Detección de Carriles en la Seguridad Vial

MARTÍN IGLESIAS LÓPEZ

Contenido

INTRODUCCIÓN

En el ámbito de la visión artificial, la detección de carriles en carreteras se ha convertido en una tecnología crucial para mejorar la seguridad y la autonomía en los sistemas de conducción. Este proyecto, desarrollado con OpenCV, tiene como objetivo principal implementar un programa capaz de identificar los carriles en una carretera y calcular la distancia de estos con respecto al vehículo, lo cual proporciona una valiosa información para la conducción asistida y autónoma.

El sistema utiliza técnicas de procesamiento de imágenes y algoritmos de detección de bordes, aplicados sobre secuencias de video capturadas en tiempo real.

Además de la detección de carriles, el programa calcula la distancia lateral entre el vehículo y las líneas detectadas, generando métricas útiles para ajustar la posición del automóvil dentro del carril.

La implementación de esta herramienta representa un avance en el desarrollo de sistemas de asistencia a la conducción, mejorando la capacidad de respuesta y la seguridad en tiempo real.

2. Herramientas

Para este proyecto de detección de carriles y cálculo de distancias respecto al vehículo se han empleado tres tecnologías fundamentales: OpenCV, NumPy y Python.

OpenCV es esencial en la visión por computadora, ya que proporciona las herramientas necesarias para realizar detección de bordes, filtrado de colores y transformaciones geométricas, facilitando así la identificación precisa de los carriles en la carretera.

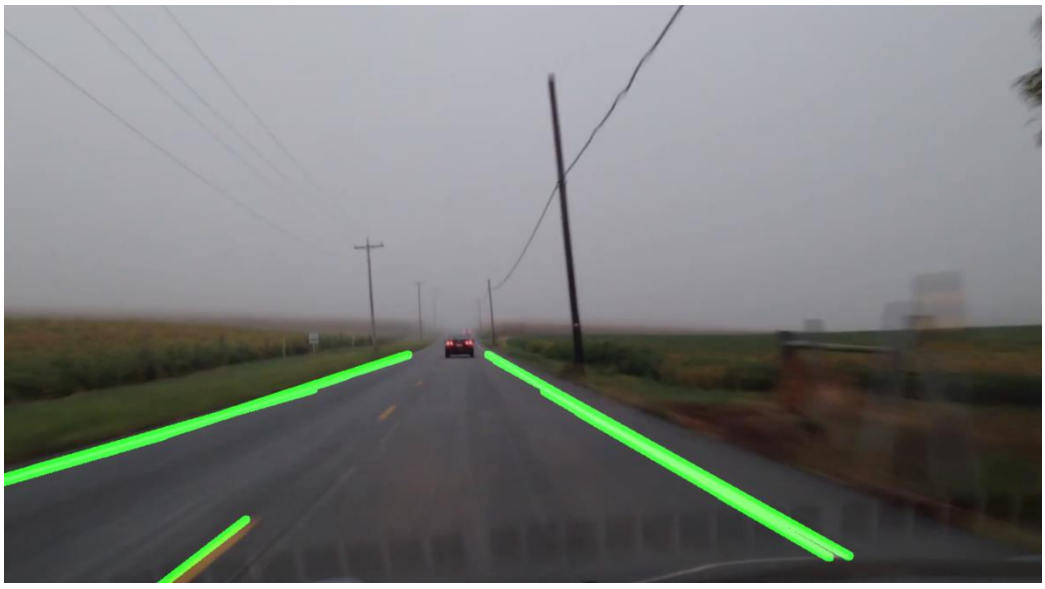
NumPy permite el manejo eficiente de arreglos y operaciones matemáticas sobre datos de imagen, optimizando el cálculo de distancias y las transformaciones de coordenadas necesarias para ubicar los carriles respecto al vehículo.

