



UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

Computación Gráfica

Operadores lógicos y operaciones en imágenes

GRUPO 3:
Renzo Vicente Castro
Andy Ñaca Rodríguez
Eduardo Sánchez Hincho
Luis Villanueva Flores

Docente:
Vicente Machaca Arceda

AREQUIPA
2020

1. PRACTICA 6

- Implemente la adición de imágenes con los elementos de la Figura 1. Se recomienda hacer un cast a la imagen antes del procesamiento para evitar el overflow en los píxeles.



Figura 1: Figuras de muestra.

Código:

Listing 1: Ejercicio 1

```
1      import cv2
2      import numpy as np
3      from matplotlib import pyplot as plt
4
5      img1=cv2.imread('add_1.jpg',0)
6      img2=cv2.imread('add_2.jpg',0)
7      img1=cv2.resize(img1,(img2.shape[1],img2.shape[0]))
8      img3=cv2.imread('blank.jpg',0)
9      f,c=img1.shape
10     for i in range(f):
11         for j in range(c):
12             r=int(img1[i][j]/2)+int(img2[i][j]/2)
13             if(r<0):
14                 img3[i][j]=0
15             elif(r>255):
16                 img3[i][j]=255
17             else:
18                 img3[i][j]=r
19
20     cv2.imshow('res',img3)
```

Explicación: Se importan las imágenes y luego se ajustan los tamaños mediante el comando resize, como las imágenes van a tener el mismo tamaño entonces sacamos el largo y ancho de la imagen para posteriormente trabajar en la adición. En una tercera imagen (en blanco), aplicamos dicho operador aritmético siempre teniendo en cuenta los límites de color y convirtiendo a entero cada píxel de cada imagen que intervendrá en dicha operación. Finalmente mostramos la imagen.

Resultado:



Figura 2: Resultado.

- Ahora implemente la adición con imágenes a colores (Figura 3).



Figura 3: Figuras de muestra.

Código:

Listing 2: Ejercicio 2

```

1   import cv2
2   import numpy as np
3   from matplotlib import pyplot as plt
4
5   img1=cv2.imread('add_10.jpg')
6   img2=cv2.imread('add_11.jpg')
7   img1=cv2.resize(img1,(img2.shape[1],img2.shape[0]))
8   img3=cv2.imread('blank2.jpg')
9   f,c,color=img1.shape
10  f,c,color=img1.shape
11  for i in range(f):
12      for j in range(c):
13          r1=int(img1[i][j][0])+int(img2[i][j][0])
14          r2=int(img1[i][j][1])+int(img2[i][j][1])
15          r3=int(img1[i][j][2])+int(img2[i][j][2])
16          if(r1<0):
17              img3[i][j][0]=0
18          elif(r1>255):
19              img3[i][j][0]=255
20          else:
21              img3[i][j][0]=r1
22          if(r2<0):
23              img3[i][j][1]=0
24          elif(r2>255):
25              img3[i][j][1]=255
26          else:
27              img3[i][j][1]=r2
    if(r3<0):

```

```

28         img3[i][j][2]=0
29     elif(r3>255):
30         img3[i][j][2]=255
31     else:
32         img3[i][j][2]=r3
33
34 cv2.imshow('res',img3)

```

Explicación: Se aplica el procedimiento del ejercicio 1 con la diferencia de que aquí tendremos 3 parámetros y no solo dos, este tercero sería el color, dicho parámetro tiene 3 valores por lo que al trabajar con colores tendremos que adaptar las operaciones para estos tres casos. Aplicando todo ello obtenemos la siguiente imagen.

Resultado:



Figura 4: Resultado.

- Implemente la sustracción de imágenes para segmentar letras. Se le esta brindando una foto del documento y otra de una hoja en blanco para eliminar el reflejo de la luz (Figura 3). Después de la sustracción aplique thresholding para obtener un resultado similar a la Figura 4. Tiene la libertad de aplicar métodos adicionales para mejorar los resultados, por ejemplo: contrast stretching, histogram equalization, etc.

Código:

Listing 3: Resta de imágenes

```

1  #Carga de imágenes
2  img = cv2.imread("sub_1.jpg",0)
3  img2 = cv2.imread("sub_2.jpg",0)
4  #Igualación de tamaño
5  img2=cv2.resize(img2,(img.shape[1],img.shape[0]))
6  #Resta bit a bit de ambas imágenes
7  for i in range(img2.shape[0]):
8      for j in range(img2.shape[1]):
9          img[i,j]=np.clip(int(img[i,j])-int(img2[i,j])+100,0,256)
10 #THRESHOLDING
11 v_f=np.vectorize(lambda x:0 if x<80 else 255)
12 #Resultado
13 cv2.imshow("out.jpg",np.uint8(v_f(img)))

```

Explicación:

Primero cargaremos las dos imágenes que serán restadas posteriormente, luego es necesario igualar sus tamaños ya que sin este paso tendríamos errores de acceso en la memoria. Luego recorremos píxel a píxel para hacer la resta correspondiente, usamos la función clip() para que esta operación no exceda los límites de 0 y 255, además se le suma 100 para evitar tener valores negativos. Finalmente aplicamos un thresholding de 80 haciendo uso de función vectorizadas, que afectan a cada elemento de una matriz

que en este caso sera nuestra imagen.

