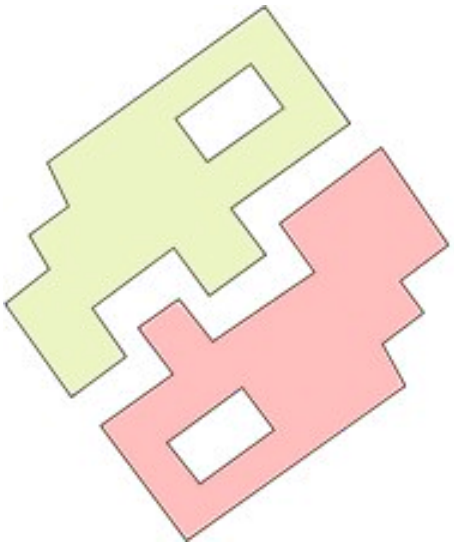


Fundación Tarpuy
Programa Ingenia

Trabajo Práctico Final – Procesamiento de Imágenes y Visión por Computadora

PRIMER NIVEL

AÑO 2023

Tutor: Nicolás Salomón

Integrantes: Gomez Roldán Malena, Gallone Francesco

Índice

1. INTRODUCCIÓN

El procesamiento de imágenes engloba un conjunto diverso de técnicas diseñadas para manipular imágenes digitales con el fin de mejorar su calidad, extraer información significativa o facilitar su posterior análisis. Este campo abarca desde operaciones básicas, como ajustes de brillo y contraste, hasta algoritmos más avanzados que permiten la identificación y segmentación de objetos en una imagen.

La motivación detrás del procesamiento de imágenes radica en la necesidad de extraer, analizar y comprender información valiosa contenida en imágenes.

Algunos ejemplos donde el procesamiento de imágenes está presente son en la medicina (ayuda en el diagnóstico de enfermedades); en aplicaciones industriales (se utiliza para inspeccionar y controlar la calidad de productos); en la biometría e identificación; en la agricultura; en la ganadería; en satélites de observación terrestre; como en tantas otras tareas de distinta índole en la sociedad.

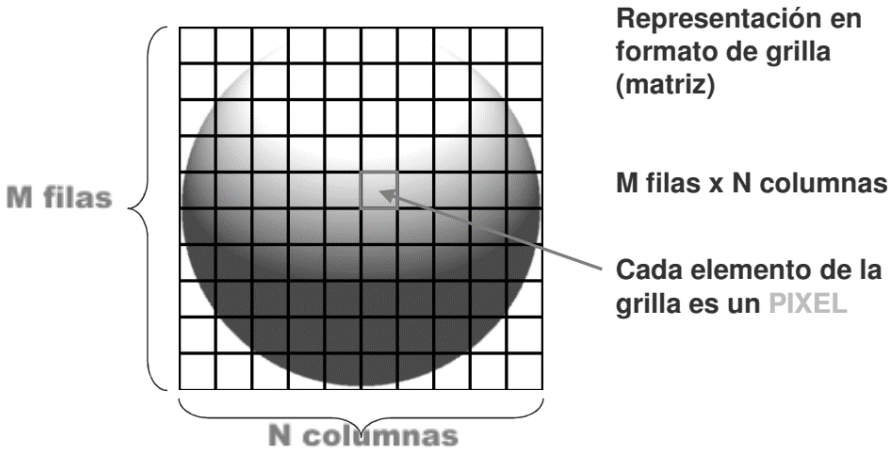
1.1 Objetivo

El objetivo principal de este trabajo es explorar y comprender el procesamiento de imágenes mediante el análisis detallado de un código específico. Se abordan conceptos clave, como la segmentación de colores, la detección de círculos y el reconocimiento de patrones en el contexto de un semáforo. A través de este estudio, se busca proporcionar una visión práctica del procesamiento de imágenes y cómo se puede aplicar en situaciones del mundo real.

