Laboratorio de segmentación avanzada en imagen digital

Cristofher Solís, José David Soto
cristofhersj@gmail.com, josedavidsz@hotmail.com
Área académica de Ingeniería Mecatrónica
Instituto Tecnológico de Costa Rica

Resumen

En el presente informe se presenta el desarrollo del "Laboratorio de segmentación avanzada en imagen digital", el cual plantea un ejercicio de segmentación mediante múltiples criterios. Se documentan asimismo las decisiones tomadas con su respectiva justificación basándose en la teoría del curso MT9008 Sistemas de Visión.

Palabras clave

1.Brillo , 2.Contraste, 3.Imagen, 4.Reflexión, 5.Visión Por Computadora

I. Introducción

El laboratorio plantea la segmentación de una señal de tráfico que informa sobre la existencia de un paso peatonal en un fragmento de vídeo. Se recalca que debe emplearse una segmentación multicriterio, es decir, que la segmentación se base, como mínimo, a dos formas distintas.

II. Metodología y planteamiento

Se plantea utilizar como herramienta de desarrollo Python, para el tratamiento de las imágenes (operaciones de segmentación por ejemplo) se emplea la librería OpenCV la cuál consiste en una librería de libre uso que contiene funciones enfocadas en visión por computadora. Cabe destacar que para cada etapa se explica la metodología empleada para la resolución.

II-A. Obtención de fotogramas y estrategia de trabajo

Primeramente, considerando que el insumo brindado corresponde a un video, y debido a que la segmentación requerida es dirigida hacia una única imagen, se procede a dividir el vídeo en "frames" o cuadros. De esta forma se obtienen imágenes para trabajar en la segmentación solicitada. ¿Cómo se logra esto? Mediante Python se utiliza una función que permite extraer los cuadros de un video (pues se puede considerar que en lo que respecta a contenido **visual**, un video es una secuencia de imágenes), con la tasa de cuadros por segundo que se desee. Para este caso, debido a lo típico que resulta el uso de cámaras a 30FPS (frames per second) -y porque no se perciben 60FPS- se decide obtener 30 imágenes por cada segundo de video.

Esto da lugar a aproximadamente 60 imágenes, de las cuales solamente 26 presentan la señal de tránsito. Primeramente se pre-filtran las imágenes en función a dicho criterio, pues se determina que la señal DEBE aparecer en las imágenes utilizadas para segmentación. Luego de esto, se procede a seleccionar estratégicamente aquellas imágenes que no sean extremadamente conflictivas para la segmentación; para la segunda etapa de filtrado se opta por descartar aquellas imágenes en las que existe una presencia importante de reflejo de la luz del sol por lo que no se llega a apreciar la señal de tránsito como se desea. Un ejemplo de las imágenes utilizadas para llevar a cabo el desarrollo de dicha segmentación son las que se pueden apreciar en la figura 1, donde se recatan dos imágenes del conjunto de utilidad.

Figura 1: Imágenes utilizadas para la segmentación.Arriba imagen A, abajo imagen B.

Debido a que se desea realizar la segmentación sobre únicamente una imagen, se procede a trabajar con la imagen A, de la figura 1. Es decir, a partir de esta imagen se procederán con las técnicas de segmentación requeridas hasta lograr diferenciar a la señal de transito del resto de elementos en el fotograma.

II-B. Segmentación por color

Al observar la figura 1, se distingue claramente que el color de la señal, el cual es azul, es distinto con respecto a los elementos más próximos, aunque esto resulta un poco más complicado de aseverar con respecto al cielo. Aún así, el color resulta entonces un método interesante para poder segmentar la señal de tránsito en la imagen, considerando que pareciese presentar un conjunto particular de tonalidad, saturación e intensidad, y se procede a segmentar por color. Con esto se espera aislar en gran medida la señal de otros elementos no deseados, que no comparten las características del color con la imagen A.

Para trabajar con el color, primeramente se desea transformar la imagen al espacio HSI ($Hue-Saturation-Intensity$), pues resulta más flexible y útil de trabajar para segmentar la imagen, ya que se puede distinguir entre tono, saturación e intensidad. El canal de tono resulta útil para obtener las tonalidades azules, el canal de saturación permite obtener una coloración más o menos pura según se desee, para este caso se aprecia que el azul de la señal presenta una pureza relativamente alta. En lo referente al color como tal, el canal de intensidad no cobra mayor rol para distinguir entre colores.[1]

Por tanto, se procede a transformar al espacio HSI la imagen A, y sobre esta transformación aplicar las operaciones necesarias para obtener la parte azul de la señal de tránsito.