Resultados:

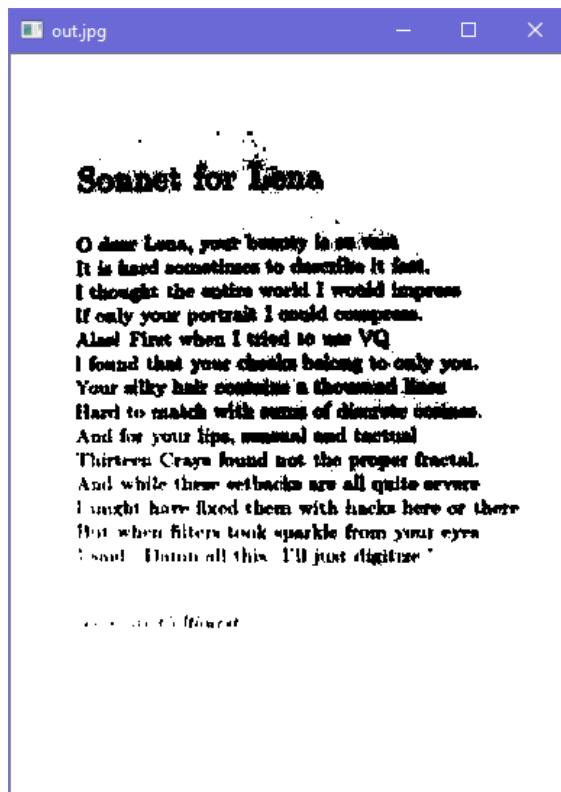


Figura 5: Imagen con mejor detalle en el texto

- Implemente la sustracción de imágenes para detectar el cambio o movimiento de objetos en fotogramas. En la Figura 5, se brindan dos fotogramas consecutivos, implemente la sustracción para obtener una imagen donde se visualice que objetos han cambiado de posición.

Código:

Listing 4: Resta de imágenes

```
1 img = cv2.imread("sub_10.jpg",0)
2 img2 = cv2.imread("sub_11.jpg",0)
3 img2=cv2.resize(img2,(img.shape[1],img.shape[0]))
4
5 for i in range(img2.shape[0]):
6     for j in range(img2.shape[1]):
7         img[i,j]=np.clip(abs(int(img[i,j])-int(img2[i,j])),0,256)
8 cv2.imshow("out.jpg",img)
```

Explicación:

Para este ejercicio se usara la resta de imágenes para detectar el cambio de una imagen a otra, para eso se realizara el mismo procedimiento anterior, igualamos el tamaño de ambos imágenes, luego píxel a píxel realizamos la resta usando la función clip() y abs() para evitar valores negativos.

Resultado:

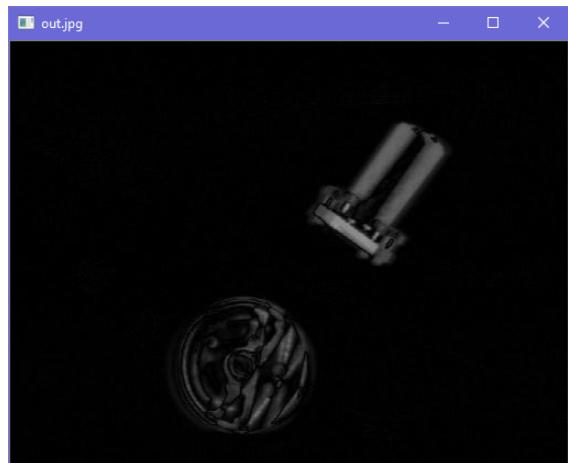


Figura 6: Cambio de una imagen a otra

- Finalmente, agregue los operadores de adición y sustracción a su software de procesamiento de imágenes.

Agregamos los botones de suma, suma color , resta a nuestra barra de acciones , también agregamos otra barra para subir otra imagen, esto para el caso que se quieran sumar o restar dos imágenes.

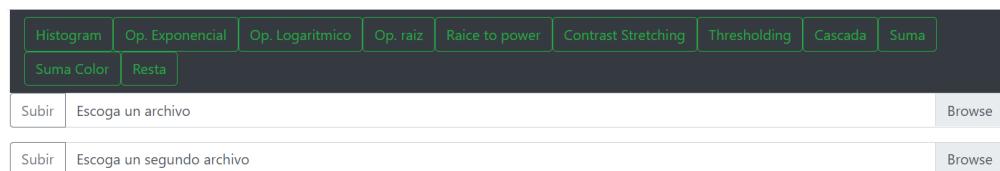


Figura 7: Barra de acciones

Se muestra a continuación la suma de dos imágenes:

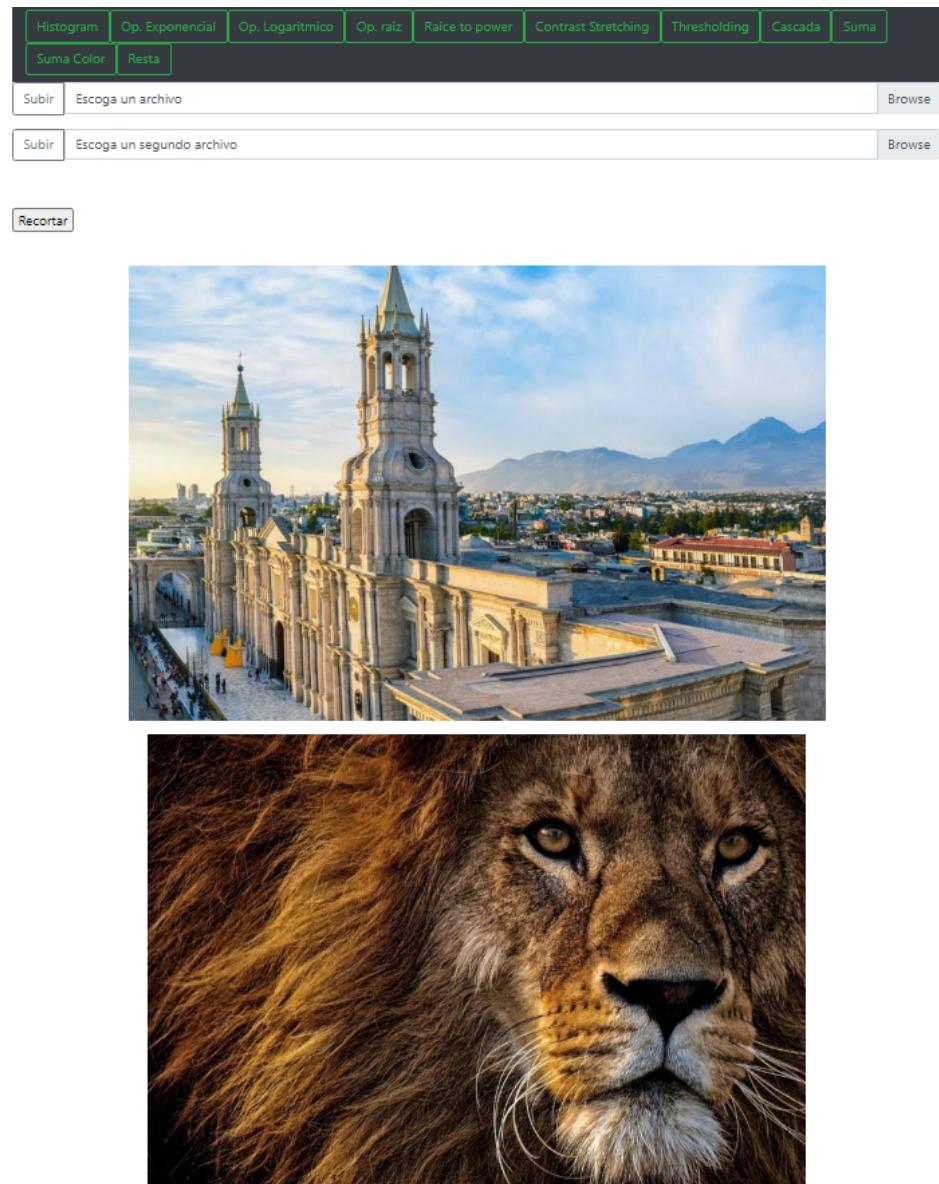


Figura 8: Imágenes a sumar

Mostramos anteriormente como el programa nos carga las imágenes que vamos a sumar, ahora se muestra cual es el resultado de dicha suma:

Lo mismo sucede para el caso de la resta.



Figura 9: Resultado de la suma de dos imágenes a color

2. Conclusiones

- La aplicación del operador suma en imágenes tiene que aplicarse con mucho cuidado teniendo en cuenta los límites de color de los píxeles, ya que si no se tiene en cuenta esto los resultados no serían favorables.
- Para mejorar los resultados a veces es necesario escalar la imagen por eso se divide entre dos el valor de cada píxel convertido a entero.
- La resta de imágenes puede ayudarnos a quitar detalles que no ayudan en el análisis de una imagen, de forma intuitiva si queremos quitar un elemento de una imagen, podemos restarle esa parte como si se tratara de un corte.
- Una resta de imágenes detecta cambios de una imagen a otro, lo cual se puede aplicar en videos para detectar movimiento por ejemplo.
- Puesto que nuestra estructura en nuestro programa web fue ordenada , no tuvimos problema alguno para integrar los demás operadores.
- A veces suele ser necesario hacer una resta de imágenes para hacer un mejor uso del thresholding, por ejemplo para el caso 3 le restamos el fondo y así el fondo de nuestra imagen resultante quedaría de la misma intensidad, pudiendo aplicar el thresholding para hacer una correcta binarización.

3. PRACTICA 7

- Ejercicio 1:
Implemente la multiplicación de imagen por una constante con la Figura 1. Evalué con $c = 2$, $c = 5$ y $c = 7$.



Figura 10: Imagen de muestra

Código:

Listing 5: Ejercicio 1

```

1   import cv2
2   import numpy as np
3   from matplotlib import pyplot as plt
4
5   img1=cv2.imread('tigre.jpeg')
6   img1=cv2.resize(img1,(400,400))
7   f,c,color=img1.shape
8   constante=int(input('Inserte constante '))
9   for i in range(f):
10      for j in range(c):
11          r1=int(img1[i][j][0])*constante
12          r2=int(img1[i][j][1])*constante
13          r3=int(img1[i][j][2])*constante
14          if(r1<0):
15              img1[i][j][0]=0
16          elif(r1>255):
17              img1[i][j][0]=255
18          else:
19              img1[i][j][0]=r1
20          if(r2<0):
21              img1[i][j][1]=0
22          elif(r2>255):
23              img1[i][j][1]=255
24          else:
25              img1[i][j][1]=r2
26          if(r3<0):
27              img1[i][j][2]=0
28          elif(r3>255):
29              img1[i][j][2]=255
30          else:
31              img1[i][j][2]=r3
32
33   cv2.imshow('res',img1)

```

Explicación: Al igual que la suma de imágenes la aplicación del operador producto es la misma pero en este caso lo aplicaremos a nivel de colores y con una constante. Importamos las librerías necesarias, importamos nuestra imagen, le damos un tamaño menor para ver los resultados de mejor manera, luego lo trabajamos como si fuera una matriz, es decir sacamos sus filas y columnas(ancho y alto) y posteriormente aplicamos el operador entre los píxeles de nuestra imagen original y la constante que será solicitada por consola.

Resultados:

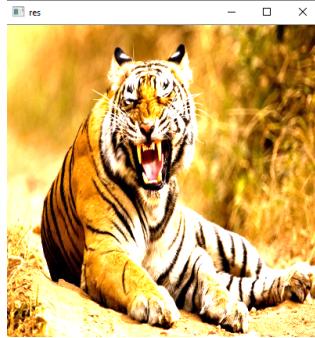


Figura 11: Resultado cuando c=2.

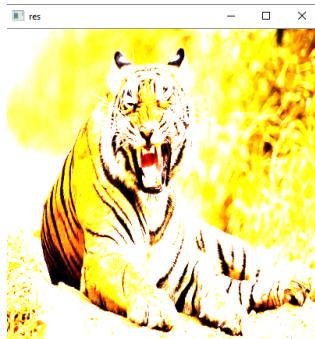


Figura 12: Resultado cuando c=5.