Finalmente, **Python** actúa como el lenguaje integrador, ofreciendo una sintaxis accesible y la flexibilidad necesaria para combinar estas tecnologías en un sistema robusto y adaptable. Juntas, estas herramientas posibilitan un sistema de detección de carriles en tiempo real, contribuyendo a la seguridad y autonomía en la conducción.

3. MÉTODOS

3.1 Función Mostrar Líneas

```
def mostrar_lineas(img, lineas):
    img_lineas = np.zeros_like(img)  # Crear una imagen en negro del mismo tamaño que la original.
    if lineas is not None:  # Verificar si hay líneas detectadas.
        for linea in lineas:  # Iterar sobre cada línea detectada.
            x1, y1, x2, y2 = linea[0]  # Extraer las coordenadas de la línea.
            cv2.line(img_lineas, (x1, y1), (x2, y2), (0, 255, 0), 10)  # Dibujar la línea en verde y con grosor de 10 píxeles en `img_lineas`.
    img_combinar = cv2.addWeighted(img, 0.8, img_lineas, 1, 1)  # Combinar `img` y `img_lineas` con transparencia para superponer las líneas detectadas.
    return img_combinar  # Retornar la imagen combinada con las líneas dibujadas.
```



Esta función toma una imagen (img) y una lista de líneas (lineas). Si hay líneas detectadas, las dibuja en una imagen negra (img_lineas) y luego las superpone en la imagen original con una transparencia del 80%.

3.2 Función Calcular Desviación

```
def calcular_desviacion(img, lineas):
    ancho = img.shape[1]  # Obtener el ancho de la imagen para calcular el centro.
    if lineas is not None:  # Verificar si hay líneas detectadas.
        x_posiciones = []  # Lista para almacenar las coordenadas x de las líneas detectadas.
        for linea in lineas:  # Iterar sobre cada línea.
            x1, y1, x2, y2 = linea[0]  # Extraer las coordenadas de la línea.
            x_posiciones.extend([x1, x2])  # Añadir las coordenadas x de los dos puntos de la línea a la lista.
        x_media = np.mean(x_posiciones)  # Calcular la posición media de las coordenadas x.
        centro_imagen = ancho / 2  # Calcular el centro de la imagen.
        desviacion = x_media - centro_imagen  # Calcular la desviación desde el centro.
    else:
        desviacion = 0  # Si no hay líneas detectadas, la desviación es cero.
    return desviacion  # Retornar el valor de desviación.
```

Esta función calcula la desviación de las líneas detectadas con respecto al centro de la imagen. Si no se detectan líneas, la desviación es cero. La desviación indica si el vehículo se encuentra alineado o desviado de los carriles.

3.3 Captura de vídeo y preprocesamiento

```
# Capturar el vídeo
cap = cv2.VideoCapture('carretera4.mp4')
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    # Preprocesamiento
    gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # Convertir a escala de grises para simplificar el procesamiento.
    gris_suavizado = cv2.GaussianBlur(gris, (5, 5), 0)  # Aplicar desenfoque para reducir el ruido.
    bordes = cv2.Canny(gris_suavizado, 50, 150)  # Aplicar Canny para detectar los bordes de los carriles.
```

