



# UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

## ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

### COMPUTACION GRAFICA

*Proyecto Final*

Alumnos:

Chayña Batallanes Josnick  
Perez Rodriguez Angelo Aldo  
Pucho Zevallos Kelvin Paul  
Vilcapaza Flores Luis Felipe  
Sihuinta Perez Luis Armando

Julio 2021

# Índice

	<b>1</b>
<b>1. Implementación de un Scanner usando opencv</b>	<b>2</b>
1.1. Descripción . . . . .	2
1.2. Deteccion de Bordes . . . . .	2
1.3. Componentes Conectados . . . . .	4
1.4. Transformaciones de Perspectivas . . . . .	6
1.5. Procesamiento de imágenes . . . . .	6
1.6. Codigo Completo . . . . .	14
<b>2. Link de los códigos en github</b>	<b>20</b>

# 1. Implementación de un Scanner usando opencv

## 1.1. Descripción

Para construir un escáner de documentos con OpenCV se hizo realizando en cuatro pasos:

- Método para la detección de bordes
- Método que utiliza los bordes para encontrar los contornos que representa el documento que se esta escaneando
- Método que aplica una transformación de perspectiva para obtener la vista completa del documento en una ventana
- Métodos que aplican procesamiento de imagenes para mejorar la calidad de escaneo

## 1.2. Deteccion de Bordes

- **Aplicar Filtro gaussiano:**

En este la aplicacion de este metodo puede funcionar para imagenes a color o escala de grises. Entonces primeramente suavizamos la imagen con un filtro gaussiano. Este para disminuir el ruido de la imagen o valores atipicos. Se uso la siguiente linea de codigo para eso:

---

```
1 imgBlur = cv2.GaussianBlur(imgGray, (5,5),1)
```

---

- **Aplicar el método Canny:**

En el metodo canny utiliza un proceso de varias etapas para detectar una amplia gama de bordes en la imagenes donde algunas de estas etapas son: - La detección de bordes de canny elimina primero el ruido de la imagen mediante la suavizacion.

- El detector Canny se basa en la magnitud del gradiente de una imagen suavizada

- El algoritmo Canny rastrea a lo largo de estas regiones y suprime cualquier píxel que no esté en el máximo (supresión no máxima).

- Y posteriormente realiza un doble Thresholding

Se uso la siguiente linea de codigo para utilizar el metodo canny:

---

```
1 imgBlur = imgCanny = cv2.Canny(imgBlur,200,200)
```

---

- **Aplicar el Operador Morfologico Closing:** El Operador Morfologico Closing que es el reverso de la Opening, la dilatación seguida de la erosión. Es útil para cerrar pequeños agujeros dentro de los objetos de primer plano, o pequeños puntos negros en el objeto.

- **Erocion:**

Este método se encarga de que todos los píxeles cercanos a los límites se descartan en función del tamaño del núcleo. Así, el grosor o el tamaño del objeto en primer

plano disminuye o simplemente la región blanca disminuye en la imagen. Es útil para eliminar pequeños ruidos blancos. Para este metodo se uso un kernel de unos de tamaño 5x5, iterando 2 veces.

Se uso la siguiente linea de codigo para eso:

---

```
1         imgDilate = cv2.dilate(imgCanny ,kernel,iterations = 2)
```

---

- **Dilate:**

Es lo opuesto a la erosion, Aumenta la región blanca en la imagen o el tamaño del objeto en primer plano. Para este metodo se uso un kernel de unos de tamaño 5x5 iterando una vez.

Se uso la siguiente linea de codigo para eso:

---

```
1         imgErode = cv2.erode(imgDilate,kernel,iterations = 1)
```

---

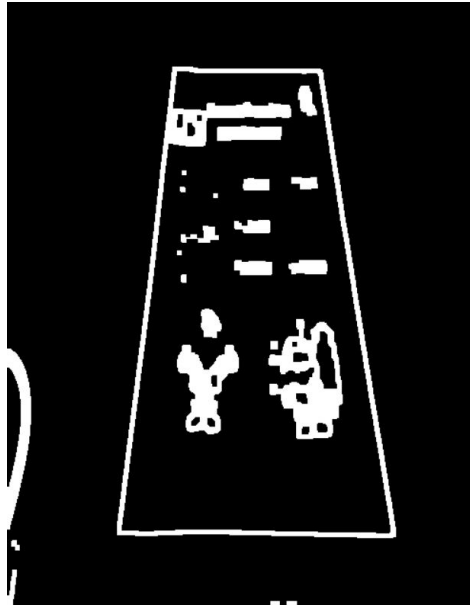
- **Método Completo para la deteccion de Bordes:**

