LABORATORIO III

Luis Ariza Ramos
Erick Barros Escorcia
Universidad del Magdalena, Ingeniería Electrónica,
Colombia
Luisarizaar@unimagdalena.edu.co
Erickbarrosde@unimagdalena.edu.co

Abstract **In this laboratory practice, the filtering of images acquired by means of a drone was carried out. With the purpose of filtering and testing our knowledge obtained in classes, in the following report we will present two images such as the one of the panels located on the roof of the the teachers' building and the one by the lake. Where you search through the Colab tool, filter each image and then count them by an algorithm to define how many are found.**

Resume En está practica de laboratorio, se realizó el filtrado de imágenes abquiridas por medio de un dron, Con el proposito de filtrar y poner a prueba nuestros conocimientos obtenidos enc lases en el siguiente informe le presentaremos dos iamgenes como la de paneles ubicados en el techo del edificio de docentes y la del lago En donde se busca por medio de la herramienta Colab, filtrar cada imagen y luego contarlos por un algoritmo para definir cuantos se encuentran.
Keywords: *RGB, filter, Colab.*

I. MARCO TEORICO

Un espacio de color es un sistema de interpretación del color, es decir, una organización específica de los colores en una imagen o video. Depende del modelo de color en combinación con los dispositivos físicos que permiten las representaciones reproducibles de color, por ejemplo, las que se aplican en señales analógicas (televisión a color) o representaciones digitales. Un espacio de color puede ser arbitrario, con colores particulares asignados según el sistema y estructurados matemáticamente.

Un modelo de color es un modelo matemático abstracto que describe la forma en la que los colores pueden representarse como tuplas de números, normalmente como tres o cuatro valores o componentes de color (por ejemplo, RGB y HSV son modelos de color). Sin embargo, un modelo de color que no tiene asociada una función de mapeo a un espacio de color absoluto es más o menos un sistema de color arbitrario sin conexión a un sistema de interpretación de color.
Se puede crear un amplio rango de colores mediante pigmentos de colores primarios (cian (C), magenta (M),segmentación se usa tanto para localizar objetos como para amarillo (Y), y negro (K)). Otra manera de crear los mismos encontrar sus bordes dentro de una imagen. El resultado de la colores es usando su matiz (eje X), su saturación (eje Y), y su segmentación de una imagen es un conjunto de segmentos que brillo (eje Z). A esto se le llama modelo de color HSV.

El modelo de color RGB está implementado de formas diferentes, dependiendo de las capacidades del Sistema utilizado. De lejos, la implementación general más utilizada es la de 24 bits, con 8 bits, o 256 niveles de color discretos por espacio de color basado en ese modelo RGB de 24 bits está limitado a un rango de 256×256×256 ≈ 16,7 millones de colores. Algunas implementaciones usan 16 bits por componente para un total de 48 bits, resultando en la misma gama con mayor número de colores. Esto es importante cuando se trabaja con espacios de color de gama amplia (donde la mayoría de los colores se localizan relativamente juntos), o cuando se usan consecutivamente un amplio número de algoritmos de filtrado digital. El mismo principio se aplica en cualquier espacio de color basado en el mismo modelo de color, pero implementado en diferentes profundidades de color.

En cuanto a la conversión del espacio de color es la traducción de la representación de un color de una base a otra. Esto ocurre normalmente en el contexto de convertir una imagen representada en un espacio de color a otro espacio de color, teniendo como objetivo que la imagen convertida se parezca lo más posible a la original [1]. Además, en Python y openCV se puede hacer un cambio de espacio de color como de RGB a gris, BGRA, YCrCb, XYZ, HSV, Lab, Luv, HLS, YUV usando la función

cvtColor, esta es una función de openCV, con más de
150 espacios disponibles, que convierte imágenes de un espacio de color a otro [2].

