

# **TRABAJO PRÁCTICO FINAL - PROCESAMIENTO DE IMÁGENES Y VISIÓN POR COMPUTADORA**

Santiago Medina

Esteban Suarez

1er Nivel

Noviembre de 2022



## ÍNDICE

1.	Introducción.....	2
2.	Marco teórico.....	3
2.1.	Espacios de color.....	3
2.1.1.	Colores.....	3
2.1.2.	Matiz/Valor.....	4
2.1.3.	Saturación.....	4
2.1.4.	Luminancia.....	4
2.2.	Espacios de colores muy utilizados.....	4
2.2.1.	RGB.....	4
2.2.2.	HSV.....	5
2.2.3.	HSL.....	5
2.2.4.	YIQ.....	5
2.3.	Conversiones entre espacios de colores.....	6
2.4.	Análisis de imágenes.....	7
2.5.	Selección por color.....	8
3.	Herramientas utilizadas en el trabajo.....	8
3.1.	Lenguaje de programación.....	8
3.2.	Librerías.....	9
3.3.	Funciones.....	9
4.	Implementación.....	10
4.1.	Resultados en algunas imágenes.....	14
5.	Conclusiones.....	16
6.	Bibliografía.....	17

## INTRODUCCIÓN

Muchas veces hemos ido por la calle y seguramente a más de uno le habrá pasado de ver gente que no respeta las señales de tránsito, o a uno mismo también puede haberle ocurrido el hecho de manejar estando medio dormido o con mucho estrés encima producto de las obligaciones rutinarias. Esto trae consecuencias negativas para el normal orden de la sociedad, puesto que lo mencionado es causa principal de los accidentes diarios que vemos por las noticias.

Por otro lado, si bien muchas de las funciones que antes cumplía el conductor o alguno de los pasajeros dentro del auto ahora las cumple una máquina, hay una que todavía está a cargo de las personas: el procesamiento de las señales de tránsito. Pensemos en que esta es una tarea cuyo desempeño sería mejor si estuviera a cargo de un software diseñado para tal fin. Por ejemplo, las tareas que pueden ser ejecutadas mediante el procesamiento de imágenes son: detección de semáforos, reconocimiento facial, reconocimiento de objetos y/o lugares, detección de formas geométricas, etc.

Por otro lado, los objetivos propuestos para este TP son los siguientes:

- Lograr un buen manejo de los espacios de color vistos durante el cursado.
- Comprender la utilidad de la binarización de imágenes.
- Aplicar otros conceptos aprendidos anteriormente sobre el lenguaje Python.



## MARCO TEÓRICO

### ESPACIOS DE COLOR

El espacio de color es la gama de colores que puede representar el ordenador. Es una lista estándar de colores codificados. Cuando la cámara captura una imagen, en la tarjeta, lo que está grabando son datos que más tarde el programa del ordenador tiene que “traducir” o descodificar y lo hace dentro de la gama de colores de la que dispone, que es mucho más limitada que la gama de colores que existen en la realidad.

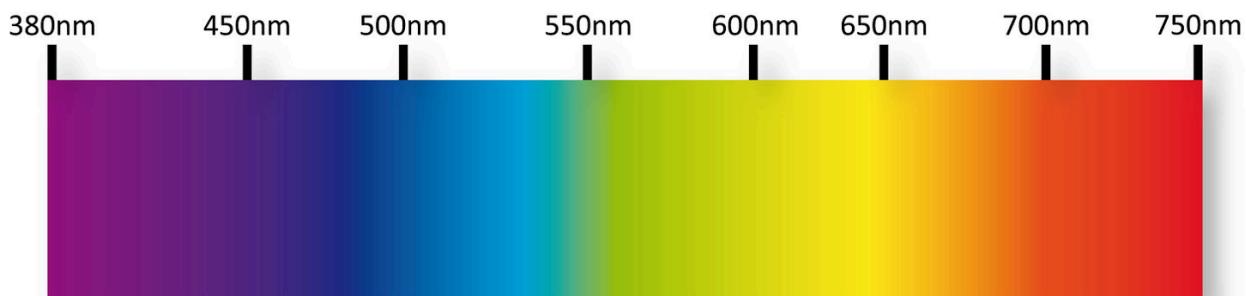


Figura 1. Colores del espectro, visibles para el ser humano

Para verlo más claro, nuestros ojos nos muestran los colores como una mezcla de rojo, naranja, amarillo, verde, azul y violeta, y vemos una combinación de estos siete. Mientras que las pantallas (cámara, tv, monitores, etc.), lo representan como una mezcla solo de verde, rojo y azul. Por otro lado, las impresoras lo hacen como una combinación de cian, magenta, amarillo y negro.