---

```
1     def MorphologyOperator (img):
2
3         imgGray = cv2.cvtColor (img,cv2.COLOR_BGR2GRAY)
4
5         #Aplicando filtrado gaussiano para suavisar la imagen
6         imgBlur = cv2.GaussianBlur (imgGray, (5,5),1)
7         #Aplicando algoritmo de deteccion de bordes canny
8         imgCanny = cv2.Canny (imgBlur,200,200)
9
10        #Aplicando operador Closing
11        kernel = np.ones((5,5)) # Kernel de unos de tamaño 5X5
12        imgDilate = cv2.dilate(imgCanny ,kernel,iterations = 2)
13        imgErode = cv2.erode(imgDilate,kernel,iterations = 1)
14        return imgErode
```

---

- **Captura:**



(a) Imagen 1

Figura 1: Deteccion de bordes

### 1.3. Componentes Conectados

Este método verifica si cada pixel vecino está conectado con el pixel actual si es así, se coloca una etiqueta y si el pixel actual no está conectado se coloca otra etiqueta mayor a uno en la última etiqueta puesta. Si en el caso fuera un pixel 0 se sobreentiende que no está conectada.

Para este caso se usó el método `findContours()` el cual encuentra los contornos de las manchas conectadas.

Se usó la siguiente línea de código para eso:

---

```
1 contours, _ = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
```

---

Para este caso se especificó los siguientes métodos:

- Modo de recuperación de contornos `cv2.RETR_EXTERNAL`: recupera sólo los contornos exteriores extremos

- Método de aproximación del contorno `cv2.CHAIN_APPROX_NONE`: Almacena todos los puntos del contorno. Primero para calcular el contorno adecuado del documento se debe seguir el siguiente procedimiento:

- Área de cada contorno de la imagen
- Perímetro de cada contorno de la imagen, se utilizó la función `cv2.arcLength()`. El segundo argumento especifica si la forma es un contorno cerrado (si se pasa como `True`) o simplemente una curva.

- Aproximación de cada contorno de la imagen  
Aproxima una forma de contorno a otra forma con menos número de vértices, dependiendo de la precisión que se especifique. El segundo argumento de esta función se denomina *épsilon*, que es la distancia máxima entre el contorno y el contorno aproximado. Es un parámetro de precisión.

Para este método se usó la siguiente línea de código para eso:

---

```

1     contours=sorted(contours,key=cv2.contourArea,reverse=True)
2
3     perimeter=cv2.arcLength(c,True) #longitud de cada contorno
4     approx=cv2.approxPolyDP(c,0.02*perimeter,True) #retorna los puntos de los puntos

```

---

Las variables *perimeter* y *approx* son ejecutados dentro de un bucle que registra cada área del contorno donde encontraremos un contorno específico que será nuestro documento que tendré 4 esquinas.

Y por último graficamos nuestro contorno.

---

```

1     cv2.drawContours(setImgContour, [approx], 0, (250,94,48),3)

```

---

**Captura:**



(a) Imagen frontal

(b) Imagen posterior

Figura 2: Gráfico de los contornos

## 1.4. Transformaciones de Perspectivas

Este método es útil para alinear la imagen correctamente. Para esto se da a conocer los puntos del contorno y los puntos de destino para la perspectiva. Estos puntos por medio de la siguiente función:

```
1 M = cv2.getPerspectiveTransform(pts1,pts2)
```

Se obtendrá la matriz de transformación el cual se multiplicará a cada posición del pixel de la imagen y será mostrada en la imagen destino. Y para ejecutar la deformación se usa el siguiente código:

```
1 imgResult = cv2.warpPerspective(img,M,(widthImg,heightImg))
```