2. MARCO TEÓRICO

2.1 Imágenes

Una imagen puede considerarse como el conjunto de puntos de colores, es decir, una sucesión coherente de puntos que conforman una matriz donde las filas y columnas corresponden a un punto en la imagen. Estos puntos se denominan 'píxeles'. Un píxel es la menor unidad homogénea en color que forma parte de una imagen digital, y el valor numérico que tienen representan la intensidad del mismo.



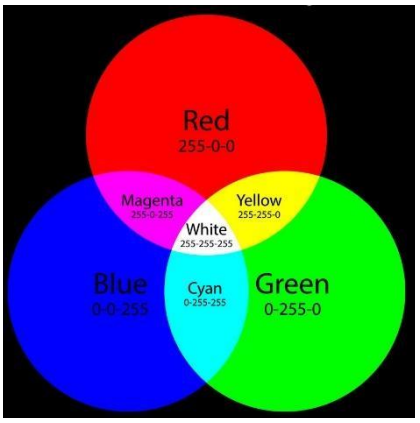
2.2 Espacios de color

Un espacio de color es un sistema de interpretación del color, es decir, una organización específica de los colores.

RGB

Los colores RGB se compone de los tres colores primarios a los que son más sensibles los conos fotorreceptores del ojo humano: **rojo, verde** y **azul**. Dado que los colores RGB se componen de los tres colores primarios, esto da lugar al término síntesis aditiva de color. Tres colores luminosos se combinan, es decir, se mezclan por adición para crear el color que se percibe.

Los tres colores primarios, rojo, verde y azul (también llamados canales de color), pueden adquirir intensidades de 0 (negro) a 255 (blanco), por lo que hay un total de 256 gradaciones por canal. Multiplicando todas las gradaciones de color disponibles por canal se obtienen16,7 millones de colores.

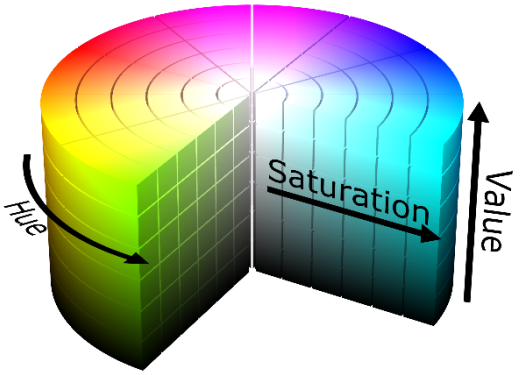


HSV

En el modelo de color HSV, un color se define por su tono o matiz (Hue), su saturación (Saturation) y su luminosidad (Value).

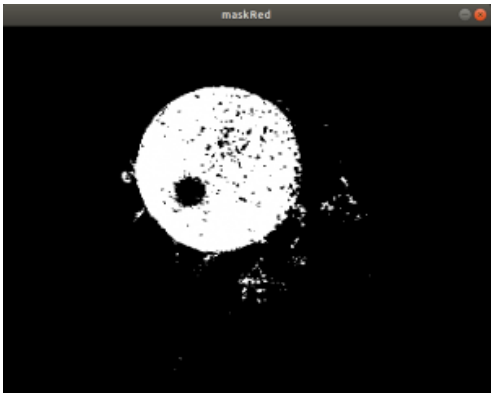
El modelo de color HSV es una transformación no lineal del modelo RGB en coordenadas cilíndricas de manera que cada color viene definido por las siguientes dimensiones:

- Tono o matiz: Ángulo que representa el matiz, normalmente definido entre 0° y 360°.
- Saturación: Nivel saturación del color, dado entre 0 y 1, 0 representa sin saturación alguna (blanco), hasta 1 que sería el matiz en toda su intensidad. Es común también darlo en percentiles 0%-100%.
- Brillo: Nivel del brillo entre 0 y 1. 0 es negro; 1, blanco. Al igual que la saturación puede darse en porcientos entre 0% y 100%. De esta forma el 50% indica el nivel medio o normal del brillo del color.