La segmentación es uno de los problemas generales del campo de la visión artificial y consiste en dividir una imagen digital en varias regiones (grupos de píxeles) denominadas segmentos. Más concretamente, la segmentación es un proceso de clasificación por píxel que asigna una categoría a cada píxel de la imagen analizada. Este problema general se divide en problemas especializados, dando lugar por ejemplo a:

- segmentación por color
- segmentación por texturas
- superpíxel
- segmentación semántica

Además, cada problema especializado le otorga un significado propio a las categorías que se usan en la clasificación de los píxeles. Uno de los casos más elementales de segmentación es la umbralización, un tipo particular de segmentación por color con solo dos categorías: claro y oscuro. Cada píxel se clasifica como claro u oscuro comparando su intensidad con una intensidad de referencia dada denominada umbral. El objetivo de la segmentación es localizar regiones con significado. La segmentación se usa tanto para localizar objetos como para encontrar sus bordes dentro de una imagen. El resultado de la segmentación de una imagen es un conjunto de segmentos que cubren toda la imagen sin superponerse. Se puede representar como una imagen de etiquetas (una etiqueta para cada píxel) o como un conjunto de contornos [3].

La detección de bordes es una herramienta fundamental en el procesamiento de imágenes y en visión por computadora, particularmente en las áreas de detección y extracción de características, que tiene como objetivo la identificación de puntos en una imagen digital en la que el brillo de la imagen cambia drásticamente o, más formalmente, tiene discontinuidades [4]. En el procesamiento de imágenes, la visión artificial y la visión por ordenador, la detección de bordes es una técnica esencial, especialmente en los campos de la identificación y la extracción de características. El objetivo de la detección de cambios bruscos en el brillo de la imagen es reconocer eventos y cambios significativos en las características del mundo. Se espera que las discontinuidades en el brillo de la imagen se correlacionen con discontinuidades en profundidad, discontinuidades en la orientación de la superficie, cambios en las características del material y fluctuaciones en la luz de la escena, dadas las hipótesis relativamente genéricas para un modelo de reproducción de imágenes [5].

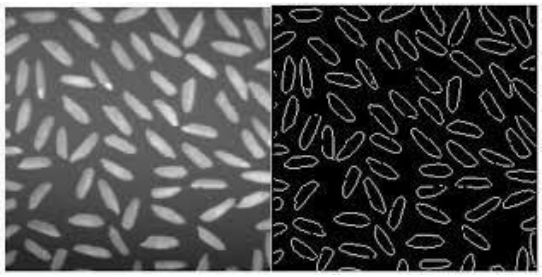


Fig 1. Resultado de un proceso de detección de borde [6].

Detección de bordes de Canny

Este es el método más utilizado, de gran triunfo y complicado en comparación con muchos otros. Es un método de varias etapas para detectar e identificar una variedad de bordes. La detección de bordes Canny utiliza un filtrado lineal con un núcleo gaussiano para suavizar el ruido y, a continuación, calcula la intensidad y la dirección de los bordes de cada píxel de la imagen suavizada. En este proceso, la intensidad de los bordes de cada píxel candidato (Los píxeles candidatos a bordes se identifican como los píxeles que perduran a un proceso de adelgazamiento llamado supresión no máxima) se pone a cero si su intensidad de borde no es mayor que la intensidad de borde de los dos píxeles adyacentes en la dirección del gradiente.

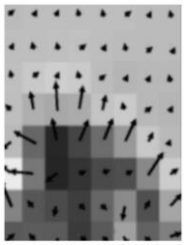


Fig 2. El gradiente define la dirección del cambio de intensidad [6].

El umbral se realiza en la imagen de magnitud de borde afinada utilizando la histéresis. En la histéresis, se utilizan dos umbrales de intensidad de borde. Todos los píxeles de borde candidatos por debajo del umbral inferior se etiquetan como no bordes y todos los píxeles por encima del umbral bajo que pueden conectarse a cualquier píxel por encima del umbral alto a través de una cadena de píxeles de borde se marcan como píxeles de borde.

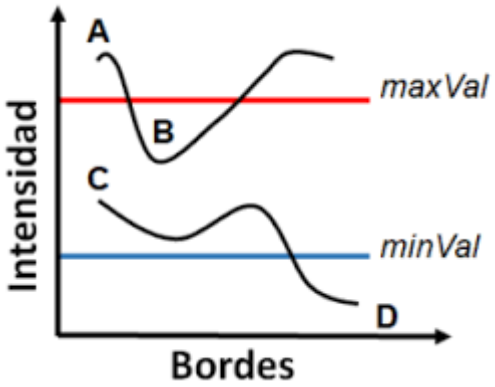


Fig 3. Límites Max y Min para determinar un borde [6].