### Colores

El color es un atributo que percibimos de los objetos cuando hay luz. La luz está constituida por ondas electromagnéticas que se propagan a unos 300.000 km por segundo. Esa luz no viaja en línea recta sino en forma de ondas. Cada onda tiene una longitud de onda diferente que es lo que produce los distintos tipos de luz, como la luz infrarroja, la luz ultravioleta o el espectro visible. Los objetos devuelven los rayos que no absorben hacia su entorno. Nuestro cerebro interpreta estas radiaciones electromagnéticas que los objetos reflejan como lo que llamamos **color**.

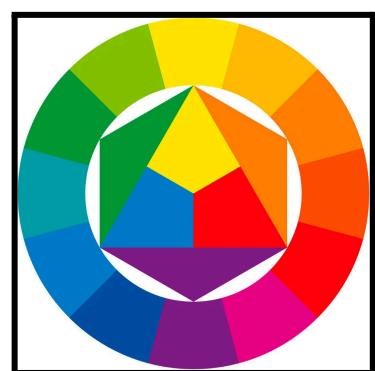


Figura 2. Colores por categoría



## Matiz/Valor

También llamado por algunos croma, es el color en sí mismo, es el atributo que nos permite diferenciar a un color de otro, por lo que podemos designar cuando un matiz es verde, violeta o anaranjado.



Figura 3. Representación gráfica de la matiz.

## Saturación

La saturación hace que los colores sean más vivos (menos color negro o blanco añadido). La desaturación hace que los colores sean más apagados (más color negro o blanco añadido).



Figura 4. Representación gráfica de la saturación.

## Luminancia

La luminosidad, también llamada claridad, es una propiedad de los colores. Ella da una indicación sobre el aspecto luminoso del color estudiado: cuanto más oscuro es el color, la luminosidad es más débil. Este término se asocia a veces con el concepto de valor, luminancia, luz.



Figura 5. Representación gráfica de la luminancia.

## Algunos espacios de colores muy utilizados

### (RGB) Red/Green/Blue

Por sus siglas en inglés, Rojo/Verde/Azul, está definido por las tres cromaticidades de los primarios aditivos, los ya nombrados, y puede producir cualquier cromaticidad que sea el triángulo definido por esos colores primarios. La especificación completa de un espacio de color RGB también requiere una cromaticidad de punto blanco y una curva de corrección gamma. En este espacio, de especial relevancia en el mundo de la electrónica, cada color queda representado con la intensidad de cada

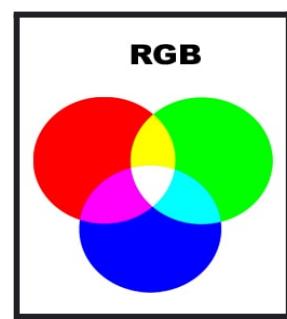


Figura 6. Espacio RGB



uno de los 3 canales en una escala del 0 al 255: el (0,0,0) sería el negro, (255, 0, 0) rojo, (0, 255, 0) verde, (0, 0, 255) azul... y el (255, 255, 255) el blanco.

### (HSV) Hue/Saturation/Value

Representación tridimensional del color basado en los componentes de matiz, saturación y valor o brillo.

- **Matiz(H):** Es lo que común -y erróneamente- llamamos color: la longitud de onda dominante.
- **Saturación(S):** Mide la pureza de un color, es decir la ausencia de otros colores: un color con el 100% de saturación se conseguirá tan solo con su matiz monocromático o espectral.
- **Valor(V):** Hace referencia a la intensidad de la luz, es decir, la cantidad de luz que hay en ese color, desde un 0% (negro) al 100%.

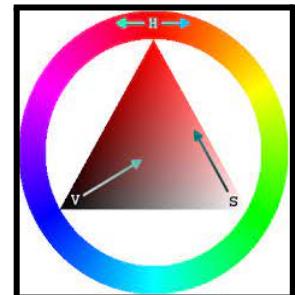


Figura 7. Espacio HSV

Se lo puede representar también en una escala del 0 al 255, al igual que al RGB.

### (HSL)Hue/Saturation/Lightness

El modelo HSL describe los colores con los parámetros **Matiz, Saturación y Luminosidad**, definiendo esto último como el promedio entre el mínimo y el máximo valor de sus coordenadas RGB. La principal diferencia con el espacio **HSV** es que si partimos de un matiz puro y bajamos la saturación al mínimo en el modelo HSV obtendremos el color blanco y en el HSL el gris.

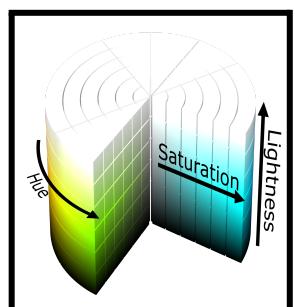


Figura 8. Espacio HSL

### (YIQ)

Espacio de color usado por el estándar de televisión NTSC. Y representa la información de luminancia e I y Q representan la información de crominancia. I significa "in-phase" (en fase), mientras que Q significa "quadrature" (cuadratura) y se refieren a los componentes usados en la modulación de amplitud en cuadratura. El sistema YIQ está destinado a aprovechar las características humanas de respuesta de color. El ojo es más sensible a los cambios en el rango naranja-azul (I) que en el rango púrpura-verde (Q); por lo tanto, se requiere menos ancho de banda para Q que para I.

