y pixel < 255 entonces
      Y(i,j) = pixel
    sino
      window = img(i-1:1+j, j-1:j+1)
      M1 = mvdm(img(i-1:i+1, j-1))
      M2 = mvdm(img(i-1:i+1, j))
      M2 = mvdm(img(i-1:i+1, j+1))
      median_value = mvdm([M1, M2, M3])

      Si median_value > 0 y median_value < 255
          Y(i,j) = median_value
      sino
          median_value = traverse(img(i-2:i+2,j-2:j+2))

          Si median_value > 0 y median_value < 255 entonces
               Y(i,j) = median_value
          sino
               median_value = median(img(i-2:i+2, j-2:j+2))
               Y(i,j) = median_value
```

Comparación entre ambos algoritmos

Para realizar la comparación entre ambos algoritmos se utilizó la función *ssim* que permite determinar la similitud entre dos imágenes, dando como resultado un valor entre 0 y 1 donde 1 es una similitud del 100%. En la figura 2 se observa el ssim para cada frame del video obtenido al aplicar el filtro de *fast median* y el de HPDBMF.

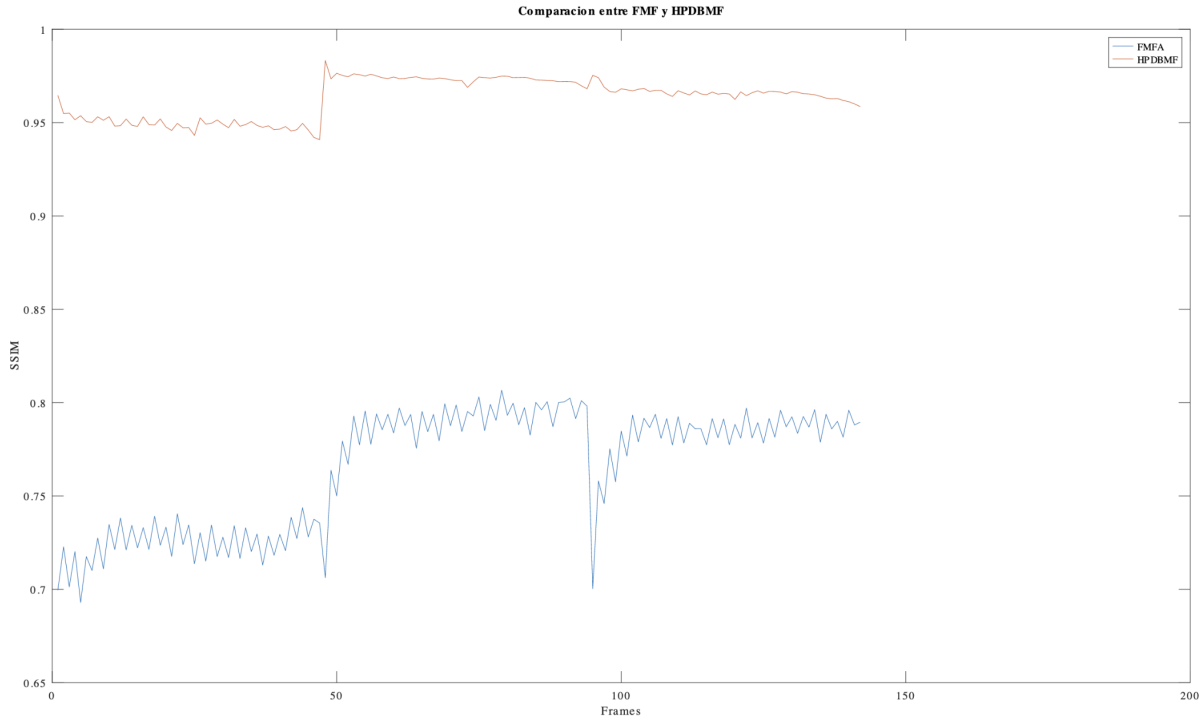

Figura 2. Comparación de valores de ssim de los algoritmos *FMF* y *HPDBMF*.

Se puede observar que el vídeo obtenido para el algoritmo de *HPDBMF* posee mayores valores de *ssim* lo que indica que eliminó de una manera más eficiente el ruido de sal y pimienta que se presentaba.

Esta eficiencia comparada con la del *fast median filter* se debe a que este no aplica el algoritmo a todos los píxeles de la imagen, sino que va evaluando cuales contienen ruido y procesa solo estos, comparado con el de *FMF* que se ejecuta sobre todos los píxeles modificando sus valores y en el de HPDBMF se mantienen, esto mejora en gran medida el tiempo de ejecución y el índice de similitud con respecto al *fast median*. También el procesamiento sobre cada píxel es más riguroso en el *HPDBMF* por lo que produce mejores resultados, los cuáles se ven reflejados en la figura 2.

Referencias

[1] O. Appiah, M. Asante y J. B. Hayfron-Acquah "*An Effective Algorithm for Denoising Salt-AndPepper Noise in Real-Time*". En: *Asian Journal of Research in Computer Science* (2020).