Una vez transformada a este espacio de colores, antes de proceder con el procedimiento de segmentación por color, resulta útil tener un punto de referencia para escoger el rango de valores que se desean para el tono (H) y la saturación (S). Por tanto, mediante $Paint$ se selecciona un píxel dentro de la señal de tránsito con el propósito de conseguir los valores HSI que le describen. Resulta IMPRESCINDIBLE destacar que el software utilizado emplea para los canales HSI el siguiente rango de valores: de 0 a 240 (exceptuando el tono, cuyo límite superior se fija en 239). El significado sigue siendo el mismo y se puede valorar de forma proporcional con otro $software$ que emplee otra escala. Se obtiene lo siguiente:

Color pixel (HSI): 139, 240, 120 (MS Paint Standard for HSI)

Dentro de las consideraciones que se deben tener en este laboratorio es que, como se menciona previamente, se emplea OpenCV como librería para procesar las imágenes y generar la segmentación. Al querer realizar la segmentación por color

con esta herramienta se debe contemplar el rango de valores para los cuales se definen los canales HSI. De forma ilustrativa, en la figura 2, se aprecia visualmente el rango de valores para los cuales se definen los tres canales HSI.

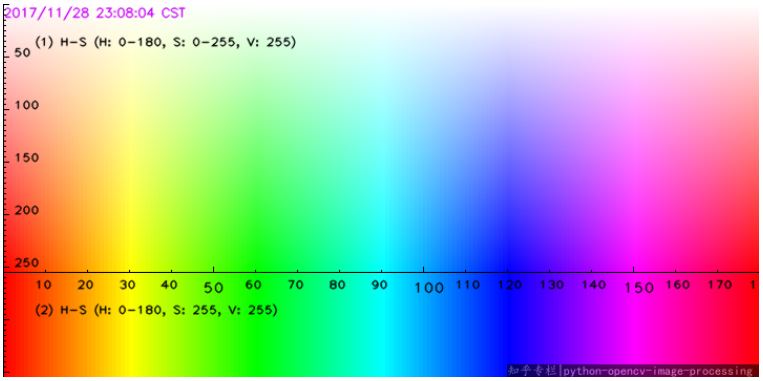

Figura 2: Ilustración de la escala del espacio HSI empleada por la librería OpenCV. [2]

Por tanto resulta oportuno destacar que las tonalidades azules se pueden distinguir entre los valores 90 y 130, y el canal S se explica como: a mayor pureza, el valor es mayor, siendo el máximo 255. Por tanto, considerando ambas referencias expuestas, se proceden a realizar iteraciones de segmentaciones, ajustando en cada una los valores de los canales H y S según se desee (los límites para el canal de intensidad son de 0 a 255 para todos los casos). El propósito del ejercicio es obtener una segmentación que -en la medida de lo posible- que obtenga la parte azul de la señal de tránsito, dejando por fuera inclusive otros elementos con tonalidades similares (como por ejemplo, el cielo).

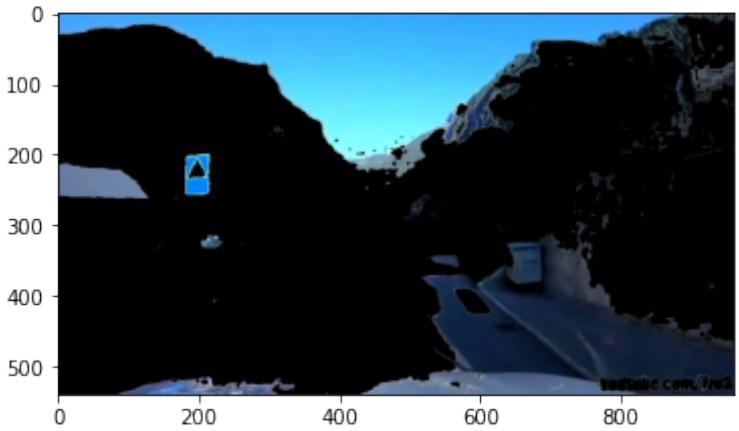

Figura 3: Segmentación de la imagen A en HSI, empleando H:[90, 130] y S: [50, 255].