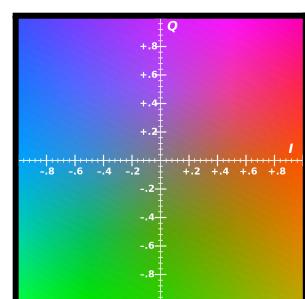


Figura 9. Espacio YIQ



## Conversiones entre espacios de colores

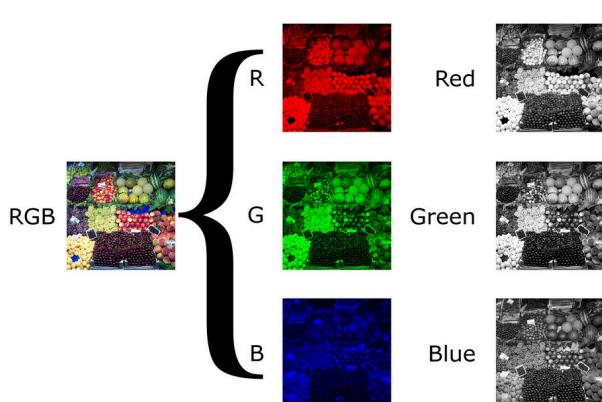


Figura 10. Ejemplo de conversión de RGB a Escala de grises.

### De RGB a escala de Grises

En fotografía, computación y colorimetría, una escala de grises o una escala de grises es aquella en la que el valor de cada píxel es una sola muestra que representa solo una cantidad de luz, es decir, solo transporta información de intensidad. Las imágenes de este tipo, también conocidas como blanco y negro o monocromáticas, están compuestas exclusivamente por tonos de gris, que varían desde el negro en la intensidad más débil hasta el blanco en el más fuerte. Las imágenes en escala de grises son distintas de las imágenes en blanco y negro de un bit, que en el contexto de imágenes por computadora son imágenes con solo dos colores, blanco y negro (también llamadas imágenes binarias o binarias). Las imágenes en escala de grises tienen muchos tonos de gris en el medio.

desde el negro en la intensidad más débil hasta el blanco en el más fuerte. Las imágenes en escala de grises son distintas de las imágenes en blanco y negro de un bit, que en el contexto de imágenes por computadora son imágenes con solo dos colores, blanco y negro (también llamadas imágenes binarias o binarias). Las imágenes en escala de grises tienen muchos tonos de gris en el medio.

Para convertir un color de un espacio de color basado en un modelo de color RGB típico gamma comprimido (no lineal) a una representación en escala de grises de su luminancia, primero se debe eliminar la función de compresión gamma mediante expansión gamma (linealización) para transformar la imagen en un RGB lineal espacio de color, para que la suma ponderada apropiada se pueda aplicar a los componentes de color lineales *R linear*, *G linear*, *B linear*, para calcular la luminancia lineal *Y linear* que luego se puede volver a comprimir con gamma si el resultado de la escala de grises también debe codificarse y almacenarse en un espacio de color no lineal típico.

Entonces, la luminancia lineal se calcula como una suma ponderada de los tres valores de intensidad lineal. El espacio de color RGB se define en términos de la *luminancia lineal* CIE 1931 *Y linear*, que viene dada por:

$$C_{\text{linear}} = \begin{cases} \frac{C_{\text{srgb}}}{12.92}, & C_{\text{srgb}} \leq 0.04045 \\ \left( \frac{C_{\text{srgb}} + 0.055}{1.055} \right)^{2.4}, & C_{\text{srgb}} > 0.04045 \end{cases}$$

Figura 11. C-linear representa al valor de intensidad lineal correspondiente a (R,G,B lineales)

$$Y_{\text{linear}} = 0.2126R_{\text{linear}} + 0.7152G_{\text{linear}} + 0.0722B_{\text{linear}}$$

Figura 12. Definición



Estos tres coeficientes particulares representan la percepción de la intensidad (luminancia) de humanos tricrómicos típicos a la luz de la Rec. 709 colores primarios aditivos (cromaticidades) que se usan en la definición de RGB. La visión humana es más sensible al verde, por lo que tiene el mayor valor de coeficiente (0.7152) y es menos sensible al azul, por lo que tiene el menor coeficiente (0.0722).