2.3 Máscaras

Las máscaras son un elemento muy importante dentro del procesamiento de imágenes. Junto a las selecciones (empleadas habitualmente por los programas de dibujo o diseño gráfico), las máscaras son la mejor herramienta para actuar selectivamente sobre ciertas regiones, elementos o propiedades de una imagen, protegiendo o dejando inalterado el resto de ella. Podemos imaginar el proceso como un filtro interpuesto entre un proceso, que intenta alterar la imagen, y ésta. Las máscaras son imágenes binarias.



2.4 Transformación de círculo de Hough

Un círculo se representa matemáticamente como $(X - X_{centro})^2 + (Y - Y_{centro})^2 = r^2$ dónde (X_{centro}, Y_{centro}) el centro del círculo, y **r** es el radio del círculo. De la ecuación, podemos ver que tenemos 3 parámetros, por lo que necesitamos un acumulador 3D para una transformación grande, lo que sería muy ineficaz. Entonces OpenCV usa un método más complicado, el método de gradiente de Hough, que utiliza la información de gradiente de los bordes.

Usamos la función: **cv.HoughCircles** (image, circles, method, dp, minDist, param1 = 100, param2 = 100, minRadius = 0, maxRadius = 0)

Parámetros

- → **image** Imagen de entrada en escala de grises, de un solo canal y de 8 bits.
- → **círcles** vectores de salida de los círculos encontrados (tipo cv.CV_32FC3). Cada vector está codificado como un vector de punto flotante de 3 elementos (x, y, radio).
- → **method** método de detección (ver cv.HoughModes). Actualmente, el único método implementado es HOUGH_GRADIENT
- → **dp** relación inversa entre la resolución del acumulador y la resolución de la imagen. Por ejemplo, si dp = 1, el acumulador tiene la misma resolución que la imagen de entrada. Si dp = 2, el acumulador tiene la mitad de ancho y alto.

→ **minDist** distancia mínima entre los centros de los círculos detectados. Si el parámetro
es demasiado pequeño, es posible que se detecten erróneamente varios círculos vecinos además de uno verdadero. Si es demasiado grande, es posible que se omitan algunos círculos.

→ **parámetro1** primer parámetro específico del método. En el caso de HOUGH_GRADIENT,
es el umbral más alto de los dos pasados al detector de bordes Canny (el inferior es dos veces más pequeño).

→ **parámetro2** segundo parámetro específico del método. En el caso de HOUGH_GRADIENT, es el umbral del acumulador para los centros del círculo en la etapa de detección. Cuanto más pequeño sea, más círculos falsos se podrán detectar. Primero se devolverán los círculos correspondientes a los valores más grandes del acumulador.

→ **minRadius** radio mínimo del círculo.

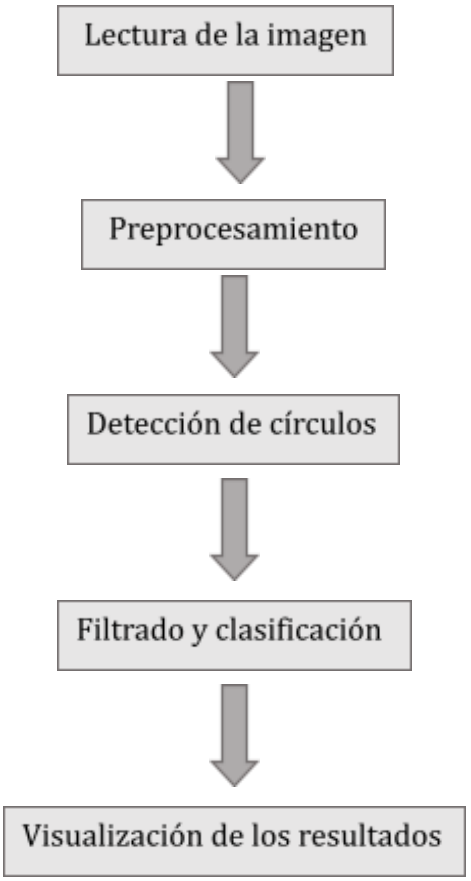
→ **maxRadius** radio máximo del círculo.