**Captura:**



Figura 3: Resultado de la Transformación de la Perspectiva

## 1.5. Procesamiento de imágenes

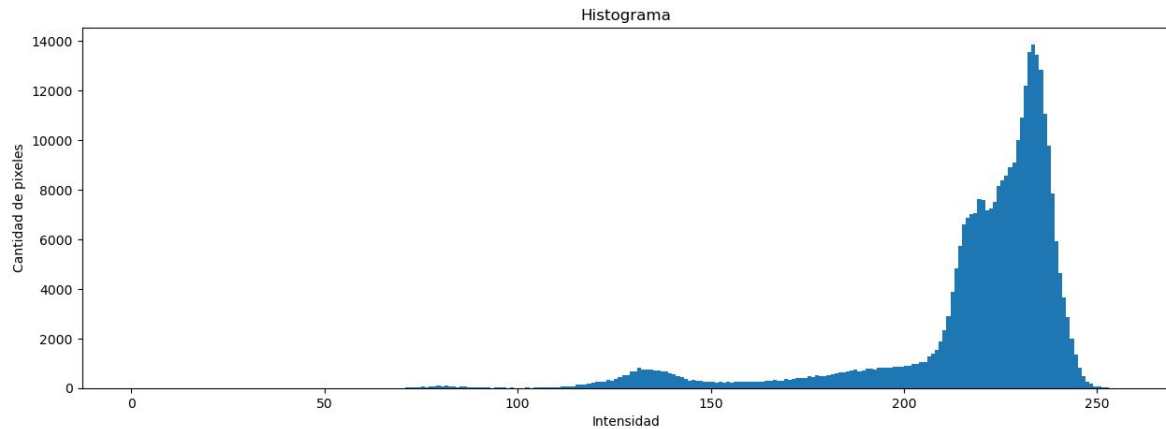
Luego de aplicar las anteriores operaciones, se obtiene la siguiente imagen:

Se procedió a aplicar las siguientes operaciones para dar un resultado que resalte el escaneo.

## ■ Aplicando Thresholding

Para segmentar se uso una umbral de 210

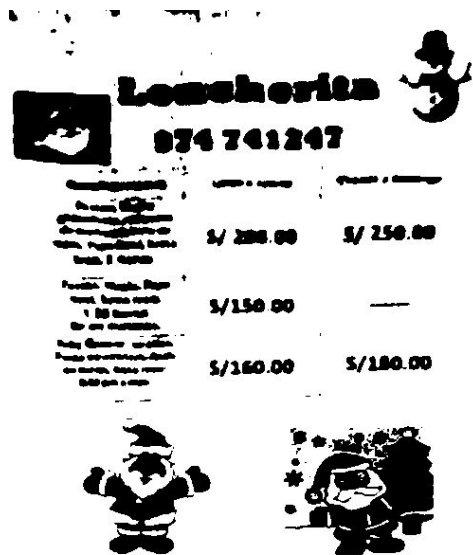
- Histograma



(a) Histograma 1

Figura 4: Histograma

- Thresholding (segmentación de la imagen)



(a) Imagen 1

Figura 5: Thresholding

## ■ Aplicando Histogram Equalization

La imagen aplicando con la Ecuación de Histograma con los colores de la imagen completa: Para corregir y tener una buena ecualización es recortar una parte de imagen original para conservar sus colores





(a) Imagen Ecualizada

Figura 6: Histogram Equalization



(a) Imagen Recortada

(b) Imagen Ecualizada

Figura 7: Histogram Equalization con imagen recortada

■ **Aplicando Exponential Operator:**

Con constantes de:

$$c = 20$$

$$b = 1.01$$

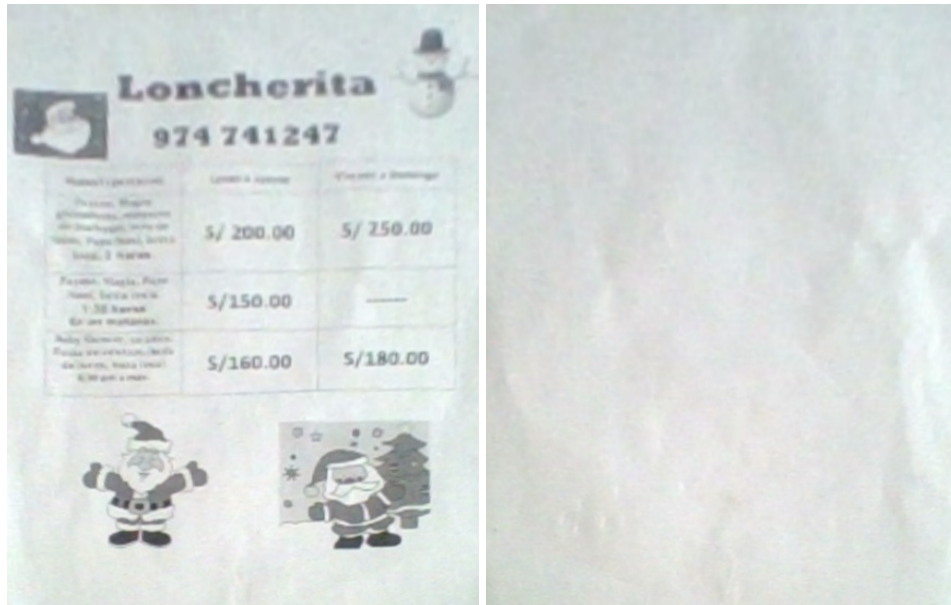




- **Aplicando Sustracción de imágenes:**

Esta Sustraccion de imagen es usado para quitar algunas manchas de la imagen originla.

- **Imagenes Escaneadas:**



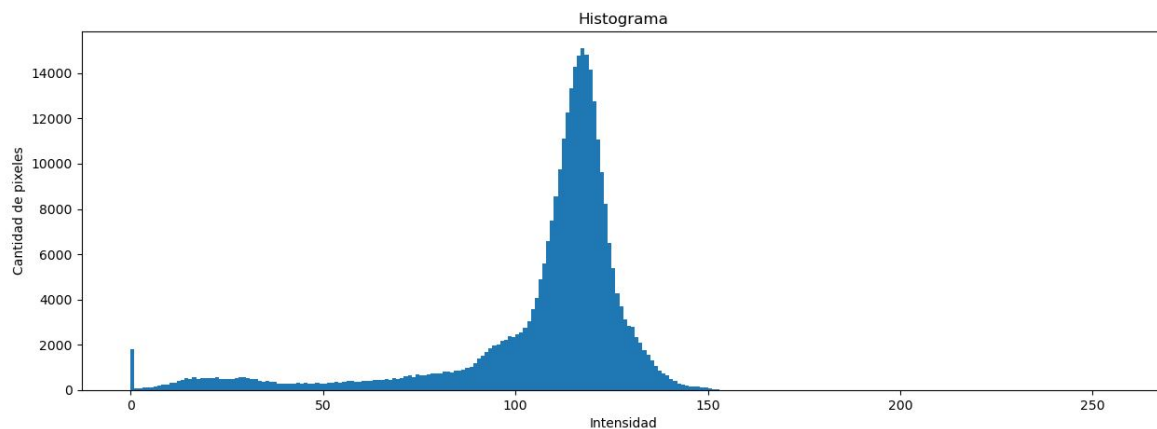
(a) Imagen 1

(b) Imagen 2

Figura 12: Imagenes Originales

- **Histograma de la Sustracción de la imagen:**

Se uso una umbral de 90

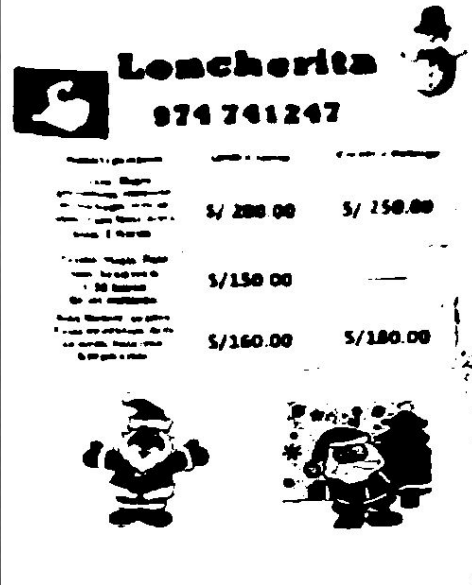


(a) Histograma

Figura 13: Resultado del Histograma

- **Resultados de la Imagen despues de aplicar la Sustraccion y Thresholding:**

Para la sustraccion se uso una constante de  $c = 110$  para aumentar la claridad de la imagen



(a) Imagen despues de aplicar la Sustracción (b) Imagen despues de aplicar Thresholding

Figura 14: Resultados

#### ■ Aplicando División de imagenes:

Esta División de imagen es usado para quitar algunas manchas de la imagen originla.