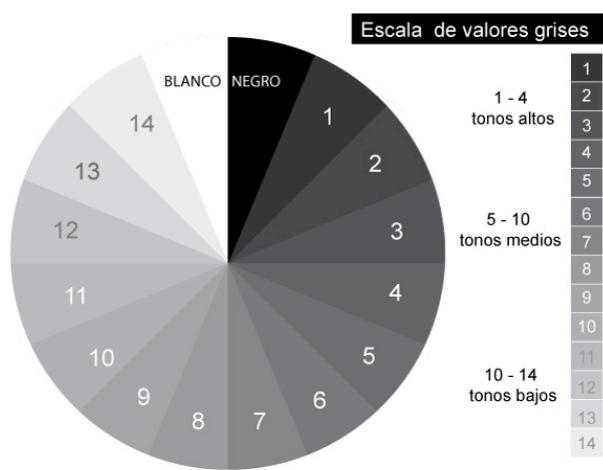


Figura 13. Representación de los valores en escala de grises similar, dependiendo las cualidades de cada uno de ellos.

Para codificar la intensidad de escala de grises en RGB lineal, cada una de las tres componentes de color se puede configurar para igualar la luminancia lineal calculada  $Y_{lineal}$  (reemplazando  $R_{lineal}$ ,  $G_{lineal}$ ,  $B_{lineal}$  por los valores  $Y_{lineal}$ ,  $Y_{lineal}$ ,  $Y_{lineal}$  para obtener la escala de grises) que luego típicamente necesita ser comprimida con gamma para volver a una representación convencional no lineal.

De manera análoga la conversión de los diferentes espacios se transforman de manera

## Análisis de imágenes

- Las imágenes, a nivel de programación se representan como matrices de dos dimensiones MxN, siendo M la cantidad de filas y N la cantidad de columnas de la matriz.
- Cada uno de los elementos de la matriz se denominan píxeles y el valor numérico que tienen representa la intensidad del mismo.
- Tipos de imágenes más utilizados: escala de grises y RGB.

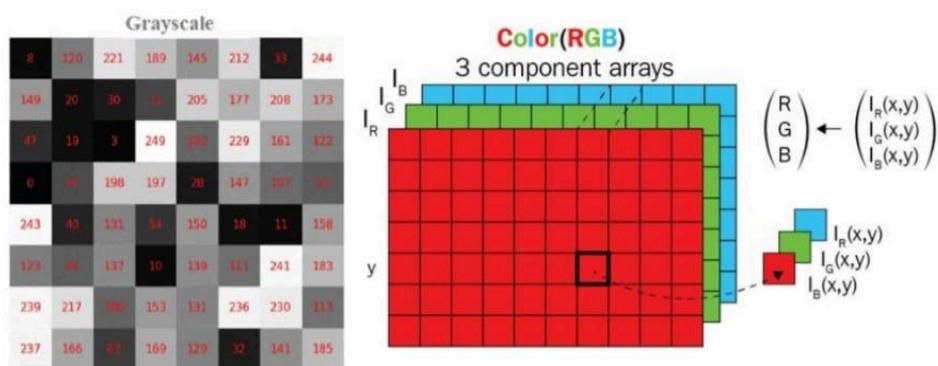


Figura 14. Representaciones matriciales de una imagen



### Selección por color

En algunos casos, se desea extraer información de la imagen según su color.

Puede definirse un umbral determinado para cada canal de la imagen, luego se recorren todos los elementos de la matriz, comprobando si están por encima del umbral. Si no cumplen este requerimiento se convierte el píxel en negro, pudiéndose extraer objetos del color deseado para posteriores análisis.

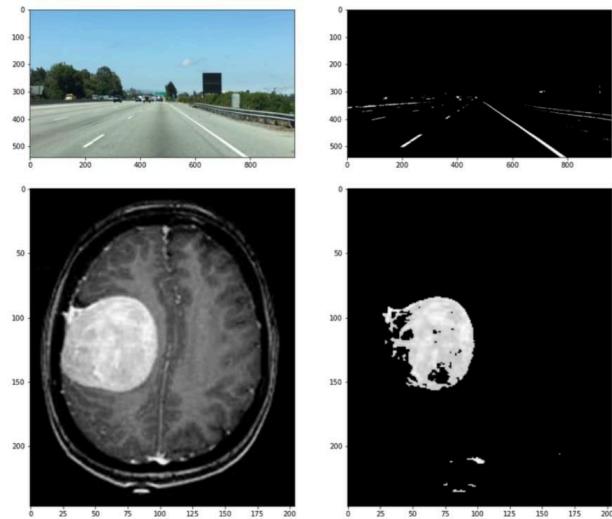


Figura 15. Ejemplo de creación de máscara

## HERRAMIENTAS

Antes de comenzar con la explicación y con los procedimientos, para realizar el procesamiento de las imágenes, se mencionarán algunos conceptos importantes de las herramientas que se utilizaron.

