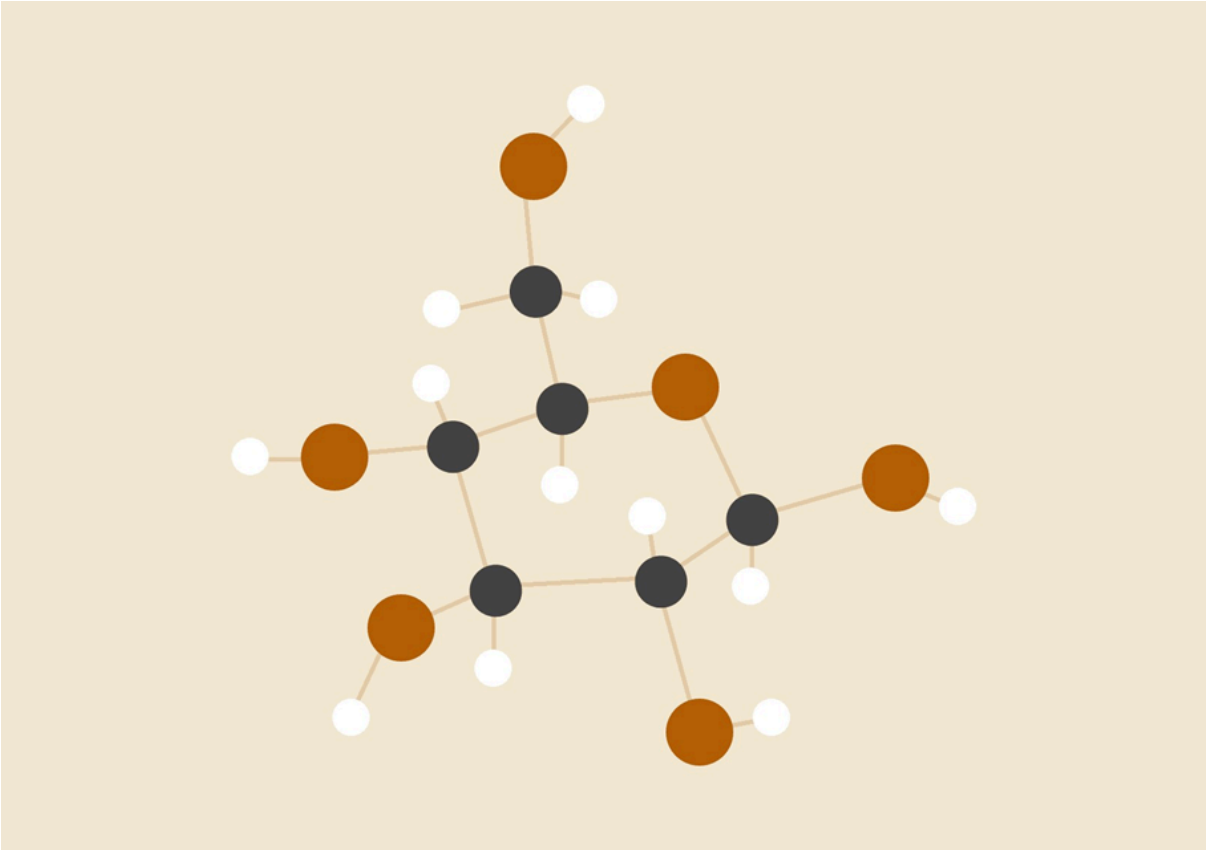
TRABAJO PRÁCTICO 3

IA4.4 Procesamiento Digital de Imágenes
Tecnicatura Universitaria en Inteligencia Artificial



Grupo 06:
Arce, Sofía
Gauto, Lucas
Rizzotto, Camila

02/07/2024
Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura

INTRODUCCIÓN

Nuestro trabajo se basa en la detección del carril de una ruta y generar un video nuevo donde se muestran las líneas que definen el carril

DESCRIPCIÓN DEL ENTORNO DE TRABAJO

Para la realización de estos ejercicios utilizamos el lenguaje Python y generamos un entorno virtual en el cual instalamos y utilizamos los siguientes paquetes:

- Numpy
- Open CV
- Matplotlib

EJERCICIO 1

Funciones:

calcular_linea_promedio:

- **Argumento**:
  - **lineas:** Array de las coordenadas que representan las líneas

```
def calcular_linea_promedio(lineas):
    x_coords = []
    y_coords = []
    for linea in lineas:
        x_coords.extend([linea[0][0], linea[0][2]])
        y_coords.extend([linea[0][1], linea[0][3]])
    if len(x_coords) == 0:
        return None
    coeficientes = np.polyfit(y_coords, x_coords, 1)
    pendiente = coeficientes[0]
    intercepto = coeficientes[1]
    return pendiente, intercepto
```

La función itera sobre cada línea en la lista lineas. Cada línea está representada como un array con cuatro elementos que corresponden a las coordenadas (x, y) de los puntos inicial y final de la línea. Extraemos las coordenadas de los puntos iniciales y finales de las líneas y ajustamos un polinomio de grado 1 (una línea recta) a los puntos. El resultado del ajuste son los coeficientes de la línea recta que mejor se ajusta a los puntos, donde coeficientes[0] es la pendiente y coeficientes[1] es el intercepto

dibujar_linea_extrapolada:

- **Argumentos:**
  - **frame:** donde se dibuja la línea
  - **pendiente**: previamente calculada
  - **intercepto**: previamente calculado

- **y1, y2:** los puntos donde empieza y termina la línea original
- **color**: 'red', 'green', 'blue', 'yellow', 'white', 'black', entre otros.
- **grosor**: Entero que determina el grosor de la línea a dibujar

```
def dibujar_linea_extrapolada(frame, pendiente, intercepto, y1, y2,
color, grosor):
    x1 = int(pendiente * y1 + intercepto)
    x2 = int(pendiente * y2 + intercepto)
    cv2.line(frame, (x1, y1), (x2, y2), color, grosor)
```

En esta función utilizamos la pendiente y el intercepto calculados previamente para determinar las coordenadas x de dos puntos dados (y1 y y2) y dibujamos una línea recta extrapolada para extender o proyectar la línea detectada más allá de los puntos dados y1 y y2 en el marco de imagen frame.

Archivo: main_tp_3.py

Cargamos el video

```
cap = cv2.VideoCapture('ruta_2.mp4')
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))
n_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

ret, frame = cap.read()
```

Creamos una máscara binaria del mismo tamaño que cada fotograma del video. Esto lo hicimos para seleccionar la ROI dentro de cada recuadro del video.

```
# Crear una máscara binaria del mismo tamaño que cada fotograma del
video
mask = np.zeros((height, width), dtype=np.uint8)
```

Definimos los puntos de la región de interés:

```
# Coordenadas del trapecio
puntos = np.array([[130,534],[905,534], [510, 316],[452,316]],
dtype=np.int32)
```

Los píxeles blancos en la máscara indican las áreas del cuadro del video que están dentro del trapecio y que serán consideradas para procesamiento posterior.

Procesamos el video:

Durante la iteración, a través de los cuadros del video, se aplica esta máscara:

```
        #---- Proceso ----------------------------------------
        frame_HSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        cv2.fillPoly(mask, [puntos], 255)
```