Este método requiere que se introduzca tres parámetros. El primero es sigma, la desviación estándar del filtro gaussiano especificada en píxeles. El segundo parámetro es low, el umbral bajo (threshold) que se especifica como una fracción del umbral alto calculado. El tercer parámetro alto es el umbral alto (threshold) que se utilizará en la histéresis y se especifica como un punto porcentual en la distribución de los valores de la magnitud del gradiente para los píxeles candidatos al borde [5].

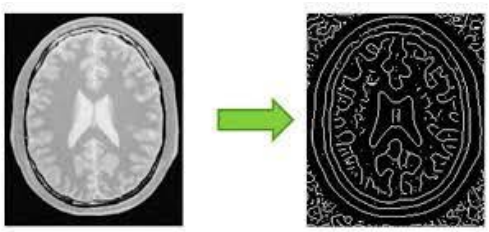


Fig 4. Resultado de un proceso de detección de borde [6].

Este método se puede usar en openCV mediante la función cv.Canny

cv2.Canny(image, minVal, maxVal)

Los parámetros son los siguientes:

1. image es la imagen gris que va a ser analizada.
2. minVal, maxVal son los valores de umbrales para que el algoritmo en la etapa Hysteresis Thresholding decida cuales son contornos.

Detección de contornos de una imagen

Existe diferencia entre un borde y un contorno. Los bordes, son cambios de intensidad pronunciados. Sin embargo, un contorno es una curva de puntos sin huecos ni saltos, es decir, tiene un principio y el final de la curva termina en ese principio [7].

Fig 5. contorno - no contorno [7].

El objetivo de esta fase es analizar todos los bordes detectados y comprobar si son contornos o no. En OpenCV lo podemos hacer con el siguiente método.

```
(contornos, jerarquia) = cv2.findContours(imagenbinarizada, modo_contorno, metodo_aproximacion)
```

Donde:

- Resultado: se obtiene 2 valores como resultados.
  - contornos: es una lista de Python con todos los contornos que ha encontrado. Luego veremos cómo dibujar estos contornos en una imagen.
  - jerarquía: la jerarquía de contornos.
- imagenbinarizada: es la imagen donde hemos detectado los bordes o umbralizado.
- modo contorno: es un parámetro interno del algoritmo que indica el tipo de contorno que se quiere. Puede tomar diferentes valores RETR_EXTERNAL (obtiene el contorno externo de un objeto), RETR_LIST, RETR_COMP y RETR_TREE.
- metodo_aproximacion: este parámetro indica se quiere aproximar el contorno. Puede tomar dos valores CHAIN_APPROX_NONE que toma todos los puntos y

CHAIN_APPROX_SIMPLE, que elimina todos los puntos redundantes.

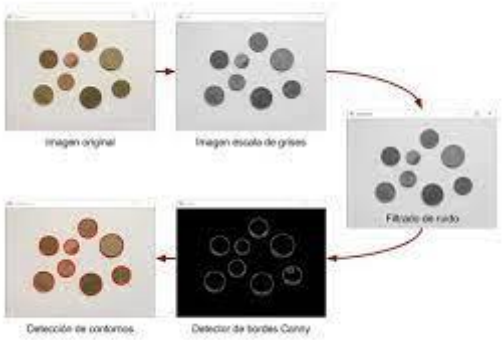
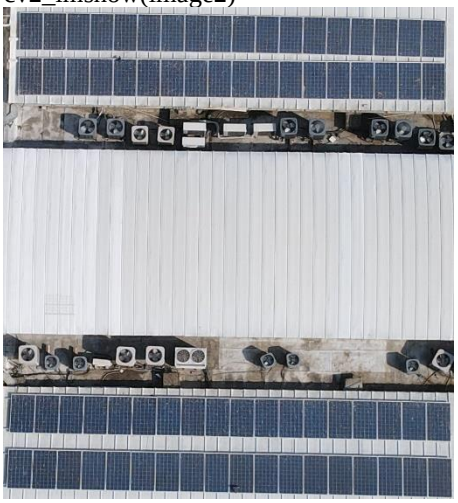


