PROCESAMIENTO DIGITAL DE IMÁGENES I

Trabajo Práctico N°3

Año 2024 - Primer Semestre

Grupo 8:
Cancio José
García Julián
Herrera Francisca

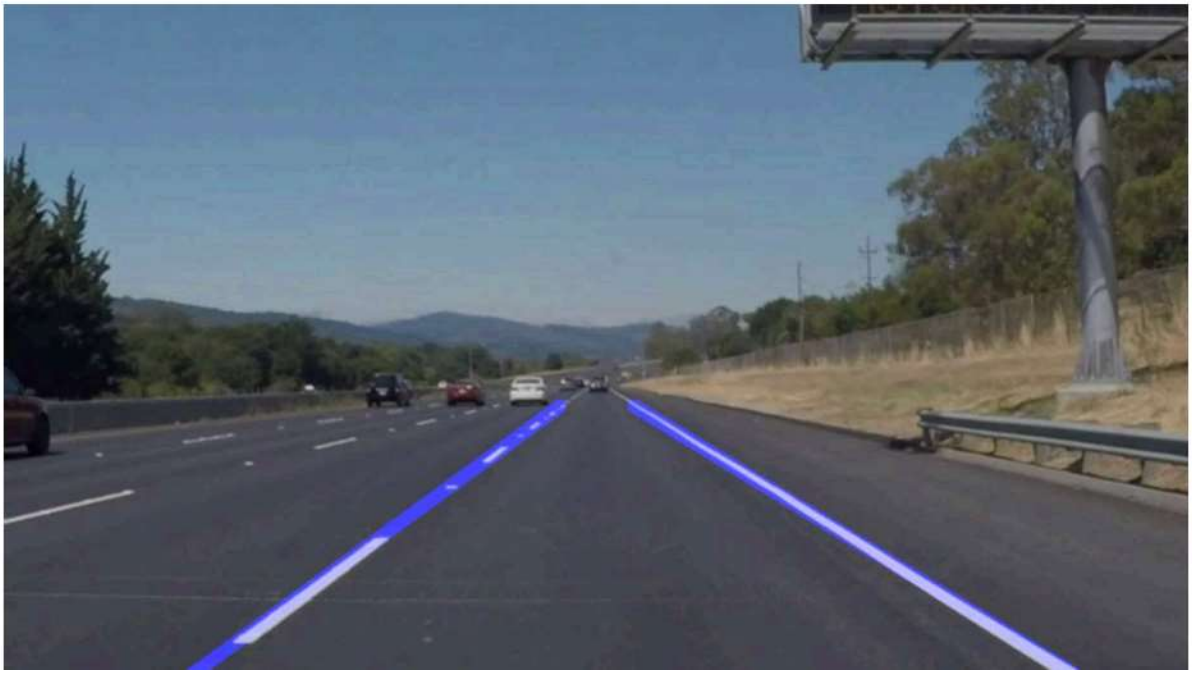
1. Introducción

Este trabajo práctico consiste en la resolución de un problema relacionado con la detección automática del carril por donde circula un auto en una ruta.

2. Objetivo

El objetivo de este informe es detallar la resolución del problema de detección y demarcación del carril de circulación de un auto en una ruta, a partir de dos videos proporcionados.
Dichos videos fueron grabados con una cámara colocada en posición fija en el interior del auto, apuntando hacia el frente mientras el mismo circulaba por la ruta, tal como se puede observar en la siguiente imagen.

Las tareas a desarrollar incluyen entonces detectar y distinguir las líneas que delimitan el carril de circulación del auto, identificarlas en color azul y generar un video de salida con el auto circulando y el carril demarcado en color, de manera similar a la siguiente imagen.



3. Desarrollo

Para resolver este problema, se han empleado diversas técnicas de procesamiento de vídeo e imágenes. Las principales herramientas y bibliotecas utilizadas incluyen OpenCV para el procesamiento de video e imágenes, Numpy para el manejo de arrays y funciones matemáticas.

Se partió analizando el código proporcionado por la cátedra (ejemplo_leer_video.py y ejemplo_grabar_video) para la manipulación de videos.

La idea inicial fue la de simplificar las imágenes (los frames) considerando que al ser fija la cámara el sector de interés de la imagen siempre ocupará el mismo lugar en la misma.
En otras palabras, interesa aplicar una imagen a cada frame como la siguiente:

Es así que la región de interés será entonces un trapecio, en donde las esquinas inferiores coincidirán con las esquinas inferiores de la imagen, mientras que las superiores resultaron luego de varias pruebas de coordenadas para lograr buenos resultados en ambos videos.