#### • Imagenes Escaneadas:

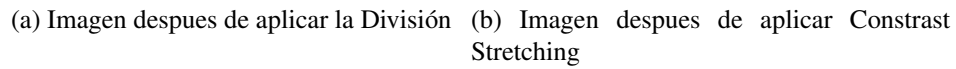


(a) Imagen 1


(b) Imagen 2

Figura 15: Imagenes Originales

Para la división se uso la formula de escala de colores para aumentar la claridad de la imagen

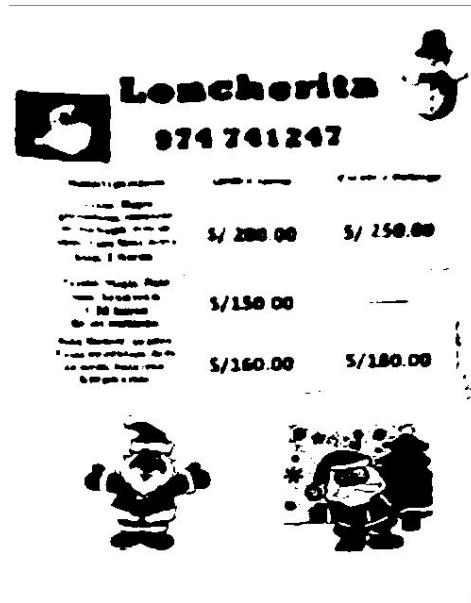


- **Histograma de la Divisiòn de la imagen:**



El histograma muestra la distribución de la intensidad de los píxeles. El eje horizontal (Intensidad) va de 0 a 255, y el eje vertical (Cantidad de píxeles) va de 0 a 25,000. La distribución es unimodal y sesgada a la izquierda, con un pico principal entre 180 y 200 unidades de intensidad.

Figura 17: Resultado del Histograma



(a) Imagen despues de aplicar Thresholding

Figura 18: Resultado

## 1.6. Codigo Completo

```

1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4  import mapper
5  import Thresholding as Ts
6  import HistogramEqualization as He
7  import ExponentialOperator as Oe
8  import LogarithmOperator as Ol
9  import raise_to_the_power_Operator as ORp
10 import ContrastStretching as Cs
11 import Subtraction as Subt
12 import Division as Div
13
14 widthImg = 500
15 heightImg = 640
16
17 cap = cv2.VideoCapture(0) #0 por defecto camera
18 cap.set(3,1040*2) # altura
19 cap.set(4,480*2) # ancho
20 cap.set(10,150) #brillo
21
22 =====
23 # aplicando OPERADORES MORFOLOGICOS
24 =====
25 def MorphologyOperator(img):

```

```

26
27     imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
28
29     #Aplicando filtrado gaussiano para suavisar la imagen
30     imgBlur = cv2.GaussianBlur(imgGray,(5,5),1)
31     #Aplicando algoritmo de deteccion de bordes canny
32     imgCanny = cv2.Canny(imgBlur,200,200)
33
34     #Aplicando operador Closing
35     kernel = np.ones((5,5)) # Kernel de unos de tamaño 5X5
36     imgDilate = cv2.dilate(imgCanny ,kernel,iterations = 2)
37     imgErode = cv2.erode(imgDilate,kernel,iterations = 1)
38
39
40     # cv2.imshow("image Canny",imgCanny)
41     # cv2.imshow("image Dilate",imgDilate)
42     # cv2.imshow("image Erode", imgErode)
43
44     return imgErode
45
46     #=====
47     # aplicando Connected Components: Metodo FindContours
48     #=====
49
50     def ConnectedComponents(img,setImgContour):
51         findContour = np.array([])
52         contours, _ = cv2.findContours(img,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
53         contours=sorted(contours,key=cv2.contourArea,reverse=True)
54
55         #el bucle extrae los contornos de los límites de la página
56         for c in contours:
57             perimeter=cv2.arcLength(c,True)#longitud de cada contorno
58             approx=cv2.approxPolyDP(c,0.02*perimeter,True)#retorna los puntos de los puntos
59
60             if len(approx)==4:
61                 color_ = (150,150,150)
62                 cv2.drawContours(setImgContour, [approx], 0, (250,94,48),3)
63                 puntos = approx.reshape((4,2))
64                 cv2.circle(setImgContour, tuple(puntos[0]), 15, color_, 5)
65                 cv2.circle(setImgContour, tuple(puntos[1]), 15, color_, 5)
66                 cv2.circle(setImgContour, tuple(puntos[2]), 15, color_, 5)
67                 cv2.circle(setImgContour, tuple(puntos[3]), 15, color_, 5)
68                 findContour=approx
69                 break
70
71         # cv2.drawContours(imgContour,findContour,-1,(0,255,0),20)
72         return findContour