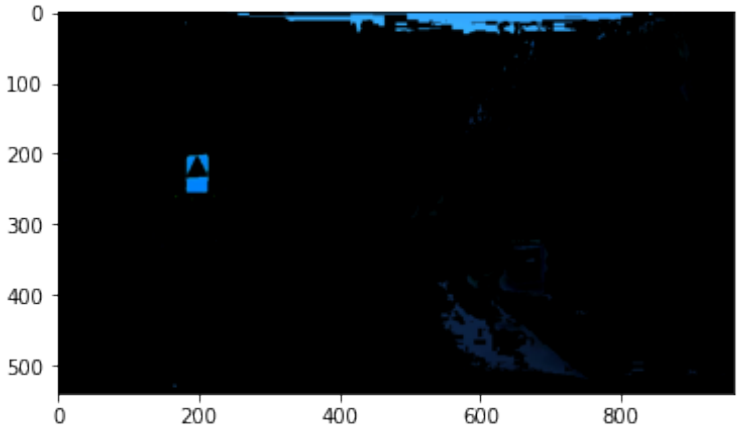

Figura 4: Segmentación de la imagen A en HSI, empleando H:[90, 130] y S: [200, 255].

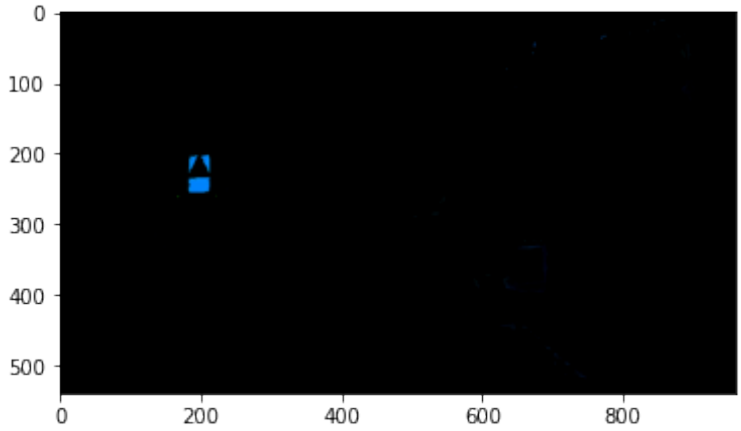

Figura 5: Segmentación de la imagen A en HSI, empleando H:[90, 130] y S: [240, 255].

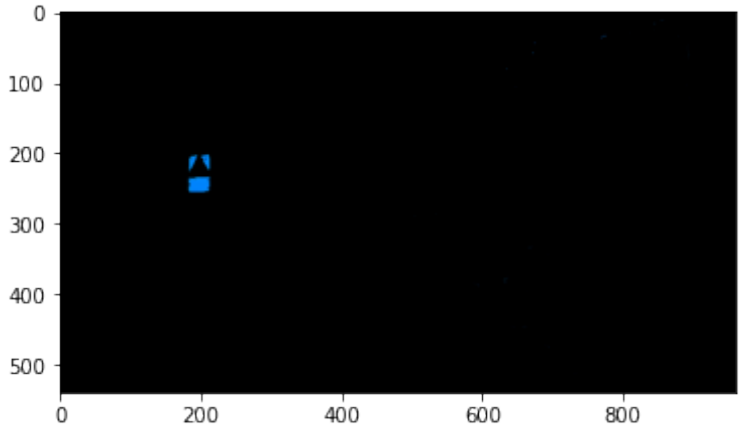

Figura 6: Segmentación de la imagen A en HSI, empleando H:[100, 105] y S: [240, 255].

Se puede apreciar en las figuras 3, 4, 5 y 6 como, progresivamente, conforme se ajusta el valor de la saturación, se produce una segmentación más acorde a lo deseado. Se opta por dejar los valores de H: [100, 105], S: [240, 255]. El canal I del resultado obtenido (se aprecia en la figura 7). Se trata dicha imagen mediante transformación morfológica para obtener los resultados más puros posibles (evitando la intromisión de cualquier tipo de valor no deseado o ruido).