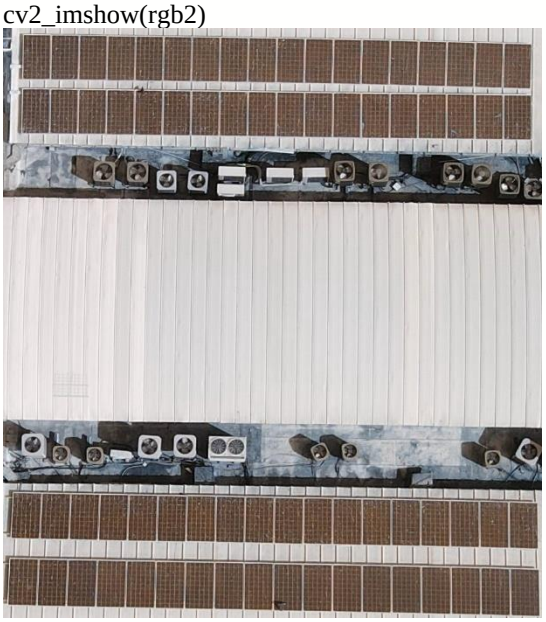
Fig 6. proceso de detección de bordes y contornos [7].

II. DESARROLLO

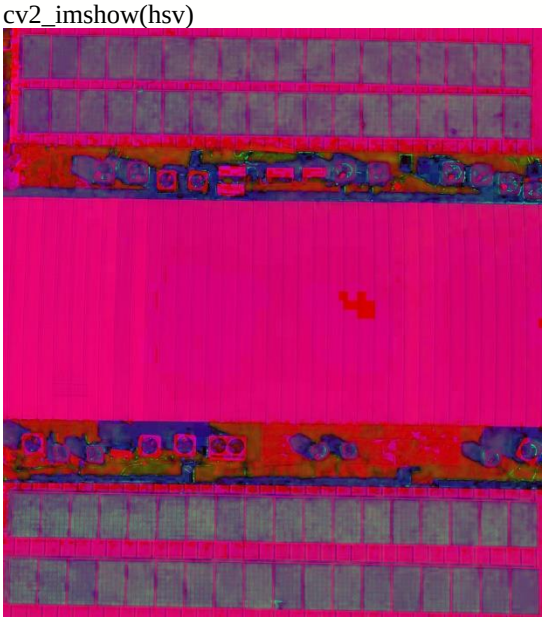
```
# PROCESAMIENTO DE CONTENIDO MULTIMEDIA
# Luis Ariza - Erick Barros
#Importando librerias
from PIL import Image
from matplotlib import image
from matplotlib import pyplot
import matplotlib.pyplot as plt
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
image2 = cv2.imread('paneles2.jpg')
cv2_imshow(image2)
```



```
rgb2=cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)
cv2_imshow(rgb2)
```

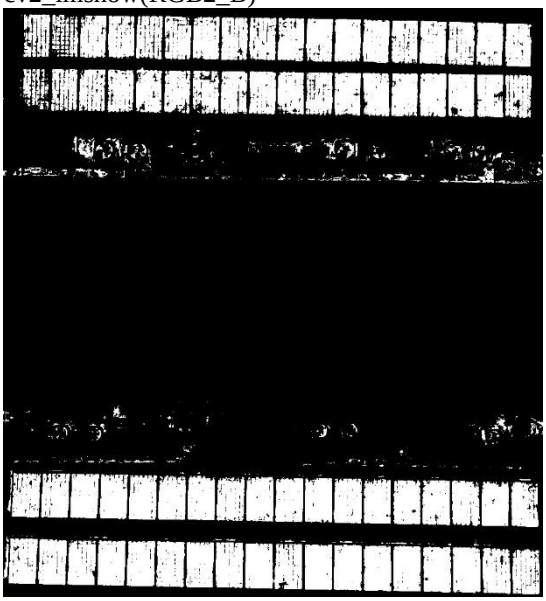


```
hsv=cv2.cvtColor(image2,
cv2.COLOR_BGR2HSV)
cv2_imshow(hsv)
```



```
#Azul
lower_B = np.array([105,60,20])
upper_B = np.array([140,255,160])
mask_B=cv2.inRange(hsv,lower_B,upper_B)
RGB_B=cv2.cvtColor(mask_B,cv2.COLOR_BGR2RGB)
cv2_imshow(RGB_B)
```

```
blue=cv2.bitwise_and(image2,image2,mask=mas
k_B)
RGB2_B=cv2.cvtColor(blue,cv2.COLOR_BGR2
RGB)
cv2_imshow(RGB2_B)
```



```
gray=cv2.cvtColor(RGB2_B,cv2.COLOR_RGB2
GRAY)
canny = cv2.Canny(gray, threshold1=0,
threshold2=358,  L2gradient = True )

cnts,_ = cv2.findContours(canny,
cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
cont = 0

for c in cnts:
  epsilon = 0.08*cv2.arcLength(c,True)
  approx = cv2.approxPolyDP(c,epsilon,True)
  if len(approx)==4:
    cont = cont + 1

print("Hay {} paneles".format(cont))
plt.axis('off')
plt.imshow(canny, cmap="gray")
plt.show()
```

