TECNOLÓGICO DE MONTERREY®

Maestría en Inteligencia Artificial Aplicada

Curso: MR4010.10 Navegación autónoma

Tecnológico de Monterrey

Prof Titular y Tutor: Dr. David Antonio Torres

Prof Asistente: Maricarmen Vázquez Rojí

ALUMNO: Luis Alfonso Sabanero Esquivel

MATRICULA: A01273286

ALUMNO: Jose Mtanous

MATRICULA: A00169781

ALUMNO: Guillermo Alfonso Muñiz Hermosillo

MATRICULA: A01793101

ALUMNO: Jorge Mariles Estrada

MATRICULA: A01335663

Actividad de la Semana 03

Actividad 2.1 - Detección de carriles en video usando transformada de Hough

Mayo 2023

Enlace al video en youtube

https://www.youtube.com/watch?v=OM0y2sLBq0c

In [ ]:
```
import numpy as np
import cv2
```

El algoritmo de la Transformada de Hough consta de los siguientes pasos:

1. Conversión de la imagen a escala de grises: Primero, se convierte la imagen original a escala de grises, ya que la detección de bordes y la segmentación de la imagen son más fáciles en imágenes en escala de grises.
2. Detección de bordes: Se utiliza un operador de detección de bordes, como el operador Sobel o Canny, para detectar los bordes en la imagen en escala de grises. Los bordes son los puntos de la imagen donde la intensidad cambia abruptamente.
3. Creación del espacio de Hough: Para cada píxel de borde en la imagen, se traza una curva en el espacio de Hough. La curva representa todas las posibles líneas que podrían pasar a través del píxel de borde. El espacio de Hough es una matriz de dos dimensiones donde cada punto representa una posible línea en la imagen original.
4. Acumulación de votos: Se acumulan votos en el espacio de Hough para cada curva. Cada vez que una curva se cruza con otra curva en el espacio de Hough, se incrementa el contador de votos en esa posición del espacio de Hough. Los cruces de curvas representan las posibles líneas en la imagen original.
5. Umbralización: Se establece un umbral en el contador de votos para identificar las líneas más prominentes en la imagen original. El umbral determina el número mínimo de votos que debe tener una curva para ser considerada como una línea en la imagen original.
6. Dibujar líneas: Finalmente, se extraen los parámetros de las líneas detectadas en el espacio de Hough y se dibujan las líneas en la imagen original. Los parámetros de la línea se calculan a partir de la ecuación de la curva correspondiente en el espacio de Hough.

Es importante tener en cuenta que la Transformada de Hough es un algoritmo computacionalmente intensivo y puede requerir mucho tiempo de procesamiento para imágenes grandes o complejas.

In [ ]:
```
def region_interes(img, vertices):
    # Creamos una máscara de ceros del mismo tamaño que la imagen
    mask = np.zeros_like(img)
    # Seleccionamos el color con el que la llenamos, en este caso es negro
    match_mask_color = 255
    # Creamos el polígono con base en los vértices la máscara y el color
    cv2.fillPoly(mask, vertices, match_mask_color)
    # Seleccionamos unicamente los valores en donde la imagen la máscara vale 1
    masked_image = cv2.bitwise_and(img, mask)
    # Desplegamos la imagen enmáscarada para fines de depuración de la región de interés
    cv2.imshow('Masked', masked_image)
    # Regresamos la imagen enmascarada
    return masked_image

def linea_guia(img, lines):
    # Hacemos una copia local de la imagen recortada por la región de interés
    img = np.copy(img)
    # creamos una matriz del mismo tamaño y dimensiones de la imagen y
    # la llenamos con 0 (imagen negra)
    blank_image = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)

    # Por cada linea que nos regresa el algoritmo de Hough la
    # sobre ponemos en la matriz de la imagen en blanco
    for line in lines:
        for x1, y1, x2, y2 in line:
            # Dibujamos las lineas usando las coordenas que regresa el algoritmo de
            # Hough
            cv2.line(blank_image, (x1,y1), (x2,y2), (0, 255, 0), thickness=10)

    # Sobreponemos el frame del video con las lineas que detectamos
    img = cv2.addWeighted(img, 0.8, blank_image, 1, 0.0)
    # regresamos la imagen con las lineas sobrepuestas
    return img

def frame(img):
    # Recortamos la imagen para para mejorar la ejecución del programa
    new_width = int(img.shape[1]/1.5)
    new_height = int(img.shape[0]/1.5)

    # Calculamos los vértices de la región de interés
    # creando un rectángulo usando las nuevas medidas de la imagen
    region_of_interest_vertices = [
```

```
    (0, new_height),
    (new_width/2 - new_width/8, new_height/2),
    (new_width - new_width / 2,  new_height/2),
    (new_width, new_height)
    ]

    # Ajustamos el tamaño de la imagen
    img = cv2.resize(img, (new_width, new_height))
    # Convertimos la imagen a escala de grises para facilitar la
    # detección de bordes
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Usamos el algoritmo de Canny para detectar los bordes
    edges = cv2.Canny(gray,50,150,apertureSize = 3)
    # Obtemos una imagen recortada del frame usando los vértices de
    # la región de interés
    cropped_image = region_interes(edges,
                np.array([region_of_interest_vertices], np.int32),)
    # Aplicamos el algoritmo de Hough probabilístico para obtener las lineas
    lines = cv2.HoughLinesP(cropped_image,
                            rho=1,
                            theta=np.pi/180,
                            threshold=40,
                            lines=np.array([]),
                            minLineLength=30,
                            maxLineGap=5)
    # Sobreponemos la imagen del frame con las lineas
    # detectadas por el algoritmo
    image_with_lines = linea_guia(img, lines)
    # Regresamos el frame con la lineas sobrepuestas
    return image_with_lines

# Cargamos el video
cap = cv2.VideoCapture('test_video.mp4')
# Iteramos cuadro por cuadro
while cap.isOpened():
    ret, img= cap.read()

    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break

    # Encontramos la lineas y las sobreponemos sobre el cuadro de video
    img_lines = frame(img)

    # desplegamos el video con las lineas sobrepuestas
    cv2.imshow('frame', img_lines)
    if cv2.waitKey(1) == ord('q'):
        break


cap.release()
cv2.destroyAllWindows()
```

```
Can't receive frame (stream end?). Exiting ...
```