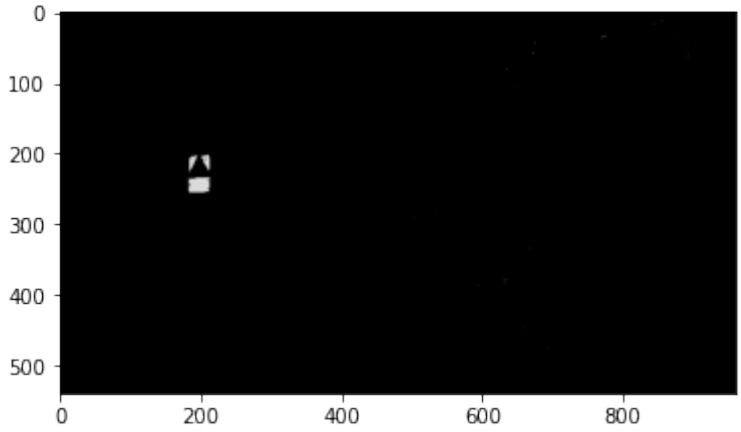

Figura 7: Canal de intensidad de la imagen segmentada en HSI, empleando H:[100, 105] y S: [240, 255].

El proceso de transformación morfológica denominada "$Opening$" consiste en realizar una erosión (delimita los bordes de los objetos al "$erosionar$" los límites y así se obtiene una imagen más limpia de ruido blanco), seguido de un proceso

de dilatación (Como la erosión de una figura vuelve más pequeña la imagen, este proceso hace un efecto contrario, en el cual se expande la imagen pero como ahora no se encuentra el ruido que se erosionó, este no va a volver). Para lograr esta transformación morfológica se utiliza la función cv2.morphologyEx(), la cual utiliza como entradas la imagen a la que se le desea realizar la transformación, el tipo de transformación (en este caso "$cv2.MORPH_OPEN$") y el tamaño del filtro o kernel a utilizar.[3]

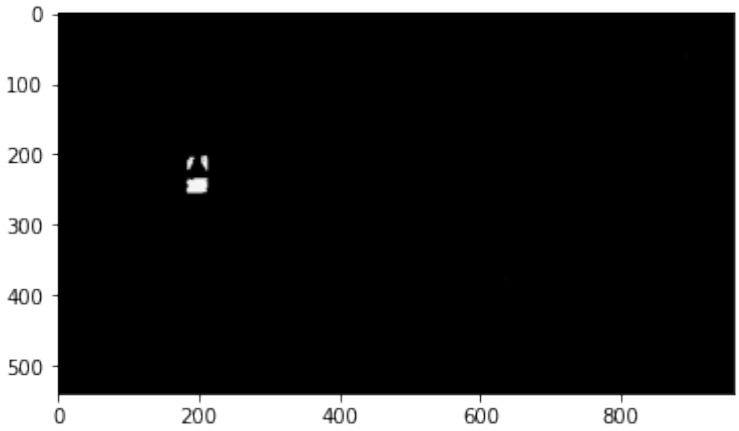

Figura 8: Imagen segmentada por color con transformación morfológica $Opening$.

En la figura 8 se aprecia el resultado obtenido al aplicar la transformación morfológica $Opening$ a la figura 7. Con esto se da por satisfecha esta etapa de segmentación por color. Cabe destacar que no se recupera en totalidad la sección azul de la señal de tránsito, sin embargo esto se acepta puesto que se desea evitar introducir otros elementos azules.

II-C. Segmentación mediante detección de bordes

La segunda estrategia para segmentar la señal de transito en las imágenes obtenidas a través del vídeo es el utilizar los bordes de dicha señal con el fin de utilizar su forma para detectarla en las imágenes. Para esta detección de bordes se requirieron de múltiples pasos, inicialmente se estudió el histograma de la imagen original (En formato de tonos de gris) con el fin de detectar un valor de tonalidad útil para definir un $threshold$ con el cual se pueda binarizar esta figura (Para efectos de mejor comprensión, esta imagen se denominará "Im"), esto se puede apreciar en la figura 9, debido a que se denotan dos picos importantes, se selecciona un punto intermedio que permita separar ambas secciones y así obtener una imagen binaria en la que se puedan denotar los bordes de los objetos con mayor facilidad. En este caso se seleccionó el valor de tonalidad de 120, el cual da origen a la figura 10 en la cual se puede denotar con suficiente claridad los bordes de la señal de transito deseada.