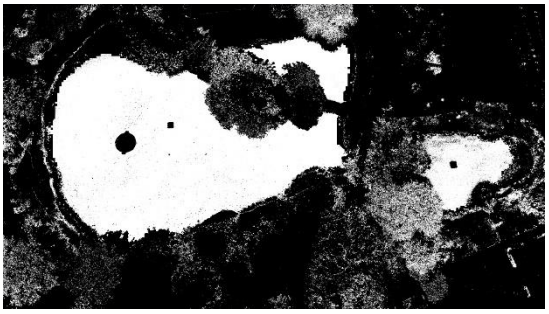
IMAGEN 2

```
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import image
from matplotlib import pyplot
from google.colab.patches import cv2_imshow

image2=cv2.imread('lago.JPG')
#Convertimos a HSV
RGB=cv2.cvtColor(image2,
cv2.COLOR_BGR2HSV)
cv2_imshow(image2)
```

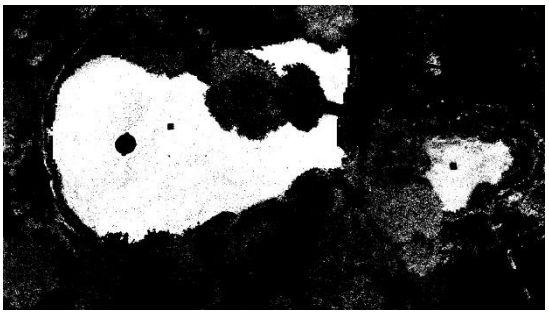
```
#ESCALA DE GRISES
gray= cv2.cvtColor(RGB,
cv2.COLOR_BGR2GRAY)
cv2_imshow(gray)
```

```
# SEGMENTACION
lowerr = np.array([32, 0, 80])
upperr = np.array([50, 105, 110])
mask = cv2.inRange(RGB, lowerr,upperr)
cv2_imshow(mask)
```



```
#Contornos
(contours,_ ) =
cv2.findContours(mask,cv2.RETR_CCOMP,cv2.
CHAIN_APPROX_SIMPLE)

#Dibujamos los contornos
cv2.drawContours(mask, contours, -1, (0,0,0), 1)
mask1=cv2.inRange(RGB,lowerr, upperr )
mask2=mask
cv2_imshow(mask)
```



```
#Segmento lago 1
contours, hierarchy =
cv2.findContours(mask2,cv2.RETR_EXTERNAL
,cv2.CHAIN_APPROX_SIMPLE)
c1 = max(contours,key=cv2.contourArea)
s1 = np.zeros_like(mask2)

cv2.drawContours(s1, [c1],
0,(255,255,255),cv2.FILLED)
Lago_p1=cv2.bitwise_not(s1)
l1=cv2.bitwise_and(mask2,mask2,mask=Lago_p1
)

#Segmento lago 2
contours, hierarchy =
cv2.findContours(l1,cv2.RETR_EXTERNAL,cv2
.CHAIN_APPROX_SIMPLE)
c2 = max(contours,key=cv2.contourArea)
s2 = np.zeros_like(mask2)

cv2.drawContours(s2, [c2],
0,(255,255,255),cv2.FILLED)
Lago_p2=cv2.bitwise_not(s2)
l2=cv2.bitwise_and(l1, l1 ,mask=Lago_p2)

#Segmento lago 3
contours, hierarchy =
cv2.findContours(l2,cv2.RETR_EXTERNAL,cv2
.CHAIN_APPROX_SIMPLE)
c3 = max(contours,key=cv2.contourArea)
s3= np.zeros_like(mask2)

cv2.drawContours(s3, [c3],
0,(255,255,255),cv2.FILLED)
Lago_p3=cv2.bitwise_not(s3)
l2=cv2.bitwise_and(l2,l2 ,mask=Lago_p3)

cv2_imshow(s1+s2+s3)
```

