Tarea 1. Detección de líneas de carril en autopista

Miguel Mendoza

I. Introducción

Encontrar señales de tránsito como son las líneas divisoras de carril en las autpistas es un labor sencilla para el ser humano, sin embargo, cuando se requiere obtener dicha información de forma autónoma con un sensor de adquisición de imágenes se convierte en una tarea de grandes retos.
Se presenta el desarrollo de un proyecto para la detección de líneas de tránsito en un carril sobre una autopista. Se lleva a cabo el procesamiento sobre una imagen y también sobre videos con distintos elementos comunes en situaciones reales que influyen significativamente para obtener resultados correctos.

II. Desarrollo

El procesamiento se realizó usando librerías de OpenCV sobre la imágen mostrada en la imagen 1 y en tres videos grabados con una cámara montada al frente de un automóvil, dos de los videos presentan un ambiente sin perturbaciones como son sombras y distintas intensidades de iluminación debido a la luz del sol, y otro de los videos presenta estás perturbaciones que son elementos comunes en una situación real.

III. Algoritmo

Se enuncia el algoritmo para encontrar las líneas del carril en la imagen 1:

1: Lectura de imagen de entrada
2: Transformación a escala de grises.
3: Aplicación de filtro de Canny.
4: Aplicación de filtro Gaussiano.
5: Definición de una máscara sobre las regiones de interés de la imagen, en éste caso son los carriles.
6: Aplicación de transformada de Hough sobre las regiones de interés.
7: Definición de pendiente mínima, pendiente máxima, cont izq, cont der, pend izq, pend der, intersección izq y intersección der.
8: **for** línea inicial de Hough *hasta* línea final de Hough **do**
9: Calcular pendiente
10: **if** x1 = x2 **and** y1 = y2 **then**
11: continuar
12: **else if** pendiente minima $<$ pendiente $<$ pendiente máxima **then**
13: Calcular intercepción (valor b de la recta)
14: **if** pendiente $<$ 0 **then**
15: Contador izq + 1 $\rightarrow$ contador izq
16: Pendiente izq + pendiente $\rightarrow$ pendiente izq
17: Intersección izq + intersección $\rightarrow$ intersección izq
18: **else**
19: Contador der + 1 $\rightarrow$ contador der
20: Pendiente der + pendiente $\rightarrow$ pendiente der
21: Intersección der + intersección $\rightarrow$ intersección der
22: **end if**
23: **end if**
24: **end for**
25: Pendiente izq / contador izq$\rightarrow$ pendiente izq
26: Intersección izq / contador izq$\rightarrow$ intersección izq
27: Pendiente der / contador der$\rightarrow$ pendiente der
28: Intersección der / contador der$\rightarrow$ intersección der
29: Definir y_1 del 60 $\%$ de los renglones totales de la imagen
30: Definir y_2 del 60 $\%$ de los renglones totales de la imagen
31: (y_1 - intersección izq) / pendiente izq $\rightarrow$ x_1
32: (y_2 - intersección izq) / pendiente izq $\rightarrow$ x_2
33: Trazar línea sobre imagen original de entrada
34: (y_1 - intersección der) / pendiente der $\rightarrow$ x_1
35: (y_2 - intersección der) / pendiente der $\rightarrow$ x_2
36: Trazar línea sobre imagen original de entrada
37: Mostrar Imagen de salida con líneas detectadas

Para el caso del algoritmo usado en los videos se realizó la modificación de acuaerdo a los comandos de OpenCV [1] de leer la imagen por cada frame del video y al final se guarda el video.

IV. Resultados

Los resultados obtenidos se muestran en las imágenes 2, 3, 5, 4, 6 y 7, donde la imagen 2 es el resultado de aplicar la transformación de escala de grises y filtro gaussiano a la imagen 1. La imagen 3 muestra la salida de la imagen 2 después de haber aplicado filtro de canny. Después se aplica la máscara mostrada en la imagen 4 a la imagen 3, el procesamiento de las operaciones restantes se realizan solamente sobre ésta región obteniendo la imagen 5 y posteriormente se muestra en la imagen 6 las líneas encontradas usando Hough y por último se muestra la imagen 7 con las líneas finales detectadas sobre la imagen inicial.
La imagen 1 es una imagen fácil de procesar y obtener buenos resultados ya que no existe ruido o perturbaciones que afecten a los filtros aplicados a diferencia de los videos. El código se encuentra disponible en la liga https://github.com/MiguelMendozaG/Vision-3d/blob/master/lineas_video.py

V. Conclusion

Los resultados obtenidos fueron buenos ya que en los videos se observa un buen comportamiento de las lineas. Se encontró un problema analizando los resultados de dos videos, uno de ellos con perturbaciones y el otro en buenas condiciones, se considera que controlando la distancia de la cámara de

tal forma que los carriles entren en un rango de los pixeles entonces la máscara puede reducirse de tal forma que filtre las regiones más importantes de la imagen. Las perturbaciones de las sombras y cambios de intensidad de la luz generan que las lineas sean promediadas con ruido ocasionando que las líneas encontradas no sean pintadas correctamente. Se visualizan otros casos en los que evidentemente se tendran problemas, como por mencionar algunos: cuando un objeto entre en el campo de visión entre los carriles o cuando las líneas no estén perfectamente pintadas.

Figure 1. Escena principal

References

[1] OpenCV. Getting started with videos, 2014.

Figure 2. Filtro gaussiano y de canny

Figure 3. Bordes

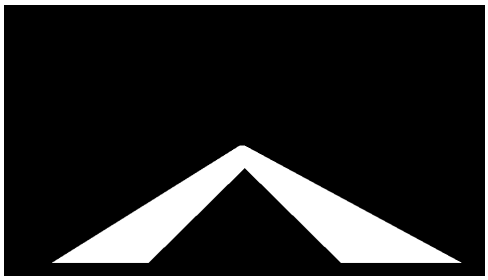

Figure 4. Máscara sobre región de interés

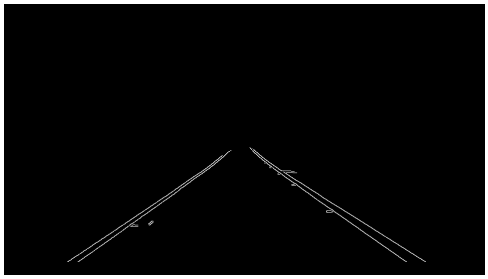

Figure 5. Bordes en región de interés

Figure 6. Líneas de Hough en región de interés

Figure 7. Líneas detectadas