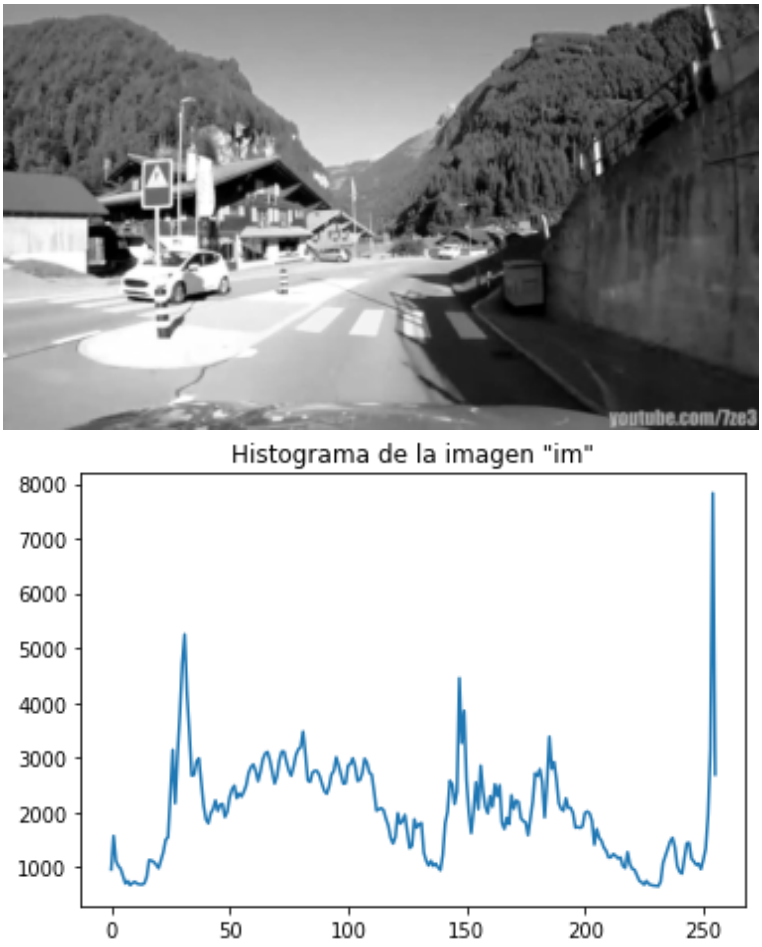

Figura 9: Imagen utilizada para la segmentación "Im" en tonos de gris y su histograma respectivo.

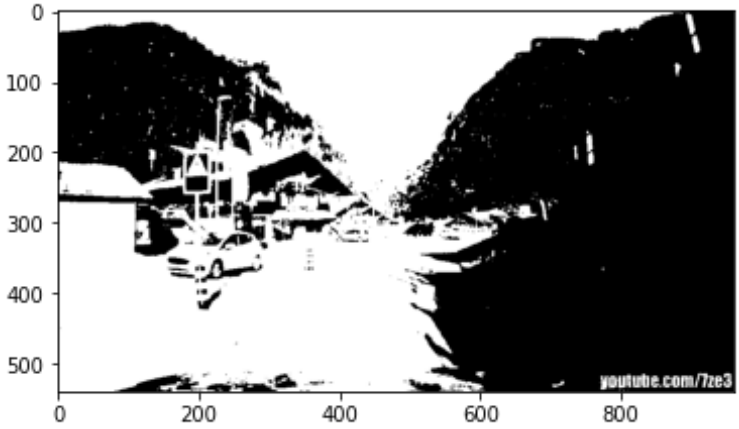

Figura 10: Imagen "Im" binarizada.

El siguiente paso por realizar es obtener los bordes de la imagen binarizada, por lo que se optó por realizar una operación morfológica de tipo gradiente, esta función se encarga de resaltar cuando hay cambios importantes en la frecuencia de la imagen, por lo que se obtiene una nueva figura que resalta únicamente los bordes de los objetos, tal y como se puede apreciar en la figura 11. Con esta figura en conjunto con el resultado obtenido con la segmentación por color, es posible el obtener una imagen que resalte el contorno de la señal de transito deseada. [3]

Figura 11: Imagen "Im" una vez se le aplicó un gradiente morfológico.

II-D. Combinación de resultados y segmentación final

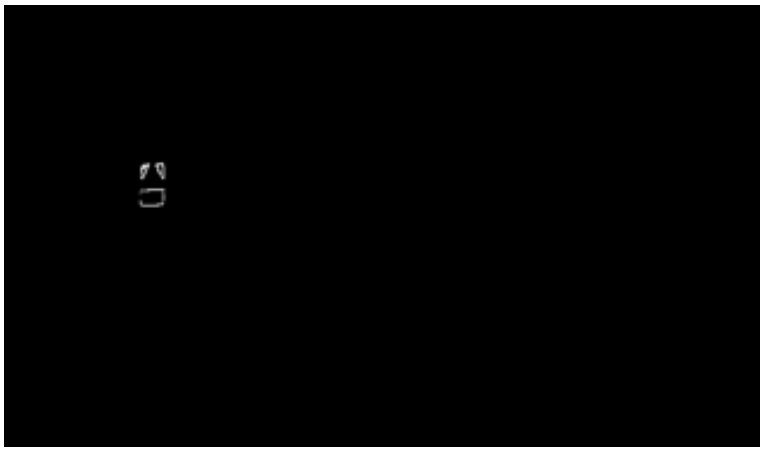

Figura 12: Combinación de los resultados de ambos tipos de segmentación utilizados mediante una operación lógica AND.