### Lenguaje de Programación

Para llevar a cabo el trabajo práctico del procesamiento de imágenes y visión por computadora se utilizó **Python**, lenguaje ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML). Los desarrolladores utilizan Python porque es eficiente y fácil de aprender, además de que se puede ejecutar en muchas plataformas diferentes. El software Python se puede descargar gratis, se integra bien a todos los tipos de sistemas y aumenta la velocidad del desarrollo.





## Librerías

- **IMAGEIO.V2:** Imageio es una biblioteca Python. Ofrece un uso sencillo para escribir y leer una amplia variedad de datos de imágenes.



- **CV2:** (OpenCV): Uno de sus usos más importantes se basa en la visión por computadora es la detección de rostros y objetos, sobre todo en ámbitos como la fotografía, el marketing o la seguridad.



- **MATPLOTLIB:** Es una librería para la generación de gráficos en dos dimensiones, a partir de datos contenidos en listas o arrays en el lenguaje de programación Python.



- **NUMPY:** Da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas.



## Funciones

- imshow(): en el módulo pyplot de la librería matplotlib se usa para mostrar datos como una imagen; es decir, en un rástreo regular 2D.
- show(): en el módulo pyplot de la librería matplotlib se usa para mostrar todo tipo de figuras.
- cvtColor(): es una función de openCV que convierte imágenes de un espacio de color a otro.
- array(): es una función de numpy que crea un arreglo a partir de una lista o tupla y devuelve una referencia a él.
- inRange(): es una función que permite filtrar una ventana de valores para píxeles en una imagen.
- add(): es una función de openCV que permite sumar el contenido de dos imágenes.
- findContours(): en openCV es una función que permite encontrar los n contornos externos e internos de una imagen binaria.
- contourArea(): es una función de openCV que sirve para calcular el área encerrada por los contornos encontrados en una imagen binaria.
- drawContours(): es una función de openCV que se utiliza para dibujar los contornos encontrados en una imagen binaria.
- moments(): es una función de openCV que devuelve los momentos de una imagen.



- `circle()`: es una función de openCV que sirve para dibujar un círculo en una imagen.
- `putText()`: es una función de openCV que como el nombre lo dice, sirve para colocar texto en una imagen.

## IMPLEMENTACIÓN

A continuación, se mostrará lo realizado durante el Trabajo Práctico. Para ello, no mostraremos el procesamiento de todas las imágenes (en total 62) por cuestiones de brevedad del documento, pero sí presentaremos los procedimientos realizados y los resultados obtenidos en 10 imágenes del total.

Por otro lado, la estrategia utilizada para la detección de los semáforos fue la siguiente:

- 1) Definir los valores H,S y V de interés para detectar los colores necesarios.
- 2) Crear una máscara donde sólo se señalan las regiones donde se detectó el color buscado.
- 3) Ubicar cartesianamente esas regiones y quedarnos únicamente con aquellas que superen un valor dado de área.
- 4) Encerrar con un círculo aquellas regiones que superaron el valor de área e indicar el color detectado.

La imagen a procesar fue la siguiente:



Figura 16. Imagen 32

Lo primero que haremos será transformar la imagen de BGR a HSV. Luego definiremos los rangos de interés para hacer un estudio de los semáforos rojos, y con dichos rangos crearemos una máscara que tendrá el mismo tamaño de la imagen y en donde se pintará de amarillo aquellas



regiones donde se detectó el color buscado, y de morado aquellas otras donde no se haya detectado.

```
import cv2
import imageio.v2 as io
import matplotlib.pyplot as plt
import numpy as np
ruta = r'E:\Downloads\32.jpg'
imagen_cv = io.imread(ruta)

#Mostramos por pantalla la imagen a procesar
plt.imshow(imagen_cv)
plt.show()

imagen_cv_hsv = cv2.cvtColor(imagen_cv, cv2.COLOR_BGR2HSV)

#Definimos los rangos para para el color rojo de los valores H,S,V a ser estudiados
redBajo1 = np.array([0,100,20], np.uint8)
redAlto1 = np.array([1,101,21], np.uint8)

redBajo2 = np.array([120,120,70], np.uint8)
redAlto2 = np.array([122,255,255], np.uint8)

#Creamos la máscara con los rangos definidos anteriormente
maskRed1 = cv2.inRange(imagen_cv_hsv, redBajo1, redAlto1)
maskRed2 = cv2.inRange(imagen_cv_hsv, redBajo2, redAlto2)
maskRed = cv2.add(maskRed1, maskRed2)

#Mostramos por pantalla la máscara creada para el rojo
plt.imshow(maskRed)
plt.show()
```