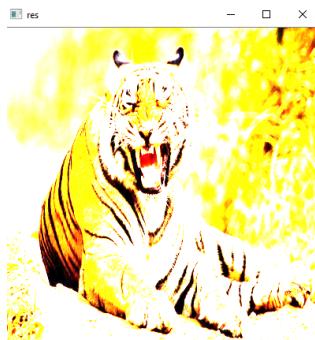


Figura 13: Resultado cuando c=7.

■ Ejercicio 2:

Implemente la división de imágenes para segmentar letras. Se le esta brindando una foto del documento y otra de una hoja en blanco para eliminar el reflejo de la luz (Figura 5). Después de la división deberá normalizar la imagen a valores entre [0 - 255] (Ecuación 1). Después aplique thresholding para obtener un resultado similar a la Figura 6. Tiene la libertad de aplicar métodos adicionales para mejorar los resultados, por ejemplo: Contrast stretching, histogram equalization, etc.

$$I' = (I - \min) * ((\text{newMax} - \text{newMin}) / (\max - \min)) + \text{newMin} \quad (1)$$



Figura 14: Fotos de imágenes para segmentar.

Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactful
Thirteen Crayas found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Collier

Figura 15: Resultado de la segmentación.

Código:

```

1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4 img1=cv2.imread('letras1.jpeg',0)
5 img2=cv2.imread('letras2.jpeg',0)
6 img3=cv2.imread('blankd.jpg',0)
7 f,c=img1.shape
8 for i in range(f):
9     for j in range(c):
10         img3[i][j]=(int(img1[i][j])/int(img2[i][j]))*100
11
12 maximo=np.max(img3)
13 minimo=np.min(img3)
14 for i in range(f):
15     for j in range(c):
16         intensidad=(int(img3[i][j])-minimo)*(255/(maximo-minimo))
17         if(intensidad<0):
18             img3[i][j]=0
19         elif(intensidad>255):
20             img3[i][j]=255
21         else:
22             img3[i][j]=intensidad
23
24 hist = cv2.calcHist([img3], [0], None, [256], [0, 256])
25
26
27 for i in range(f):
28     for j in range(c):
29         if(img3[i][j]<180):
30             img3[i][j]=0

```

```

31
32     else:
33         img3[i][j]=255
34
35     cv2.imshow('res',img3)
36     plt.plot(hist, color='gray')
37     plt.xlabel('Intensidad de iluminacion')
38     plt.ylabel('Cantidad de pixeles')
39     plt.show()

```

Explicación: Primeramente importamos las librerías necesarias y luego 3 imágenes, la primera será la imagen de una hoja que tiene escrito una serie de caracteres y está con el reflejo de la luz, la otra es una imagen de una hoja en blanco que nos permitirá quitar dicho reflejo y la última un lienzo en blanco donde estará nuestra imagen resultado. Aplicamos el operador división a nivel de píxeles entre la imagen 1 y la imagen 2, a este resultado le multiplicamos por una constante para que pueda visualizarse, posteriormente lo sobreescrivimos en la imagen 3, a esta última seguido le aplicamos la fórmula que se brinda en el enunciado para escalar la imagen. Finalmente se le aplica thresholding basándose en el mejor umbral y para ello calculamos el histograma de nuestra imagen, obviamente previo al thresholding.

Resultado:

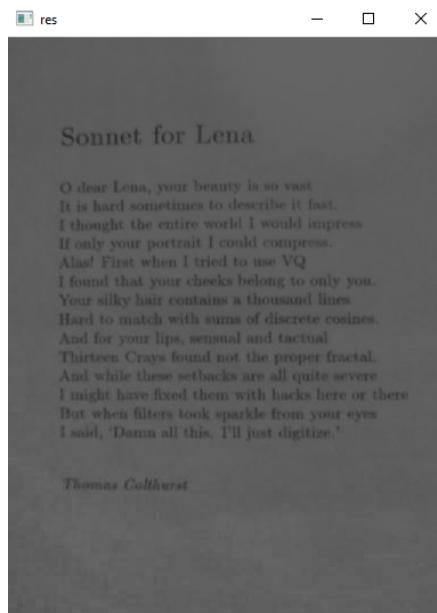


Figura 16: Imagen después de la división.

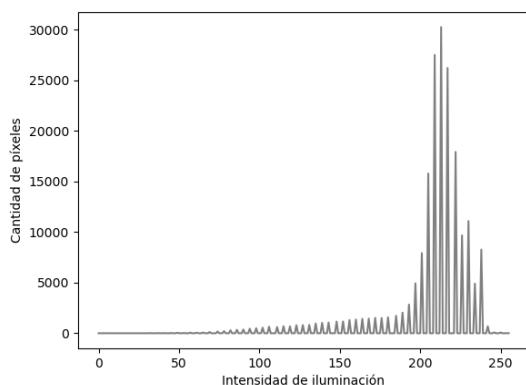


Figura 17: Histograma de la imagen dividida.



Sonnet for Lena

O dear Lena, your beauty is so vast
 It is hard sometimes to describe it fast.
 I thought the entire world I would impress
 If only your portrait I could comprise.
 Alas! First when I tried to use VQ
 I found that your cheeks belong to only you.
 Your silky hair contains a thousand lines
 Hard to match with sums of discrete cosines.
 And for your lips, sensual and tactile
 Thirteen Crays found not the proper fractal.
 And while these setbacks are all quite severe
 I might have fixed them with hacks here or there
 But when filters took sparkle from your eyes
 I said, ‘Darn all this. I’ll just digitize.’

Thomas Colkeret

Figura 18: Imagen segmentada con thresholding.