Una vez obtenidos los resultados de ambos tipos de segmentación, se hizo una operación lógica AND, con la cual se obtiene una imagen que resalta los límites azules de la señal de tránsito y descarta toda la información que no sea perteneciente a este objeto, tal y como se presenta en la figura 12. Es decir, se obtienen los elementos en común entre la segmentación por color y los cambios de intensidad detectados por la segmentación por bordes, dando a lugar a los bordes de los elementos azules de la señal. Esto representa gran parte de la señal y es un insumo suficiente para la última operación de la segmentación, la cual dará lugar a un rectángulo que encierra la señal de tránsito.

Esta última operación se describe a continuación; nótese que actualmente se presentan contornos individuales separados, considerando que la forma de la señal es rectangular, se puede construir un solo contorno a partir de los contornos obtenidos, mediante la dilatación de estos y su respectiva unificación (fusión). Se filtran según el área formada por la fusión de contornos y se filtran las áreas pequeñas (según un umbral previamente definido, en este caso se utiliza de forma plana en 500 porque es apropiado con el tamaño de la señal, sin embargo, debe considerarse que esto es relativo a las dimensiones de la señal y los contornos dibujados por esta). Se obtienen las coordenadas del contorno obtenido y finalmente se traza sobre la imagen original un rectángulo que indica dónde se ubica la señal (las dimensiones de dicho rectángulo se definen en función de las proporciones del contorno mencionado anteriormente). Todo esto da a lugar a la figura 13 en donde se aprecia el resultado final de la segmentación.

Figura 13: Imagen final que segmenta correctamente la posición de la señal de tránsito.

III. Validación de resultados

En la sección de anexos se puede apreciar la figura 15, en la cual se presentan múltiples fotogramas los cuales al ser cargados por el algoritmo generado, se obtiene una salida que segmenta de forma exitosa la señal de transito deseada. A pesar de que esta se encuentre en localizaciones distintas de la imagen. Por tanto, resulta exitoso el ejercicio de segmentar la señal de tránsito mediante el algoritmo computacional implementado, cuya arquitectura de alto nivel puede observarse en la figura 14.

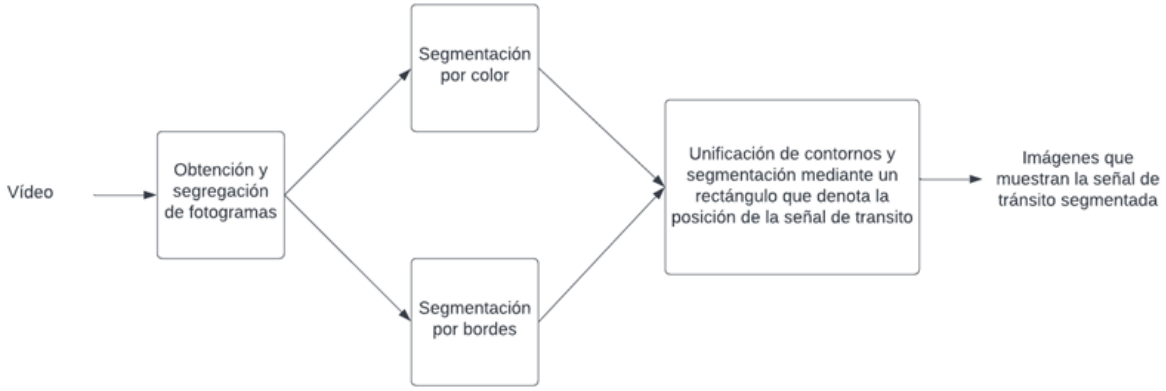

Figura 14: Arquitectura para el algoritmo confeccionado.