```



```

73
74 #=====
75 # aplicando Transformacion de Perspectiva
76 #=====
77
78
79 def PerspectiveTransformation(img,Contour):
80     if(len(Contour) == 0): return img
81     #Aplicando ordenamiento a los puntos
82     ptsContour = mapper.mapp(Contour)
83
84     #Deformando la imagen real a la ventana de la salida deseada
85     pts1 = np.float32(ptsContour)
86     pts2 = np.float32([[0,0],[widthImg,0],[0,heightImg],[widthImg,heightImg]]) # windows
87
88     M = cv2.getPerspectiveTransform(pts1,pts2)
89     imgResult = cv2.warpPerspective(img,M,(widthImg,heightImg))
90     # M = cv2.getAffineTransform(pts1,pts2)
91     # imgResult = cv2.warpAffine(img,M,(widthImg,heightImg))
92
93     #recortando errores
94     imgCropped = imgResult[20:imgResult.shape[0]-20,20:imgResult.shape[1]-20]
95     imgCropped = cv2.resize(imgCropped,(widthImg,heightImg))
96     return imgCropped
97
98 def hconcat_resize_min(im_list, interpolation=cv2.INTER_CUBIC):
99     for i in range(len(im_list)):
100         im_list[i] = cv2.resize(im_list[i],(600,600))
101     h_min = min(im.shape[0] for im in im_list)
102     im_list_resize = [cv2.resize(im, (int(im.shape[1] * h_min / im.shape[0])), h_min), im
103                       for im in im_list]
104     return cv2.hconcat(im_list_resize)
105
106
107 def concat_tile(im_list_2d):
108     return cv2.vconcat([cv2.hconcat(im_list_h) for im_list_h in im_list_2d])
109
110
111 global ImgResult
112
113 while True:
114     success, img = cap.read()
115     img = cv2.resize(img,(widthImg,heightImg))
116     imgEdge = MorphologyOperator(img)
117
118     imgContour = img.copy()
119     Contour = ConnectedComponents(imgEdge,imgContour)

```

```

120     # print(biggest)
121     imgPersp = PerspectiveTransformation(img,Contour)
122
123     #Mostrando deteccion de bordes
124     cv2.imshow("Imagen con Bordes",imgEdge)
125     #Mostrando contornos
126     cv2.imshow("Imagen con Contorno",imgContour)
127     #Mostrando Resultados Scanner
128     cv2.imshow("Imagen Resultado",imgPersp)
129     #Camera Real
130     # cv2.imshow("Camera Principal",img)
131
132     key = cv2.waitKey(1) & 0xFF
133
134     if key == ord('q'):
135         ImgResult = imgPersp
136         break
137     elif key == ord('z'):
138         cv2.imwrite("front.jpg", imgPersp)
139     elif key == ord('x'):
140         cv2.imwrite("back.jpg", imgPersp)
141
142     #=====
143     # Lectura del imagenes escaneadas
144     #=====
145     img1 = cv2.imread("front.jpg",cv2.IMREAD_GRAYSCALE)
146     cv2.imshow("Imagen Original 1", img1)
147
148     img2 = cv2.imread("back.jpg",cv2.IMREAD_GRAYSCALE)
149     cv2.imshow("Imagen Original 2", img2)
150
151     #=====
152     # Aplicando Thresholding
153     #=====
154     #tablas para los histogramas
155     # f, (ax1) = plt.subplots(1, 1, sharey=True,figsize=(15, 5))
156     # ax1.hist(img1.ravel(),256,[0,256])
157     # ax1.set_title("Histograma")
158     # ax1.set_xlabel('Intensidad')
159     # ax1.set_ylabel('Cantidad de pixeles')
160
161     imgThresold = Ts.getThresholding(img1,210)#210
162     cv2.imshow("Imagen Thresolding", imgThresold )
163
164     #=====
165     # Aplicando Histogram Equalization
166     #=====