Ejercicio 3:

Implemente la división de imágenes para detectar el cambio o movimiento de objetos en fotogramas. En la Figura 4, se brindan dos fotogramas consecutivos, implemente la división para obtener una imagen donde se visualice que objetos se movieron, quizás sea necesario multiplicar el resultado por una constante para poder visualizar mejor los resultados. Después puede aplicar contrast stretching para mejorar aun mas los resultados.

Código:

```

1     img = cv2.imread("sub_10.jpg",0)
2     img2 = cv2.imread("sub_11.jpg",0)
3     img2=cv2.resize(img2,(img.shape[1],img.shape[0]))

```

```

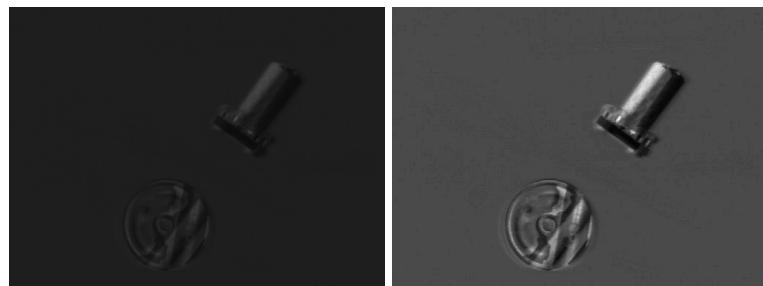
4     #Division
5     for i in range(img2.shape[0]):
6         for j in range(img2.shape[1]):
7             img[i,j]=int(img[i,j])/int(img2[i,j])*30
8     mi=np.min(img)
9     ma=np.max(img)
10    #Contrast stretching
11    for i in range(img2.shape[0]):
12        for j in range(img2.shape[1]):
13            img[i,j]=(int(img[i,j])-mi)*(255-0)/(ma-mi)+0
14    cv2.imwrite("out.jpg",img)

```

Explicación:

Como siempre, cargaremos nuestras imágenes e igualaremos sus tamaños. Luego procedemos a realizar la división entre las imágenes y es necesario multiplicarlo por una constante, ya que los valores obtenidos están entre 0 y 1. Al momento de guardarlo en la matriz que sera nuestra imagen se realiza un casteo a entero, perdiendo toda la información decimal de la división. Finalmente aplicaremos contrast stretching para mejorar la calidad y contrastes.

Resultado:



(a) División multiplicada por 30 (b) Mejora con contrast stretching

Figura 19: Detección de cambios entre fotogramas

■ Ejercicio 4:

Implemente el operador Blending (ecuación 2) y evalúe sus resultados con imágenes de su preferencia, también pruebe diferentes valores de X. Código:

```

1     img = cv2.imread("4.jpg")
2     img2 = cv2.imread("7.jpg")
3     img=cv2.resize(img,(400,300))
4     img2=cv2.resize(img2,(400,300))
5     x=0.7
6     for i in range(img.shape[0]):
7         for j in range(img.shape[1]):
8             img[i,j]=x*img[i,j]+(1-x)*img2[i,j]
9     cv2.imshow("out.jpg",img)

```

Explicación:

Primero cargaremos nuestras dos imágenes, luego es importante igualar su tamaño, en este caso les damos un tamaño fijo para evitar obtener resultados muy grandes o muy pequeños, es decir sea siempre el mismo tamaño independientemente de las imágenes que se carguen. Luego aplicamos la fórmula de "blending" con nuestra variable X, que irá cambiando según nosotros veamos conveniente.

Resultado:



(a) Paisaje



(b) Espacio

Figura 20: Imágenes iniciales



(a) $x = 0.2$

(b) $x = 0.5$

(c) $x = 0.7$

Figura 21: Resultados

■ Ejercicio 5:

Finalmente, agregue los operadores de multiplicación, división y blending a su software de procesamiento de imágenes.

Agregamos los botones de multiplicación, división y blending a nuestra barra de acciones

Histogram	Op. Exponencial	Op. Logaritmico	Op. raiz	Raice to power	Contrast Stretching	Thresholding	Cascada	Suma
Suma Color	Resta	Multiplicación	División y escalamiento	División para detectar movimiento	Blending			

Subir Escoga un archivo

Subir Escoga un segundo archivo

Se muestra a continuación la multiplicación por una constante, en este caso es 5

Histogram	Op. Exponencial	Op. Logarítmico	Op. raíz	Raíz to power	Contrast Stretching	Thresholding	Cascada	Suma
Suma Color	Resta	Multiplicación	División y escalamiento	División para detectar movimiento	Blending			
Subir	Escoga un archivo							
Subir	Escoga un segundo archivo							
Recortar								



El resultado en nuestra pagina sera el siguiente



Ahora probaremos con el blending, seleccionamos dos imágenes

Blending

Histogram Op. Exponencial Op. Logarítmico Op. raiz Raíz to power Contrast Stretching Thresholding Cascada Suma
Suma Color Resta Multiplicación División y escalamiento División para detectar movimiento Blending

Subir Escoga un archivo Browse

Subir Escoga un segundo archivo Browse

Recortar



El resultado sera el siguiente



4. Conclusiones

- La división trabaja de la misma manera que la resta, esto se puede notar en las aplicaciones que se le puede dar como detectar cambios entre fotogramas. Pero se obtiene mejores resultados con la división.
- Una aplicación interesante de la adición entre imágenes es el "blending", que podría traducirse entre juntar imágenes jugando con la opacidad de cada una. Esto podría ser aplicado en edición de fotos.
- La multiplicación de imágenes es muy similar a la suma solo se aplica el operador a nivel de píxeles entre dos imágenes o con una constante y obtenemos nuestro resultado.
- Para que una imagen pueda ser visualizada en la división entre dos imágenes, se le tiene que multiplicar al resultado una constante. Dependiendo de dicha constante se obtendrá un mejor o peor resultado de visualización. Esto se puede observar al aplicar dicho operador en la segmentación de caracteres.
- En el caso del blending se podría asumir que la variable es el porcentaje que queramos ver de la foto, y si agregáramos mas imágenes, las variables que multipliquen a las imágenes, tendrán que sumar 1.
- Al momento de dividir dos imágenes, como el resultado sera demasiado bajo, no se podrá visualizar bien, para mejorar esto se puede utilizar la misma técnica de escalado usada en el contrast stretching.