IV. Conclusiones

- Simular visión humana, ventaja de coloración de imagen
- La segmentación multicriterio es una estrategia potente a tomar en consideración cuando se tratan imágenes en las que se encuentran múltiples objetos y se quiere enfocar uno o varios en específico.
- El uso de espacios de colores como el HSI permite una segmentación por color efectiva que disminuye el riesgo de generar ruido en exceso al haber varias tonalidades similares en una imagen.
- Las transformaciones morfológicas permiten filtrar ruido de distintos tipos lo cual permite un mejor estudio de las imágenes con las que se va a trabajar.
- La segmentación por bordes permite construir una representación de los cambios de intensidad en una imagen.
- Es posible unificar bordes adyacentes para construir un contorno con el cual sea posible definir una operación (como por ejemplo encerrar el contorno del objeto en un rectángulo).

Referencias

[1] Mark S. Nixon and Alberto S. Aguado. “Feature Extraction Image Processing for Computer Vision”. Academic Press, 2012
[2] S. Fourk. Çhoosing the correct upper and lower HSV boundaries for color detection with‘cv::inRange‘ (OpenCV)”. StackOverflow, 2012.
[3] S. Gollapudi, *Learn computer vision using OpenCV*. Berkeley, CA: Apress, 2019.

V. ANEXOS

V-A. Resultados adicionales

Figura 15: Salidas al ingresar diferentes fotogramas del vídeo.