```

```

167 #Recortar una imagen
168 imageOut = img1[450:550,300:450]
169 imgEqualized = He.getHistogramEqua(img1,imageOut)
170 cv2.imshow("Imagen Equalizada", imgEqualized)
171
172 # cv2.imshow("Imagen recortada", imageOut)
173
174 #=====
175 # Aplicando Exponential Operator
176 #=====
177 imgExpn = Oe.ExponentialOperator(img1)
178 cv2.imshow("Operador Exponencial", imgExpn)
179
180
181 #=====
182 # Aplicando Logarithm Operator (opcional)
183 #aca se aplico de exponencial a logaritmico
184 #=====
185 imgLog = Ol.LogarithmOperator(imgExpn)
186 cv2.imshow("Operador Logarithm", imgLog)
187
188 #=====
189 # Aplicando Raise to the power Operator
190 #
191 #=====
192 imgRpower = ORp.raise_to_the_power_Operator(img1)
193 cv2.imshow("Operador Raise to the Power", imgRpower)
194
195 #=====
196 # Aplicando Constrast Stretching
197 #=====
198 # por escala de grises en 8 bit entonces
199 arrayImg = img1.ravel()
200 sortArray = np.sort(arrayImg)
201 # 0% y el 100 %
202 min_ = int(len(sortArray) * 0.0)
203 max_ = int((len(sortArray) * 1.0) - 1)
204 c = sortArray[min_]
205 d = sortArray[max_]
206 imgConstrast = Cs.EquationContrastStretching(img1.astype(int),int(c),int(d))
207
208 cv2.imshow("Constrast Stretching", imgConstrast)
209
210
211 #=====
212 # Aplicando Sustraccion de imagenes
213 #=====

```

```

214 imgResultSust = Subt.ImagesSubtraction(img1,img2)
215 cv2.imshow("Resultado Sustraccion", imgResultSust)
216
217
218 threshold = 90
219 imgResultSust = Ts.getThresholding(imgResultSust,threshold)
220 cv2.imshow("Sustraccion Thresolding", imgResultSust)
221
222 #=====
223 # Aplicando Division de imagenes
224 #=====
225 imgResultDiv = Div.ImagesDivision(img1,img2)
226 cv2.imshow("Resultado Division", imgResultDiv)
227
228 # por escala de grises en 8 bit entonces
229 arrayImg = imgResultDiv.ravel()
230 sortArray = np.sort(arrayImg)
231 # 0% y el 100 %
232 min_ = int(len(sortArray) * 0.0)
233 max_ = int((len(sortArray) * 1.0) - 1)
234 c = sortArray[min_]
235 d = sortArray[max_]
236 imgConstrast = Cs.EquationContrastStretching(imgResultDiv.astype(int),int(c),int(d))
237
238 cv2.imshow("Division Constrast Stretching", imgConstrast)
239
240 threshold = 164
241 imgResultDiv = Ts.getThresholding(imgConstrast,threshold)
242 cv2.imshow("Division Thresolding", imgResultDiv)
243
244 f, (ax1) = plt.subplots(1, 1, sharey=True,figsize=(15, 5))
245 ax1.hist(imgConstrast.ravel(),256,[0,256])
246 ax1.set_title("Histograma")
247 ax1.set_xlabel('Intensidad')
248 ax1.set_ylabel('Cantidad de pixeles')
249
250 # im1_s = cv2.resize(im1, dsize=(0, 0), fx=0.5, fy=0.5)
251 # im_tile = concat_tile([[ , , ]])
252 # cv2.imwrite('data/dst/opencv_concat_tile.jpg', im_tile)
253
254
255
256 plt.show()
257 cv2.waitKey(0)
258 cv2.destroyAllWindows()

```

---

## **2. Link de los códigos en github**

`https://github.com/kpzaolod6000/Graphics-Computing/tree/main/  
parcial\_3/lab\_10/examen\_3`