Obtenida la imagen anterior, resulta simple poder definir las líneas blancas sobre el pavimento, aplicando un umbralizado y logrando lo siguiente:

Para lograr lo anterior se genera la función mask(frame), la cual entonces recibe una imagen (frame), selecciona el ROI en función de las coordenadas preestablecidas y devuelva una imagen binaria luego de haber aplicado un umbralamiento.

La imagen resultante es el punto de partida que buscamos para poder realizar la detección de líneas de carril.
Para ello utilizamos la función cv2.HoughLinesP(imagen_mascara,
rho=1,
theta=np.pi / 180
threshold=50,
np.array([]),
minLineLength=50,
maxLineGap=200)

Dicha función devuelve una lista con las líneas detectadas en la imagen binaria (imagen_mascara), representadas por un conjunto de cuatro valores que describen los puntos de inicio y fin de la línea en el formato [x1, y1, x2, y2]. Almacenamos los resultados en la lista “lineas”.
Los valores utilizados para los parámetros surgieron de diversas iteraciones.

Obtenidas las líneas ahora es posible dibujarlas sobre el frame original a partir de la función cv2.line.

Utilizando el código proporcionado, es posible entonces aplicar lo desarrollado para cada frame del video, reproducirlo y guardarlo, para lo cual se crea la función procesar_video_con_lineas, que recibe un video y devuelve otro con las líneas azules montadas sobre el video original.

Se observan no obstante algunas oportunidades de mejora del código. Por ejemplo en uno de los videos aparecen líneas sobre el asfalto transversales a la demarcatoria, como se puede ver en la imagen siguiente.

Este tipo de líneas no son deseadas y la manera de eliminarlas es filtrando las que tengan una pendiente menor a -22.5° o mayor a 22.5° (pi/8 radianes). Las pendientes pueden ser calculadas a partir de las coordenadas de los puntos de inicio y fin de cada línea, almacenadas en la lista de líneas.

El resultado ahora será el siguiente:

Otra observación que se hace es que las líneas no siempre llegan hasta el borde inferior de la imagen. Se pretende corregir esto, y para ello se recurre nuevamente a las coordenadas de las líneas, de manera que extenderlas hasta el borde inferior en caso de ser necesario.

El resultado es el siguiente:

Ambas correcciones se aplicaron dentro de la función procesar_video_con_lineas, aplicando diversos bucles sobre las listas de líneas para filtrar y ajustar.

Como resultado final se obtiene la función mejorada que permite detectar de manera satisfactoria las líneas demarcatorias de carril, para ambos videos.

4. Conclusiones

El código desarrollado realiza eficazmente la detección de las líneas que demarcan el carril de circulación del auto. Las técnicas de procesamiento de imágenes y video utilizadas permiten una identificación, demarcación y visualización clara de los carriles por donde circulan los autos. La mayor importancia de la solución desarrollada radica en la transformada probabilística de Hough, la cual es una técnica potente y robusta para la detección de líneas.

5. Anexo

5.1. Descripción de funciones

Función **mask(frame)**

- Esta función toma un fotograma de vídeo de entrada (frame) y genera una imagen de máscara binaria.
- Crea una máscara con las mismas dimensiones que el fotograma, inicializándola con ceros.
- Define los puntos del polígono para especificar la región que se va a oscurecer.
- Llena el polígono de blanco en la máscara.

- Convierte la máscara a 3 canales (BGR) para que coincida con el espacio de color del fotograma.
- Aplica la máscara al fotograma, oscureciendo la región especificada.
- Convierte el fotograma enmascarado a escala de grises.
- Aplica la umbralización para obtener una imagen binaria.
- Devuelve la imagen de máscara binaria.