V-B. Código

Preferiblemente ingresar al GoogleColab.
(desde: https://colab.research.google.com/drive/1tdqVtRFVc7iJntM39BNsKft3esBl5oi?usp=sharing)

```
# -*- coding: utf-8 -*-
"""Laboratorio 2 Visi n.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1tdqVtRFVc7iJntM39BNs_Kft3esBl5oi

# Parte 1: Obtener frames clave del v deo
"""

from datetime import timedelta
import cv2
import numpy as np
import os
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

# i.e if video of duration 30 seconds, saves 10 frame per second = 300 frames saved in total
SAVING_FRAMES_PER_SECOND = 30

#ref: https://www.thepythoncode.com/article/extract-frames-from-videos-in-python
#ref: https://www.youtube.com/watch?v=b-WViLMs_4c
def format_timedelta(td):
    """Utility function to format timedelta objects in a cool way (e.g 00:00:20.05)
    omitting microseconds and retaining milliseconds"""
    result = str(td)
    try:
        result, ms = result.split(".")
    except ValueError:
        return result + ".00".replace(":", "-")
    ms = int(ms)
    ms = round(ms / 1e4)
    return f"{result}.{ms:02}".replace(":", "-")


def get_saving_frames_durations(cap, saving_fps):
    """A function that returns the list of durations where to save the frames"""
    s = []
    # get the clip duration by dividing number of frames by the number of frames per second
    clip_duration = cap.get(cv2.CAP_PROP_FRAME_COUNT) / cap.get(cv2.CAP_PROP_FPS)
    # use np.arange() to make floating-point steps
    for i in np.arange(0, clip_duration, 1 / saving_fps):
        s.append(i)
    return s

def video_capture(video_file):
    filename, _ = os.path.splitext(video_file)
    filename += "-opencv"
    # make a folder by the name of the video file
    if not os.path.isdir(filename):
        os.mkdir(filename)
    # read the video file
```

```
    cap = cv2.VideoCapture(video_file)
    # get the FPS of the video
    fps = cap.get(cv2.CAP_PROP_FPS)
    # if the SAVING_FRAMES_PER_SECOND is above video FPS, then set it to FPS (as maximum)
    saving_frames_per_second = min(fps, SAVING_FRAMES_PER_SECOND)
    # get the list of duration spots to save
    saving_frames_durations = get_saving_frames_durations(cap, saving_frames_per_second)
    # start the loop
    count = 0
    while True:
        is_read, frame = cap.read()
        if not is_read:
            # break out of the loop if there are no frames to read
            break
        # get the duration by dividing the frame count by the FPS
        frame_duration = count / fps
        try:
            # get the earliest duration to save
            closest_duration = saving_frames_durations[0]
        except IndexError:
            # the list is empty, all duration frames were saved
            break
        if frame_duration >= closest_duration:
            # if closest duration is less than or equals the frame duration,
            # then save the frame
            frame_duration_formatted = str(count)
            cv2.imwrite(os.path.join(filename, f"{frame_duration_formatted}.jpg"), frame)
            # drop the duration spot from the list, since this duration spot is already saved
            try:
                saving_frames_durations.pop(0)
            except IndexError:
                pass
        # increment the frame count
        count += 1

video_capture("video-carretera-suiza.mp4")

!zip -r /content/sample_data.zip /content/video-opencv

"""# Parte 2: Segmentaci n multicriterio"""

#Se cargan las imagenes con las que se van a trabajar
#img1 = plt.imread('19.jpg')
img1 = plt.imread('Frame 5.jpg')
plt.imshow(img1)
#img2 = plt.imread('20.jpg')
#plt.imshow(img2)

hsv_img1 = cv2.cvtColor(img1, cv2.COLOR_RGB2HSV)
# HSI 139, 240, 120
# RGB 0, 132, 255

#ITERACION 1
# define range of blue color in HSV
# c mo definir l mites
lower_blue = np.array([90,50,0])
upper_blue = np.array([130,255,255])
```

```
# Threshold the HSV image to get only blue colors
mask = cv2.inRange(hsv_img1, lower_blue, upper_blue)

# Bitwise-AND mask and original image
res = cv2.bitwise_and(img1,img1, mask= mask)
plt.imshow(res)

#ITERACION 2
# define range of blue color in HSV
# c mo definir l mites
lower_blue = np.array([90,200,0])
upper_blue = np.array([130,255,255])
# Threshold the HSV image to get only blue colors
mask = cv2.inRange(hsv_img1, lower_blue, upper_blue)

# Bitwise-AND mask and original image
res = cv2.bitwise_and(img1,img1, mask= mask)
plt.imshow(res)

#ITERACION 3
# define range of blue color in HSV
# c mo definir l mites
lower_blue = np.array([90,240,0])
upper_blue = np.array([130,255,255])
# Threshold the HSV image to get only blue colors
mask = cv2.inRange(hsv_img1, lower_blue, upper_blue)

# Bitwise-AND mask and original image
res = cv2.bitwise_and(img1,img1, mask= mask)
plt.imshow(res)

#ITERACION 4
# define range of blue color in HSV
# c mo definir l mites
lower_blue = np.array([100,240,0])
upper_blue = np.array([105,255,255])
# Threshold the HSV image to get only blue colors
mask = cv2.inRange(hsv_img1, lower_blue, upper_blue)

# Bitwise-AND mask and original image
res = cv2.bitwise_and(img1,img1, mask= mask)
plt.imshow(res)

#gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
##plt.imshow(gray,cmap='gray')
#th, img1_bin = cv2.threshold(gray, 235, 255, cv2.THRESH_TOZERO)
#plt.imshow(img1_bin,cmap='gray')

res_gray = cv2.cvtColor(res, cv2.COLOR_RGB2GRAY)
histr =  cv2.calcHist([res_gray],[0],None,[256],[0,256])

plt.plot(histr[0:10])
plt.title('Histograma de la imagen "im"')
plt.show()
plt.imshow(res_gray,cmap='gray')

#Morph
kernel = np.ones((3,3),np.uint8)
```

```
contorno = cv2.morphologyEx(res_gray, cv2.MORPH_OPEN, kernel)
plt.imshow(contorno,cmap='gray')

img1_gray = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
# Grafica el histograma generado anteriormente

histr = cv2.calcHist([img1_gray],[0],None,[256],[0,256])
plt.plot(histr)
plt.title('Histograma de la imagen "im"')
plt.show()
plt.imshow(img1_gray,cmap='gray')

#Detectar bordes de imagen original
#Primero binarizar
th, img1_bin = cv2.threshold(img1_gray, 120, 255, cv2.THRESH_BINARY)
plt.imshow(img1_bin, cmap='gray')

#Aplicar morfol gico
kernel = np.ones((5,5),np.uint8)
gradient = cv2.morphologyEx(img1_bin, cv2.MORPH_GRADIENT, kernel)

plt.imshow(gradient, cmap='gray')

img_merge = gradient & contorno
plt.imshow(img_merge, cmap='gray')

# ref: https://stackoverflow.com/questions/71739517/detect-squares-paintings-in-images-and-drav
# Two pass dilate with horizontal and vertical kernel
horizontal_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (9,5))
dilate = cv2.dilate(img_merge, horizontal_kernel, iterations=2)
vertical_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5,9))
dilate = cv2.dilate(dilate, vertical_kernel, iterations=2)

# Find contours, filter using contour threshold area, and draw rectangle
cnts = cv2.findContours(dilate, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
for c in cnts:
    area = cv2.contourArea(c)
    if area > 500:
        x,y,w,h = cv2.boundingRect(c)
        segmentacion = cv2.rectangle(img1, (x, y), (x + w, y + h), (36, 255, 12), 5)

plt.imshow(segmentacion)
```