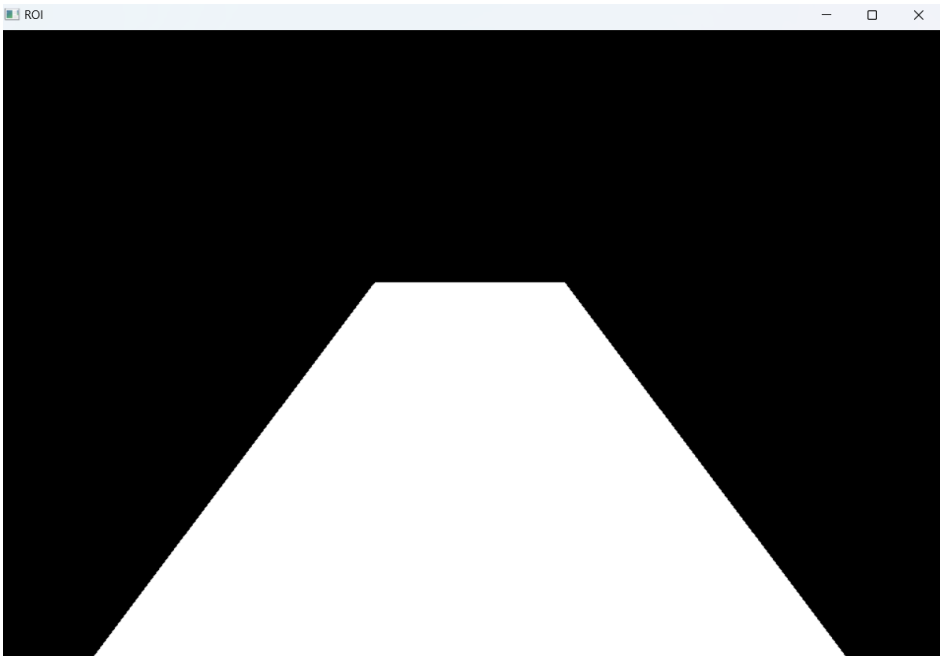
Preprocesamiento:

- Convierte cada frame del video a escala de grises para reducir la complejidad de la imagen, ya que el análisis de bordes es más sencillo en una imagen monocromática.
- Aplica un desenfoque gaussiano para suavizar la imagen y reducir el ruido, lo que mejora la detección de bordes.
- Usa el detector de bordes de Canny para resaltar los bordes de la imagen, esencialmente identificando posibles bordes de los carriles.

```
# Definir región de interés (ROI)
    altura = frame.shape[0]
    ancho = frame.shape[1]
    poligono = np.array([[  # Definir la región de interés en forma de trapecio.
        (0, altura),  # Esquina inferior izquierda
        (ancho, altura),  # Esquina inferior derecha
        (ancho, int(altura * 0.6)),  # Esquina superior derecha
        (0, int(altura * 0.6))  # Esquina superior izquierda
    ]])
    mascara = np.zeros_like(bordes)  # Crear una imagen en negro para la máscara.
    cv2.fillPoly(mascara, poligono, 255)  # Rellenar la máscara en la región definida.
    recorte = cv2.bitwise_and(bordes, mascara)  # Aplicar la máscara a la imagen de bordes.
```

Región de Interés (ROI):

- Define un trapecio que cubre la zona donde generalmente aparecen los carriles en una carretera.
- Crea una máscara negra del tamaño de la imagen, y en esa máscara rellena de blanco el área del trapecio.
- Aplica esta máscara a la imagen de bordes, para que solo se analice la región de interés donde es probable que se encuentren los carriles. Esto reduce la cantidad de ruido y mejora la precisión de la detección.




```
# Detectar líneas con la Transformada de Hough
lineas = cv2.HoughLinesP(recorte, 1, np.pi / 180, threshold=50, minLineLength=50, maxLineGap=150)
```

Transformada de Hough:

- Utiliza la Transformada de Hough Probabilística para detectar líneas rectas en la imagen.
- La transformada detecta líneas basándose en las intersecciones en el acumulador de Hough.
- Los parámetros minLineLength y maxLineGap ayudan a ajustar la detección de líneas, asegurando que solo se consideren segmentos de línea suficientemente largos y cercanos, representando de manera más confiable los carriles de la carretera.