3. IMPLEMENTACIÓN

3.1 Explicación general del funcionamiento

El código desarrollado tiene como finalidad la detección automatizada de semáforos en imágenes digitales. Utilizando técnicas avanzadas de procesamiento de imágenes, se busca identificar y clasificar semáforos en función de sus colores característicos: rojo, verde y amarillo. La implementación se apoya en el cambio de espacio de color, la segmentación por color y la detección de círculos mediante el método de Hough, logrando así una robusta identificación de objetos circulares con colores específicos en una escena. El resultado final proporciona una representación visual de los semáforos detectados, resaltando su posición y color en la imagen original.

3.2 Diagrama de bloques del sistema



3.3 Explicación etapa por etapa

Lectura de la Imagen:

La primera etapa del código implica lectura una imagen de un directorio específico utilizando la biblioteca OpenCV.

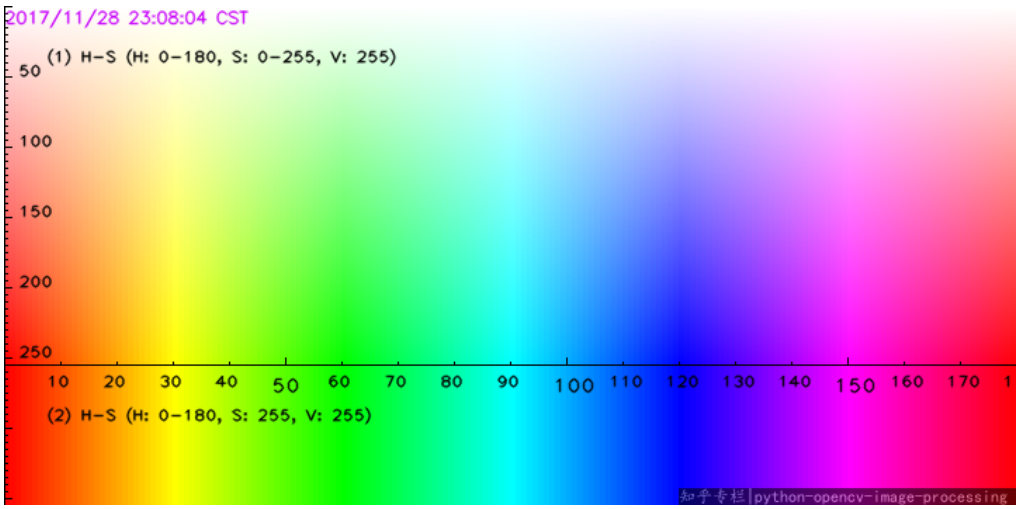
Preprocesamiento:

La imagen cargada se somete a un proceso de preprocesamiento. Cuando una imagen es leída en OpenCV, por defecto la lee en BGR, por ello en este paso se necesita ser transformada al espacio de color HSV.

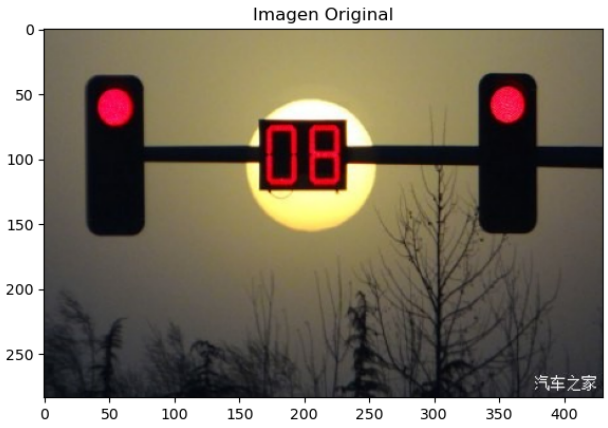
¿Por qué HSV? Se usa este espacio de color debido a que es más sencillo determinar el o los colores a detectar, para ello se emplea especial atención de la componente H, que corresponde al matiz. En OpenCV sus componentes pueden tomar los siguientes valores:

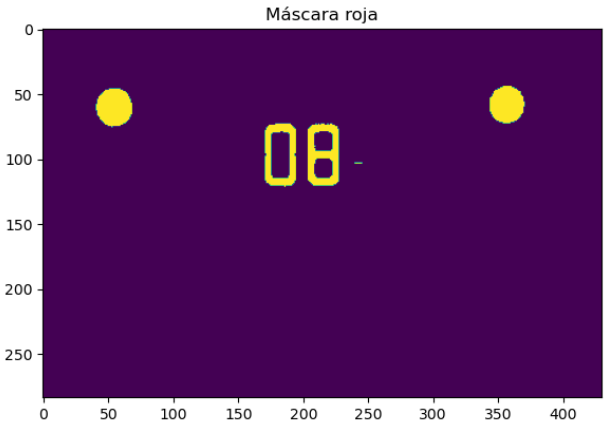
- H: 0 a 179
- S: 0 a 255
- V: 0 a 255

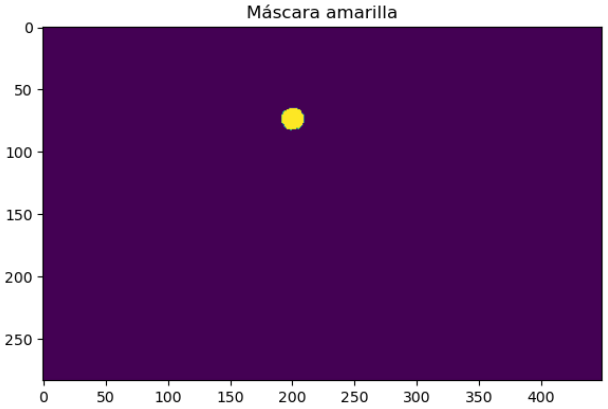
Como vemos en la figura el componente en H va de 0 a 179, y en este pasan colores de rojo, naranja, amarillo, verde, azul, violeta y nuevamente rojo, entonces vamos a determinar 2 rangos para el color rojo que está presente al principio y al final

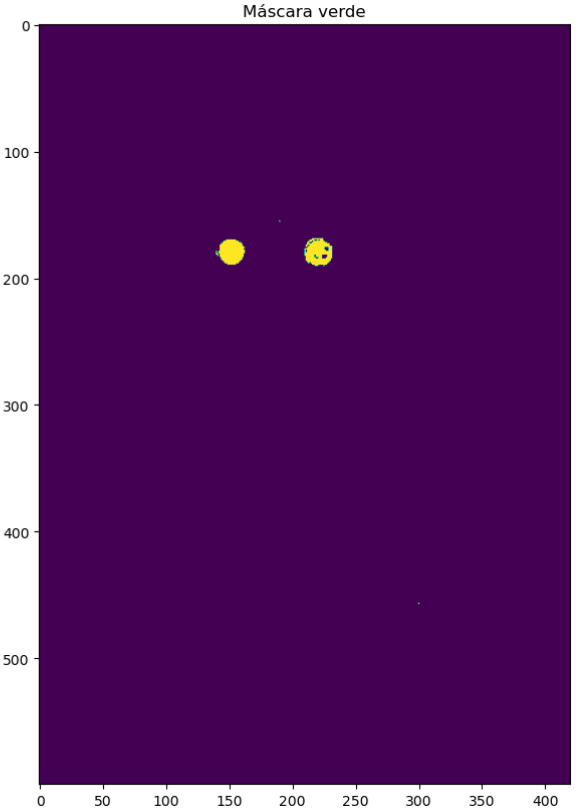


Definimos rangos de color para rojo, verde y amarillo en HSV. Luego, creamos máscaras para cada color basadas en los rangos definidos para identificar las áreas de color rojo, verde y amarillo en la imagen.









Detección de Círculos:

Se utiliza la función cv2.HoughCircles para detectar círculos en cada una de las máscaras creadas. La función de OpenCV devuelve un array de círculos detectados en la imagen. En este caso, los círculos detectados en la imagen se representan como un vector de tres elementos, donde el primer elemento es la coordenada x del centro, el segundo elemento es la coordenada y del centro, y el tercer elemento es el radio del círculo.

Filtrado y Clasificación:

Esta parte del código se encarga de procesar la información de los círculos detectados en la máscara correspondiente.