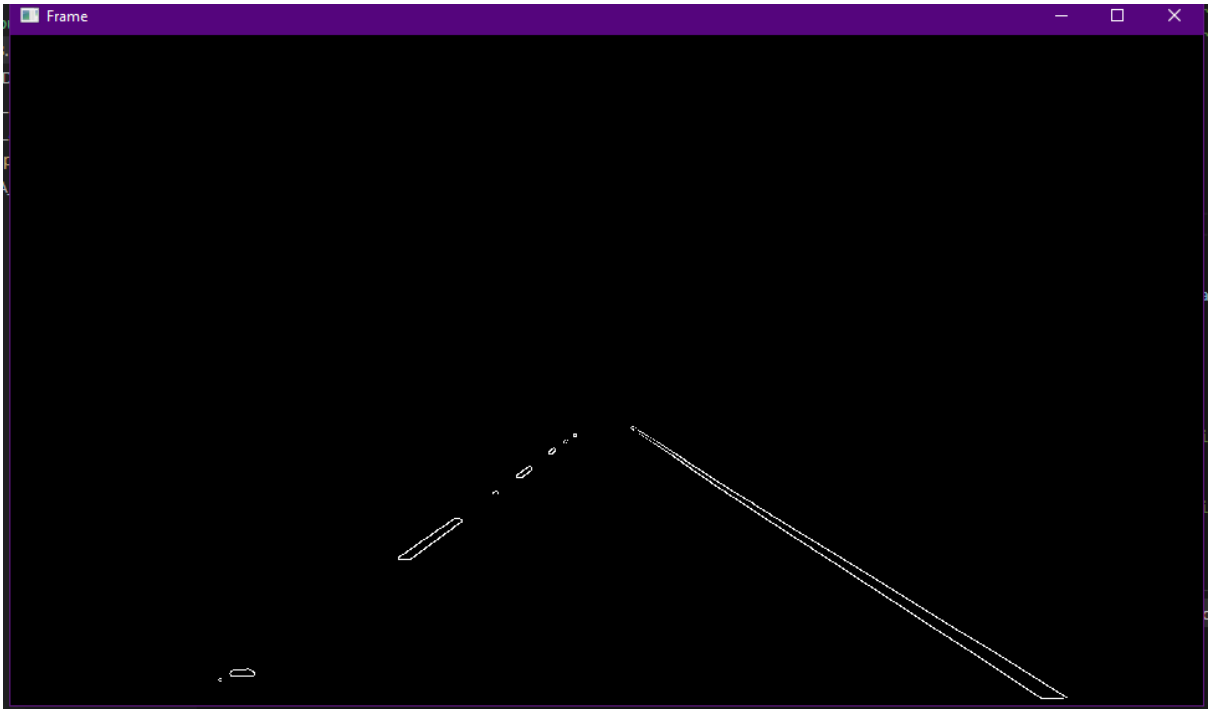
para "recortar" la región de interés del cuadro original que establecimos anteriormente

Después, nos quedamos solo con los píxeles dentro del trapecio definido (marcados como blancos en la máscara)

```
        # Aplicar la máscara a la imagen
        masked_image = cv2.bitwise_and(frame_HSV, frame_HSV, mask=mask)
        frame_threshold = cv2.inRange(masked_image, (0, 0, 171), (255,
255, 255))
```

Detectamos los bordes en la imagen binaria usando Canny y luego marcamos las líneas usando la transformada de Hough probabilística.

```
        edges = cv2.Canny(frame_threshold, 100, 170, apertureSize=3)
        lines = cv2.HoughLinesP(edges, rho=1, theta=np.pi/180*0.2,
threshold=50, minLineLength=5, maxLineGap=200)
```



Finalmente, marcamos las líneas de los carriles detectadas de la derecha y de la izquierda. Calculando la pendiente entre lso puntos finales de cada segmento detectado

```
        if lines is not None:
            lineas_izquierda = []
            lineas_derecha = []
            for linea in lines:
                x1, y1, x2, y2 = linea[0]
                pendiente = (y2 - y1) / (x2 - x1) if x2 - x1 != 0 else
np.inf
                if pendiente < 0:
                    lineas_izquierda.append(linea)
                else:
                    lineas_derecha.append(linea)
```

Ahora calculamos para ambas líneas del carril una línea promedio con la función que previamente definimos. Nos quedamos con los puntos finales e iniciales de la línea a dibujar y la dibujamos en el frame. Finalmente superponemos frame_copy y frame con igual intensidad (50% cada una) para conservar las líneas originales de los carriles y las dibujadas.