4. INNOVACIONES

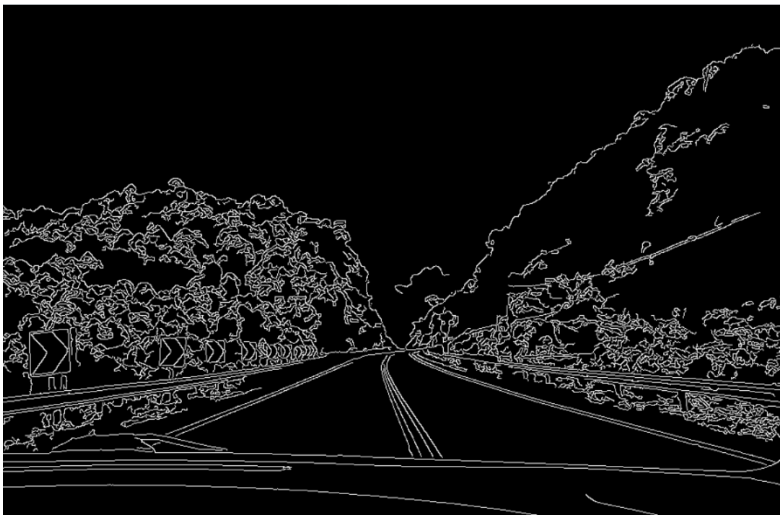
Este programa incorpora innovaciones clave en la detección de carriles y asistencia al conductor para mejorar la precisión y efectividad en carretera:

1. **Transformada de Hough Probabilística**: Permite detectar y trazar los carriles en tiempo real con precisión, identificando solo las líneas más relevantes y filtrando el ruido. Esta técnica ajusta la longitud y continuidad de las líneas, facilitando la detección de carriles, incluso en condiciones difíciles.
2. **Cálculo de Desviación del Vehículo**: Evalúa la posición del vehículo respecto al centro del carril. Mediante la distancia entre el centro de la imagen y el promedio de los carriles detectados, el programa indica si el vehículo está alineado o desviado, lo que ayuda a generar alertas visuales para mejorar la seguridad.

5. ELEMENTOS DE CLASE

Este programa utiliza conceptos clave vistos en clase, como el **preprocesamiento** y el uso del **detector de bordes de Canny**, esenciales para preparar la imagen y mejorar la detección de carriles.

1. **Preprocesamiento**: Convertimos la imagen a escala de grises para simplificar el análisis y aplicamos un filtro gaussiano para reducir el ruido. Esto facilita la detección de bordes al hacer la imagen más clara y manejable.
2. **Detector de Bordes de Canny**: Este algoritmo permite resaltar los contornos en la imagen, detectando cambios bruscos de intensidad que representan los bordes de los carriles. Este paso ayuda a definir las líneas clave que luego se procesan para identificar los carriles.



Estos conceptos básicos son fundamentales para una detección precisa y estable de carriles en el programa.

6. APLICACIÓN

Este programa de detección de carriles y cálculo de distancias es una herramienta muy útil en el contexto de los avances tecnológicos en los autos modernos. Con la demanda creciente de sistemas de ayuda a la conducción y el desarrollo de los vehículos autónomos, contar con un sistema que identifique bien los carriles y calcule la posición del coche en la carretera se ha vuelto fundamental para mejorar la seguridad y la comodidad al volante.

En los últimos años, se han implementado en los coches tecnologías como el *Lane Keeping Assist* (asistente para mantenerse en el carril) y el *Lane Departure Warning* (alerta de salida de carril). Estos sistemas ayudan a los conductores a no desviarse y reducen el riesgo de accidentes por distracción o pérdida de control. Nuestro programa sigue esta misma línea, detectando los carriles y calculando la distancia del coche con respecto a ellos, lo cual es una base importante para la seguridad en la conducción.

Además, este sistema, basado en herramientas como OpenCV y Python, resulta accesible y útil para desarrolladores e investigadores que quieren trabajar en proyectos de conducción asistida sin recurrir a tecnologías

costosas. A diferencia de los sistemas más complejos y caros que usan algunos autos modernos, este programa funciona con procesamiento de imágenes y cálculos matemáticos al alcance de casi cualquiera. Esto significa que su tecnología podría usarse en vehículos más básicos, acercando estas funciones avanzadas a más personas.

En resumen, este tipo de programas ayudan a hacer la conducción más segura y a llevar estos avances a vehículos que aún no cuentan con tecnología avanzada de ayuda al conductor. Además, contribuyen al desarrollo de la conducción autónoma de una forma accesible y escalable.