5. PRACTICA 8

- Implemente el operador AND con las imágenes de la Figura 1 para segmentar el objeto que aparece en ambas imágenes (intersección).

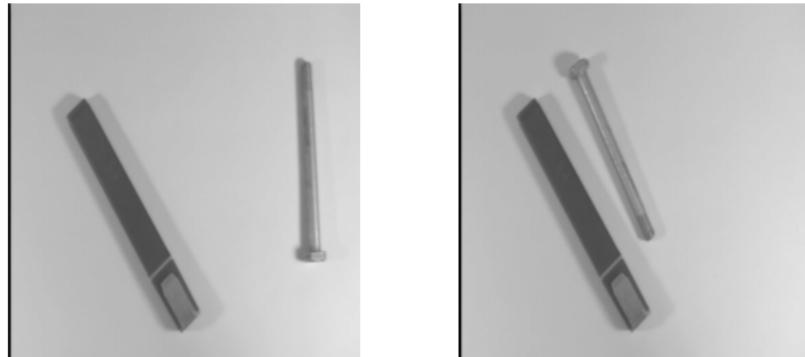


Figura 22: Figuras de muestra.

Código:

Listing 6: Ejercicio 1

```
1 import cv2
2 import numpy as np
3 img1=cv2.imread('log_3.png',0)
4 img2=cv2.imread('log_4.png',0)
5 img3=cv2.imread('blank1.png',0)
6 f,c=img1.shape
7 for i in range(f):
8     for j in range(c):
9         img3[i][j]=np.bitwise_and(img1[i][j], img2[i][j])
10 cv2.imshow('res',img3)
```

Explicación: Para el uso de estos operadores lógicos se hará uso de Bitwise una utilidad de la librería Numpy que permite realizar operaciones lógicas. Aparte de todo ello haremos uso de la librería OpenCV para trabajar a nivel de imágenes. Después de importar las librerías se abrirán 3 imágenes, en escala de grises. Las 2 primeras que son las imágenes de muestra y una tercera que estará en blanco y que servirá como el lienzo del resultado final, a continuación extraemos el largo y el ancho de nuestra imagen para recorrerla y aplicar el operador lógico AND a nivel de píxel entre ambas imágenes.

Resultado:



Figura 23: Uso del operador lógico AND.

- Implemente el operador OR con las imágenes de la Figura 1 para unir los objetos que aparecen en las imágenes (fusión). Código:

Listing 7: Ejercicio 2

```

1  import cv2
2  import numpy as np
3  img1=cv2.imread('log_3.png',0)
4  img2=cv2.imread('log_4.png',0)
5  img3=cv2.imread('blank1.png',0)
6  f,c=img1.shape
7  for i in range(f):
8      for j in range(c):
9          img3[i][j]=np.bitwise_or(img1[i][j], img2[i][j])
10 cv2.imshow('res',img3)

```

Explicación: El procedimiento es el mismo para este caso, se emplean las librerías con las que se implemento el uso del operador AND, y haremos uso de Bitwise, utilidad de la librería Numpy. Posteriormente trabajamos con 3 imágenes las 2 primeras son las de muestra y serán abiertas en escala de grises, la tercera será la imagen en la que irá el resultado. Seguido se le extraerá el largo y ancho para finalmente hacer uso del operador lógico OR a nivel de píxeles entre ambas imágenes.

Resultado:



Figura 24: Uso del operador lógico OR.

- Implemente el operador XOR con las imágenes de la Figura 1 para detectar los cambios.
Código:

Listing 8: Ejercicio 3

```

1      import cv2
2      import numpy as np
3      img1=cv2.imread('log_3.png',0)
4      img2=cv2.imread('log_4.png',0)
5      img3=cv2.imread('blank1.png',0)
6      f,c=img1.shape
7      for i in range(f):
8          for j in range(c):
9              img3[i][j]=np.bitwise_xor(img1[i][j], img2[i][j])
10
11      cv2.imshow('res',img3)

```

Explicación:

Por último tenemos al operador XOR, el procedimiento es el mismo para este caso, se emplean las librerías con las que se implementó el uso de los operadores AND y OR, y se hará uso de Bitwise, utilidad de la librería Numpy. Posteriormente trabajamos con 3 imágenes las 2 primeras son las de muestra y serán abiertas en escala de grises, la tercera será la imagen en la que irá el resultado. Seguidamente se le extraerá el largo y ancho para finalmente hacer uso del operador lógico XOR a nivel de píxeles entre ambas imágenes.

Resultados:

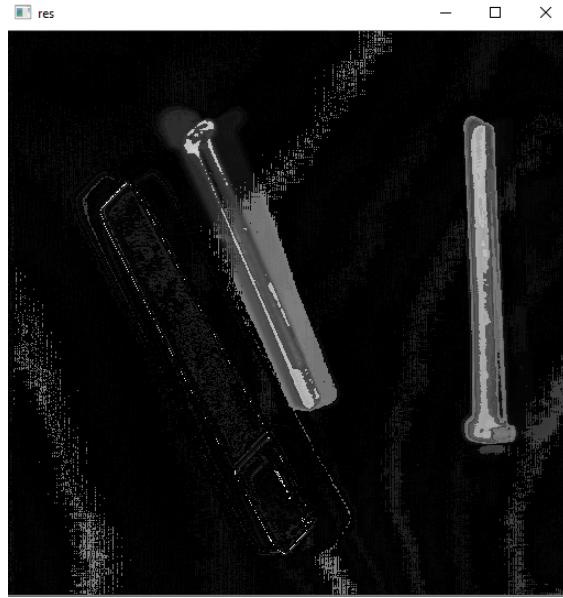


Figura 25: Uso del operador lógico XOR.