Función **procesar_video_con_lineas (archivo, salida)**

- Abre el archivo de video de entrada (archivo) usando la función OpenCV VideoCapture y almacena el objeto en la variable cap.
- Obtiene las propiedades del video (ancho y alto de los fotogramas del video y los fotogramas por segundo - fps).
- Crea un objeto VideoWriter(out) para guardar el vídeo de salida con las mismas propiedades que el video de entrada y especifica el códec de vídeo (MP4V).
- Establece los parámetros que definen la configuración para la transformación de línea de Hough (algoritmo HoughP) utilizada para detectar líneas en los fotogramas de vídeo: **rho theta y threshold.**
- Mientras el archivo de vídeo permanezca abierto (bucle while) lee los fotograma del vídeo.
- Aplica la función de máscara al frame actual por medio de la función **mask(frame)**. Esta función oscurece una región específica en el marco en función de los puntos de polígono definidos.
- Aplica la transformada de línea de Hough (algoritmo HoughP) a la imagen enmascarada para detectar líneas. Los parámetros son:
    1. **rho:** La resolución en píxeles para el parámetro de distancia (rho). Un valor más bajo da como resultado que se detecten más líneas, mientras que un valor más alto filtra las líneas menos significativas.
    2. **theta:** La resolución en radianes para el parámetro de ángulo (theta). Un valor más bajo da como resultado una mayor precisión angular, mientras que un valor más alto reduce el costo computacional.
    3. **threshold:** La cantidad mínima de votos (intersecciones) que se requiere para que una línea se considere válida. Un valor más alto excluye las líneas más débiles o con menos respaldo.
    4. **minLineLenght**: Longitud mínima de una línea que se detectará.
    5. **maxLineGap**: Espacio máximo entre dos puntos de una línea que se considerará una sola línea.
- El resultado se almacena en la variable **lines**, que contiene una lista de segmentos de línea: los puntos (x1,y1) y (x2,y2) son los puntos finales de los segmentos que fueron detectados.
- Filtrar líneas horizontales: Para cada par de puntos (x1,y1) y (x2,y2), calcula el ángulo del segmento de línea y comprueba si el ángulo cae dentro del rango

especificado (-π/8 a π/8), que se considera como horizontal. Si no es horizontal, agrega el segmento de línea a la lista **filtered_lines.**

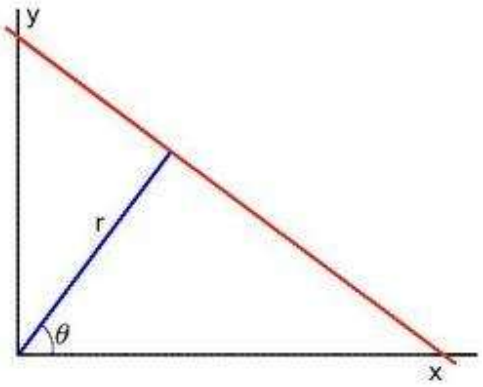
- Extender las líneas hasta el borde inferior: Para cada par de puntos (x1,y1) y (x2,y2) calcula la pendiente de la línea y redetermina el punto inferior de manera de que se ubique sobre la extensión de la línea y coincida con el borde inferior de la imagen.
- Crea una copia del frame original para evitar modificarlo (**frame_with_lines**) y dibuja la línea filtrada en la imagen copia usando el color azul y un grosor de 3 píxeles.
- Escribe el fotograma con las líneas detectadas **frame_with_lines** en el archivo de vídeo de salida (**out**).

5.2. Transformada de línea de Hough

La Transformada de línea de Hough es una transformación que se utiliza para detectar líneas rectas.
Para aplicar la Transformada, primero es conveniente realizar un preprocesamiento de detección de bordes.

¿Cómo funciona?
Una línea en el espacio de la imagen se puede expresar con dos variables. Por ejemplo:

- En el sistema de coordenadas cartesianas: Parámetros: (m,b).
- En el sistema de coordenadas polares: Parámetros: (r,θ)



Para las Transformadas de Hough, expresaremos las líneas en el sistema polar. Por lo tanto, una ecuación de línea se puede escribir como:

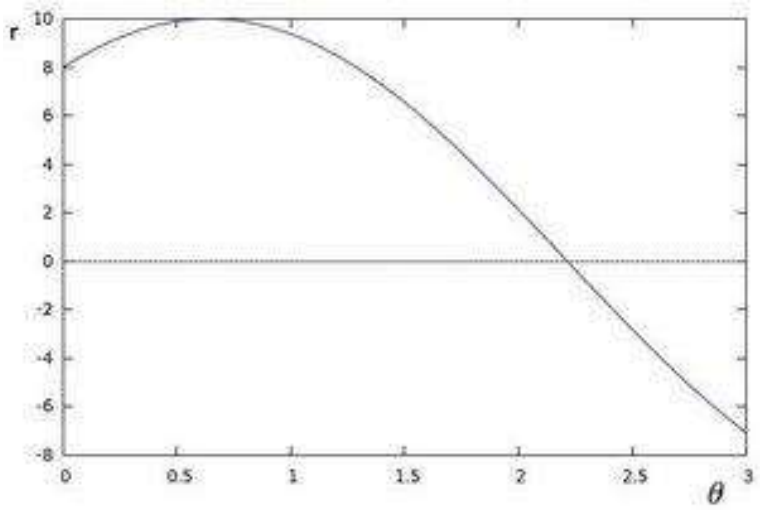
$$r = x \cos\theta + y \sin\theta$$

En general, para cada punto (x0,y0), podemos definir la familia de líneas que pasa por ese punto como:

$$r\theta = x0 \cdot \cos\theta + y0 \cdot \sin\theta$$

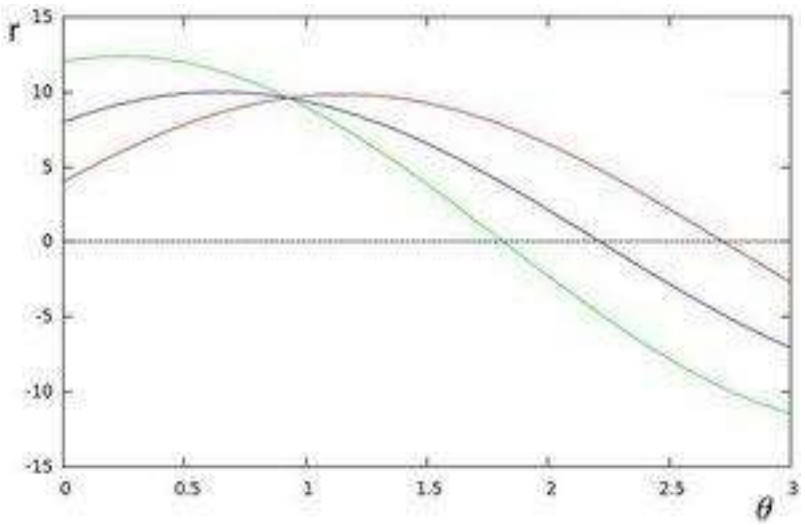
Lo que significa que cada par (rθ,θ) representa cada línea que pasa por (x0,y0).

Si para una determinada (x0,y0) graficamos la familia de rectas que la atraviesan, obtenemos una senoide. Por ejemplo, para x0 = 8 e y0 = 6 obtenemos la siguiente gráfica (en un plano θ - r):



Consideramos solo los puntos tales que r > 0 y 0 < θ < 2π.

Podemos hacer la misma operación anterior para todos los puntos de una imagen. Si las curvas de dos puntos diferentes se cortan en el plano θ - r, eso significa que ambos puntos pertenecen a una misma recta. Por ejemplo, siguiendo con el ejemplo anterior y dibujando la gráfica para dos puntos más: x1 = 4, y1 = 9 y x2 = 12, y2 = 3, obtenemos:



Las tres gráficas se cortan en un único punto (0,925,9,6), estas coordenadas son los parámetros (θ,r) o la recta en la que se encuentran (x0,y0), (x1,y1) y (x2,y2).

¿Qué significa todo lo anterior? Significa que, en general, una línea se puede detectar al encontrar el número de intersecciones entre curvas. Cuantas más curvas se intersecan, más puntos tiene la línea representada por esa intersección. En general, podemos definir un umbral del número mínimo de intersecciones necesarias para detectar una línea.
Esto es lo que hace la Transformada de Línea de Hough. Realiza un seguimiento de la intersección entre curvas de cada punto de la imagen. Si el número de intersecciones es superior a un umbral, la declara como una línea con los parámetros (θ,rθ) del punto de intersección.

Transformada de Línea de Hough Estándar y Probabilística

OpenCV implementa dos tipos de Transformadas de Línea de Hough:

a. La Transformada de Hough Estándar

Consiste básicamente en lo que acabamos de explicar en la sección anterior. Da como resultado un vector de pares (θ,rθ). En OpenCV se implementa con la función **HoughLines()**

b. Transformada lineal probabilística de Hough

Es una implementación más eficiente de la Transformada lineal de Hough. Proporciona como salida los extremos de las líneas detectadas (x0,y0,x1,y1). En OpenCV se implementa con la función **HoughLinesP()**

Fuente: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html