#sumamos los segmentos del lago para imprimirlos
LagoSeg=cv2.bitwise_and(image2, image2,mask=s1+s2+s3)
cv2_imshow(LagoSeg)

LagoSeg2= cv2.bitwise_and(LagoSeg,LagoSeg,mask=mask1)
cv2_imshow(LagoSeg2)

III. CONCLUSIONES

En conclusión, el filtrado de imagen RGB y la detección de bordes son técnicas fundamentales en el procesamiento de imágenes y visión por computadora.
El filtrado de imagen RGB implica la manipulación de los canales de color rojo, verde y azul de una imagen para mejorar su calidad, eliminar ruido y resaltar características de interés. Algunos de los métodos de filtrado de imagen más comunes incluyen el suavizado Gaussiano, la eliminación de ruido de mediana y el enfoque.

Por otro lado, la detección de bordes se refiere al proceso de identificar los límites entre las diferentes regiones de una imagen. Esto se puede lograr utilizando diferentes algoritmos, como el operador de Sobel, el operador de Canny y el operador de Laplace. La detección de bordes es útil para identificar objetos en una imagen, segmentar regiones y realizar otras operaciones de procesamiento de imágenes.

En conjunto, estas técnicas son ampliamente utilizadas en diversas aplicaciones, como el reconocimiento de objetos, la clasificación de imágenes, la identificación de características faciales y la automatización de tareas industriales. Además, la combinación de ambas técnicas puede mejorar aún más la precisión y la calidad del procesamiento de imágenes.

Finalmente, se pudo extraer solo los paneles de imagen original capturada con el dron mediante el filtrado y asi segmentar la imagen evidenciando el numero de paneles que hay, por medio de la busqueda de contornos y la sumatoria de aristas que este tiene para identificación de los rectangulos de los paneles. Además, se asimiló como encontrar contornos de diferentes figuras e identificarlas para así extraer contorno ideal del proceso como en el caso del lago.

IV. BIBLIOGRAFIA

[1] WikiPedia, «Espacio de color,» [En línea]. Available: https://es.wikipedia.org/wiki/Espacio_de_color.

[2] kipunaEc, «Cambio de espacios de color openCV - python,» [En línea]. Available: https://noemioocc.github.io/posts/Cambio-deespacio-de-color-openCV-python/.

[3] WikiPedia, «Segmentación (procesamiento de imágenes),» [En línea]. Available: https://es.wikipedia.org/wiki/Segmentación_(proc es amiento_de_imágenes).

[4] «Detector de bordes,» [En línea]. Available: https://es.wikipedia.org/wiki/Detector_de_bordes#:~:text=La%20detección%20de%20bordes%20es,la%20imagen%20cambia%20drásticamente%20o%2 C.

[5] Jesús, «Lo Que Necesitas Saber Sobre la Detección de Bordes,» [En línea]. Available: https://www.datasmarts.net/lo-que-necesitas-sabersobre-la-deteccion-de-bordes/.

[6] kipunaEc, «Detección de bordes Canny - openCV python,» [En línea]. Available: https://noemioocc.github.io/posts/Detección-debordes-Canny-openCV-python/.

[7] L. d. V. Hernández, «Detector de bordes Canny cómo contar objetos con OpenCV y Python,» [En línea]. Available: https://programarfacil.com/blog/visionartificial/detector-de-bordes-canny-opencv/.