Figura 17. Código ejecutado para la creación de la máscara

Mostramos por pantalla la máscara creada

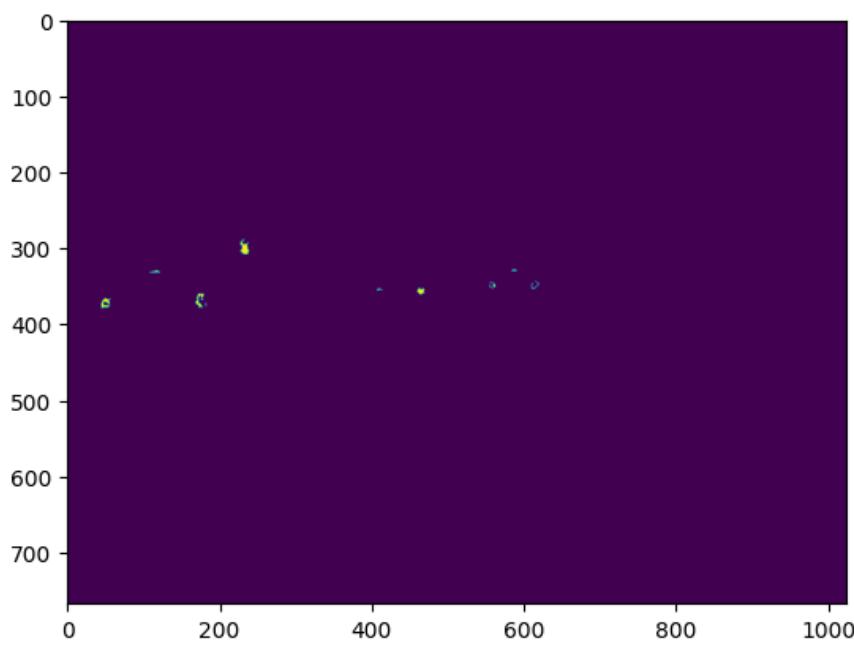


Figura 18. Máscara creada



Como podemos ver, se detectaron pequeñas regiones donde en realidad no hay semáforos, para ello nos quedaremos únicamente con aquellas “manchas” de la máscara cuya área sea mayor a 96 y las contornearemos con la función cv2.contourArea().

Por otro lado, mediante la función cv2.moments() encontraremos la posición cartesiana de cada región y una vez hecho eso, dibujaremos el círculo y escribiremos el nombre del color correspondiente.

```
#Señalamos en la imagen original los lugares donde fueron detectados los semáforos en rojo
contornos, hierarchy = cv2.findContours(maskRed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
for c in contornos:
    area = cv2.contourArea(c)
    #Nos quedamos con aquellas manchas de la máscara cuya área sea mayor a 96
    if area > 96:
        cv2.drawContours(imagen_cv, [c], -1, (255,0,0), 1)
        M = cv2.moments(c)
        #Encontramos la posición cartesiana de la mancha
        cx = int(M["m10"] / M["m00"])
        cy = int(M["m01"] / M["m00"])
        #Dibujamos el círculo e indicamos el color detectado
        cv2.circle(imagen_cv, (cx,cy), 20, (0,0,255), 2)
        cv2.putText(imagen_cv, "Rojo", (cx+30,cy-10), 1, 2, (0,0,255), 2)
```