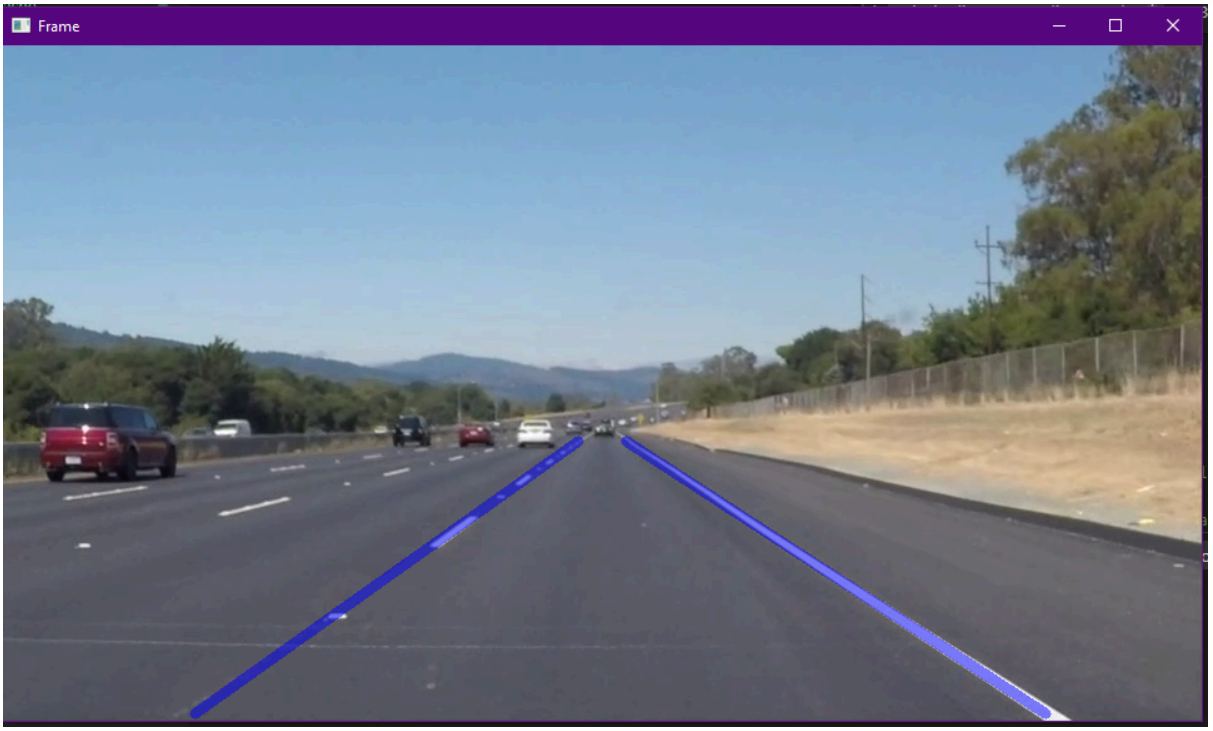
```
            # Calcular y dibujar la línea promedio para la izquierda
            pendiente_izquierda, intercepto_izquierda =
calcular_linea_promedio(lineas_izquierda)
            if pendiente_izquierda is not None:
                y1, y2 = puntos[2][1], puntos[0][1]
                frame_copy = frame.copy()
                dibujar_linea_extrapolada(frame_copy,
pendiente_izquierda, intercepto_izquierda, y1, y2, (255, 0, 0), 8)
                cv2.addWeighted(frame_copy, 0.5, frame, 0.5, 0, frame)

            #Calcular y dibujar la línea promedio para la derecha
            pendiente_derecha, intercepto_derecha =
calcular_linea_promedio(lineas_derecha)
            if pendiente_derecha is not None:
                y1, y2 = puntos[3][1], puntos[1][1]
                frame_copy = frame.copy()
                dibujar_linea_extrapolada(frame_copy,
pendiente_derecha, intercepto_derecha, y1, y2, (255, 0, 0), 8)
                cv2.addWeighted(frame_copy, 0.5, frame, 0.5, 0, frame)
```

Mostramos el resultado

```
        cv2.imshow('Frame',frame)
#Muestro el frame
        if cv2.waitKey(25) & 0xFF == ord('q'):
#Corto la repoducción si se presiona la tecla "q"
            break
    else:
        break                                       #Corto la
reproducción si ret=False, es decir, si hubo un error o no quedán mas
frames.
cap.release()                   #Cierro el video
cv2.destroyAllWindows()
```

RESULTADO FINAL:

Video: ruta_1



Video: ruta_2


Frame