- Finalmente, agregue los operadores de AND, OR y XOR a su software de procesamiento de imágenes. Se muestra a continuación los botones agregados para dichos operadores:



Figura 26: Botones de operadores lógicos agregados

Se muestra a continuación las dos imágenes con las que se probará los operadores lógicos en nuestra interfaz gráfica:

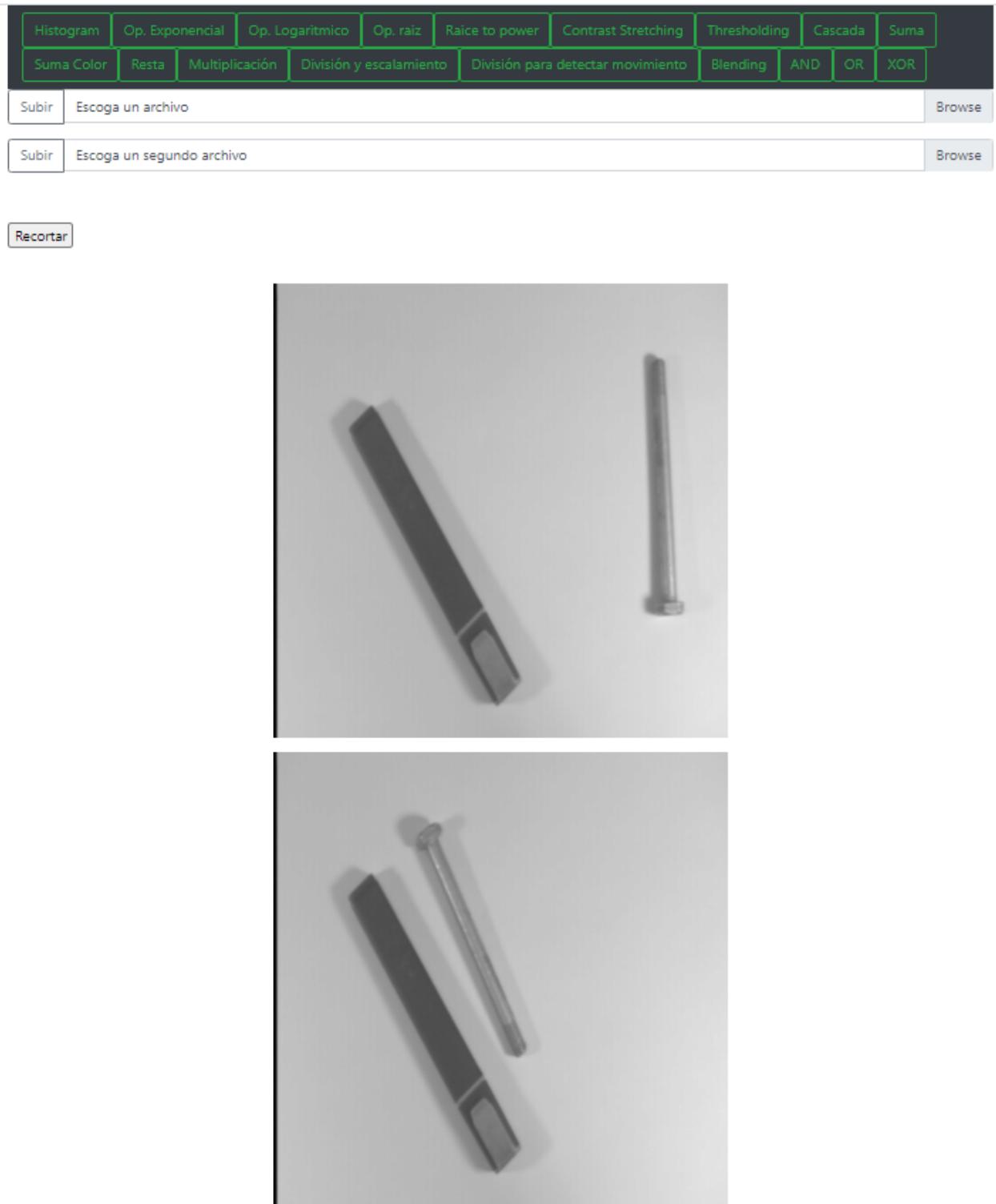


Figura 27: Imágenes a usar para los operadores lógicos

Ahora se probará el operador AND para dichas imágenes:



Figura 28: Resultado con el operador AND en la interfaz

Se muestra el resultado con el operador OR:

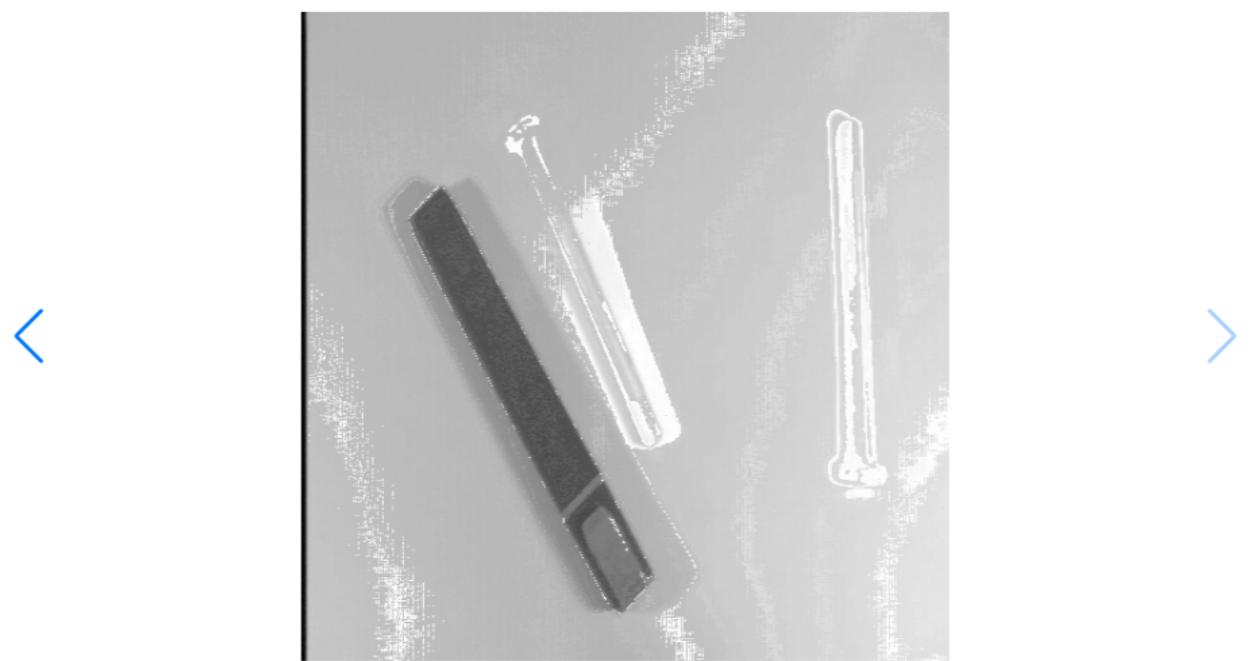


Figura 29: Resultado con el operador OR en la interfaz

Por último se muestra el resultado con el operador XOR:

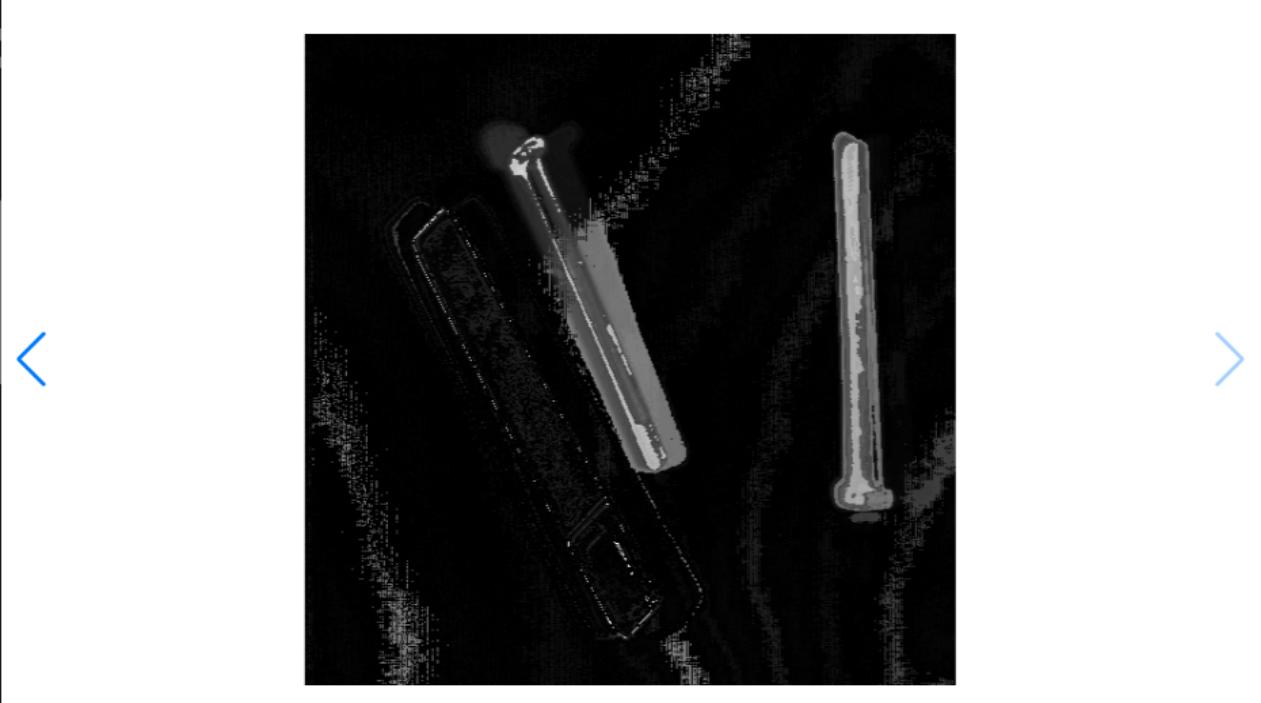


Figura 30: Resultado con el operador XOR en la interfaz

6. Conclusiones

- Bitwise trabaja de forma binaria los operadores lógicos, es decir cada valor decimal de cada píxel es convertido a binario y se hace uso del operador lógico deseado. En este caso se emplea entre los valores de los píxeles a nivel binario de dos imágenes.
- Para que funcionen mejor estos operadores se recomienda hacer la binarización de imágenes, usando la técnica del Thresholding, de esa forma se apreciaría mejor los resultados.
- Podemos usar el operador and para hacer una intersección de imágenes y el operador or para hacer la unión de dos imágenes.
- Podemos observar que los resultados al hacer una operación con Bitwise es diferente al usar la operación AND del lenguaje de programación ,en este caso Python, puesto que el de Python trabaja a nivel Booleano.

7. Link del repositorio

- <https://github.com/ZzzandyzzZ/Computacion-grafica-grupal/tree/master/Add%20Sub/prueba>
- <https://github.com/ZzzandyzzZ/Computacion-grafica-grupal/tree/master/examen%201/grafi>