Figura 19. Código ejecutado para señalar el semáforo rojo

De esta manera, así quedaría el resultado parcial

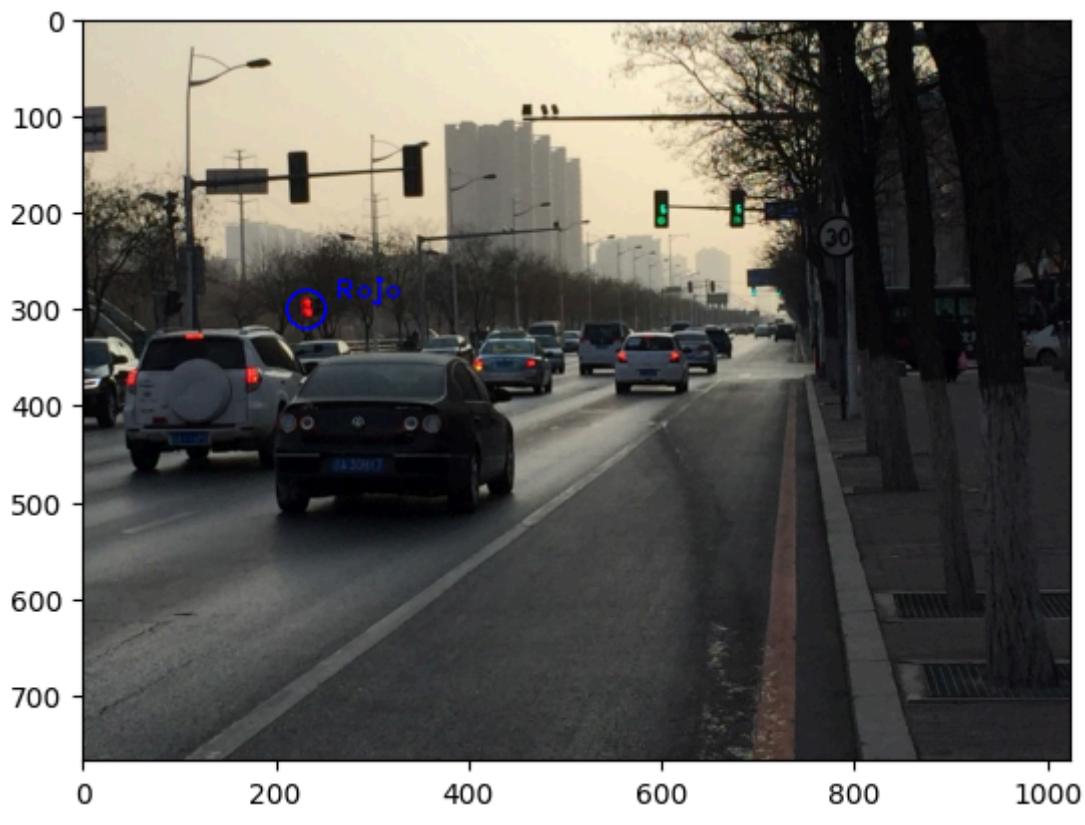


Figura 20. Resultado de la detección del semáforo rojo



Análogamente se procesa la imagen para detectar el/los semáforo/s verde/s:

```
#Definimos los rangos para para el color verde de los valores H,S,V a ser estudiados
greenBajo = np.array([35,100,20], np.uint8)
greenAlto = np.array([70,255,255], np.uint8)

#Creamos la máscara del verde
greenMask = cv2.inRange(imagen_cv_hsv, greenBajo, greenAlto)

#Mostramos por pantalla la máscara creada para el verde
plt.imshow(greenMask)
plt.show()
```

Figura 21. Código ejecutado para crear la máscara del verde

Mostramos por pantalla la máscara creada

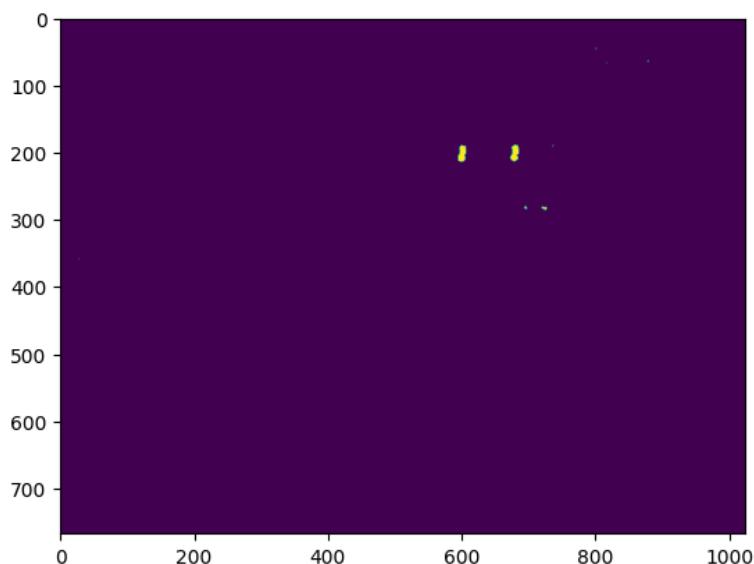


Figura 22. Máscara creada para la detección de semáforos verdes

```
#Señalamos en la imagen original las regiones donde se detectó semáforo en verde
bordes, hierarchy = cv2.findContours(greenMask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
for c in bordes:
    area = cv2.contourArea(c)
    if area > 100:
        cv2.drawContours(imagen_cv, [c], -1, (0,255,0), 1)
        M = cv2.moments(c)
        cx = int(M["m10"])/M["m00"]
        cy = int(M["m01"])/M["m00"])
        cv2.circle(imagen_cv, (cx,cy), 20, (0,0,255), 2)
        cv2.putText(imagen_cv, "Verde", (cx+30,cy-10), 1, 2, (0,0,255), 2)

#Mostramos por pantalla el resultado final
plt.imshow(imagen_cv)
plt.show()
```