Se verifica si se han detectado círculos en la máscara. Si hay círculos, el código continúa; de lo contrario, no realiza ninguna acción.

Se redondean las coordenadas de los círculos detectados y se convierten a un tipo de dato entero.

Se itera sobre los círculos detectados. Para cada círculo:

- *Filtrado por Posición:* Se verifica que el círculo no esté fuera de los límites de la imagen o más allá de un cierto límite vertical definido por “limite”. Si está fuera de estos límites, se omite el círculo.
- *Cálculo de Intensidad:* Se realiza un cálculo de intensidad promedio (h / s) alrededor de la posición del círculo en la máscara. Esto implica sumar los valores de píxeles en una región definida por r alrededor del círculo.
- *Condición de Intensidad:* Si la intensidad promedio es mayor a un umbral, se considera que el círculo corresponde a una luz de color de la máscara a la que pertenecen.

Se clasifican como círculos rojos, verdes o amarillos según la máscara correspondiente.

Visualización de Resultados:

Los círculos detectados se resaltan en la imagen original. Además, cada círculo se etiqueta con su respectivo color (rojo, verde o amarillo). La imagen resultante muestra claramente la posición y clasificación de los semáforos en la escena.

Imágenes del resultado final

Fallas en el código:

Podemos ver que el código a veces ya sea por demasiadas luces de misma tonalidad por ejemplo en un embotellamiento como el de la foto, el código no logra encontrar puntos de interés cuando hacemos la máscara, la posible solución sería como los semáforos tienen cierta altura se encuentran separados de las demás luces (de autos) y podríamos aplicar una función que tenga en cuenta ese distanciamiento

En este otro caso cuando se hace la máscara quizás al no encontrar un círculo muy perfecto (por el ángulo de la fotografía) no se puede encontrar el círculo, una solución a esto sería implementar una herramienta que sea más permisiva y encuentre figuras que se parecen a círculos (como el de esta fotografía).

Como una autocrítica al trabajo en general podríamos implementar una cadena de filtros con estas herramientas planeadas y la detección de muchos semáforos , filtrando las luces de los automóviles, con esto creemos que podríamos lograr una eficiencia mayor al 90%

3.4 Métricas finales

En el desarrollo de nuestro sistema de detección de luces de tráfico, evaluamos su rendimiento utilizando un conjunto de pruebas compuesto por 20 imágenes. La métrica principal que utilizamos fue la precisión, definida como la proporción de luces de tráfico detectadas correctamente con respecto al total de luces en las imágenes de prueba. Nuestro sistema logró una precisión del 70%, identificando correctamente 14 de las 20 luces de tráfico presentes en el conjunto de prueba. Además, la tasa de detección, que representa la proporción de luces de tráfico detectadas con respecto al total real de luces en las imágenes, fue del 85%, indicando una capacidad sólida de identificación de luces en condiciones variadas.

4. CONCLUSIÓN

Podemos concluir que si bien la precisión no es >95% (muy fiable) , con la utilización de bibliotecas y funciones no muy complejas podemos lograr un programa funcional capaz de identificar un semáforo y si lo implementamos decirnos si deberíamos parar o seguir adelante en un sistema de navegación autónoma, que también acompañado de un sistema de detección de pavimento u otro tipo de camino podríamos lograr lo que se viene buscando desde hace mucho, que es que la máquina reemplace las tareas monótonas del humano y nos de tiempo de ocio, claramente en este caso si perfeccionamos un sistema de navegación autónoma deberíamos seguir prestando atención en la ruta pero no estaríamos tan estresados en viajes largos.

5. CÓDIGO Y MUESTRAS

Código

https://github.com/francescogallone/Deteccion-de-semaforos/tree/main

Resultados

https://drive.google.com/drive/folders/1G0O4FMu5oWJIKc9RWNujuH6XdaW2FJMU

6. BIBLIOGRAFÍA

Classroom Ingenia

https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html

https://e-archivo.uc3m.es/bitstream/handle/10016/18081/PFC_Victor_Alonso_Mendieta.pdf?sequence=1&isAllowed=y