Figura 23. Código ejecutado para señalar el semáforo verde



Figura 24. Resultado final del procesamiento

## RESULTADOS DE ALGUNAS IMÁGENES



Figura 25. Imagen 4



Figura 26. Imagen 5

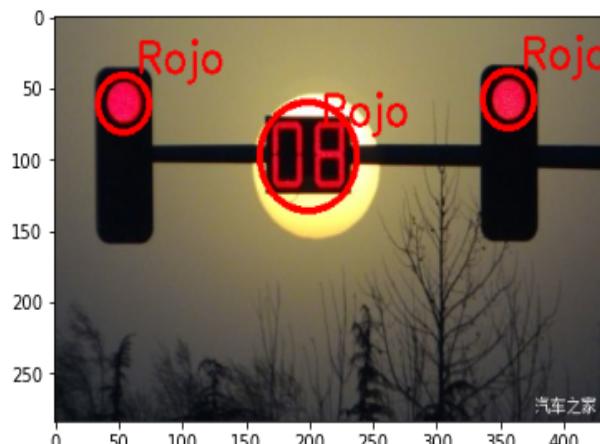


Figura 27. Imagen 11

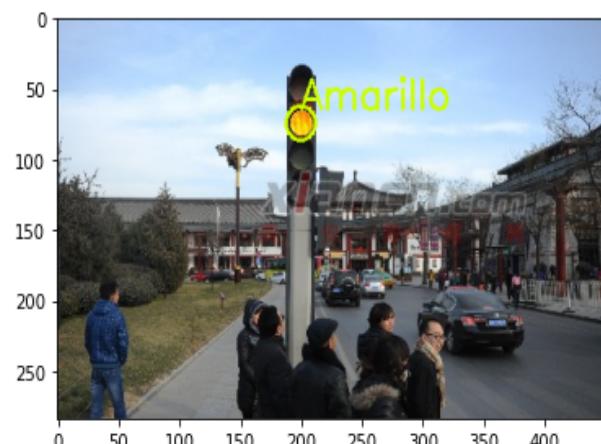


Figura 28. Imagen 21



Figura 29. Imagen 29



Figura 30. Imagen 30



Figura 31. Imagen 31



Figura 32. Imagen 36



Figura 33. Imagen 42



Figura 34. Imagen 59

Como se puede ver en la figura 33 (imagen 42), hay un semáforo rojo que no fue detectado. Eso se debe a que si uno observa los colores con más detenimiento, hay semáforos que sí se ven completamente rojos, mientras que el que no fue marcado tiene un tono más anaranjado, lo cual trajo complicaciones a la hora de crear la máscara de tonos rojos.

Métricas finales: se detectaron correctamente 61 imágenes frente al total (62).

Si se desea consultar el código fuente del presente proyecto, puede dirigirse a:

## CONCLUSIONES

Finalmente, podemos concluir en que se pudieron lograr todos los objetivos propuestos al principio de este informe haciendo uso de los espacios de color y algunas funciones de las librerías mencionadas.

Por otro lado, si bien surgieron dificultades con respecto a los valores a utilizar en el parámetro Hue del espacio de color HSV, se pudo solucionar ese problema y se pudo procesar las imágenes como corresponde.

Cabe mencionar que este sistema de detección tiene algunas limitaciones, como las siguientes:

- Modificación manual de los parámetros H, S y V para la creación de la máscara de filtro.
- Necesidad de cambiar rotundamente los parámetros H, S y V cuando cambiamos de una imagen que está de día a otra que está de noche.

Además, existen posibles mejoras para este diseño, como las siguientes:



- Utilización de otro espacio de color para la imagen que no fue detectada correctamente (imagen 42).
- Utilización de la convolución para el engrosamiento de bordes finos (semáforos que se ubican a mucha distancia de quien sacó la foto).

## BIBLIOGRAFÍA

[¿Qué Es y Para Qué Sirve el Espacio de Color? | Blog del Fotógrafo](#)

[Todos los matices o colores - Colegio Concepción San Pedro](#)

[Saturación y desaturación](#)

[Luminancia](#)

[Colorimetría III: espacio de color RGB, HSV y HSL • Lledó Energía](#)

[Espacio de color YIQ | HiSoUR Arte Cultura Historia](#)

[Qué es el color?](#)

[Escala de grises | HiSoUR Arte Cultura Historia](#)

[¿Qué es Python? | Guía de Python para principiantes de la nube | AWS](#)

[OpenCV, Instalación en Python y ejemplos básicos](#)

[matplotlib.pyplot.imshow\(\) en Python – Acervo Lima.](#)

[Imageio: la biblioteca Python dedicada a los datos de imágenes.](#)

[Matplotlib.pyplot.show\(\) in Python - GeeksforGeeks.](#)

<https://unipython.com/caracteristicas-los-contornos/>

[matplotlib.pyplot.imshow\(\) en Python – Acervo Lima](#)

[Imageio: la biblioteca Python dedicada a los datos de imágenes.](#)

[Matplotlib.pyplot.show\(\) in Python - GeeksforGeeks.](#)

[Características de los contornos - ▷ Cursos de Programación de 0 a Experto © Garantizados](#)

Filminas y notebooks de clase sobre Procesamiento de Imágenes:

[https://drive.google.com/drive/folders/18xD4k3ZxUkqW5ID\\_zfntGWwD8jsOv396?usp=sharing](https://drive.google.com/drive/folders/18xD4k3ZxUkqW5ID_zfntGWwD8jsOv396?usp=sharing)