# **Maestría en Inteligencia Artificial Aplicada**

## **Curso: Navegación autónoma**

### Tecnológico de Monterrey

### Dr. David Antonio Torres

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

**Nombres y matrículas de los integrantes del equipo:**

* Julio Cesar Lynn Jimenez
* Francisco Javier Parga García A01794380
* Carlos Roberto Torres Ferguson
* Fernando Sebastian Sanchez Cardona

# Importar librerias necesarias

import cv2  
import numpy as np

class LaneDetector:  
 # Constructor de la clase LaneDetector  
 def \_\_init\_\_(self):  
 """  
 Clase para procesar video y detectar líneas  
   
 Métodos:  
 -region\_of\_interest: Método para crear la región de interés para detectar líneas  
 -draw\_lines: Método para dibujar las líenas detectadas en la región de interés  
 -process\_frame: Método para procesar cada fotograma  
 -process\_video: Método para procesar el video  
 """  
 pass  
  
 # Método para aplicar una máscara en la región de interés de la imagen  
 def region\_of\_interest(self, img, vertices):  
 """  
 Método para crear la región de interés para detectar líneas  
   
 Inputs:  
 -img: Fotograma de imagen  
 -vertices: Numpy array con los tuples de 4 vértices de enteros [[(H1,W1), (H2,W2), (H3,W3), (H4,W4)]]  
   
 Outputs:  
 -masked\_image: Imagen con la máscara en la región de interés  
 """  
   
 # Crear una máscara negra del mismo tamaño que la imagen  
 mask = np.zeros\_like(img)  
 # Definir el color de la máscara que coincide con la región de interés  
 match\_mask\_color = 255  
 # Rellenar la máscara con el color en la región de interés  
 cv2.fillPoly(mask, vertices, match\_mask\_color)  
 # Aplicar la máscara a la imagen  
 masked\_image = cv2.bitwise\_and(img, mask)  
 # Devolver la imagen con la máscara aplicada  
 return masked\_image  
  
 # Método para dibujar líneas en la imagen  
 def draw\_lines(self, img, lines):  
 """  
 Método para dibujar las líenas detectadas en la región de interés  
   
 Inputs:  
 -img: Fotograma de imagen  
 -lines: arreglo con la transformación de Hough  
   
 Outputs:  
 -img: Imagen con líneas dibujadas  
 """  
   
 # Crear una copia de la imagen  
 img = np.copy(img)  
 # Crear una imagen negra del mismo tamaño que la imagen original  
 line\_img = np.zeros((img.shape[0], img.shape[1], 3), dtype=np.uint8)  
 # Recorrer todas las líneas  
 for line in lines:  
 # Recorrer todos los puntos de la línea  
 for x1, y1, x2, y2 in line:  
 # Dibujar la línea en la imagen de líneas  
 cv2.line(line\_img, (x1, y1), (x2, y2), (0, 255, 0), 3)  
 # Combinar la imagen original con la imagen de líneas  
 img = cv2.addWeighted(img, 0.8, line\_img, 1, 0)  
 # Devolver la imagen con las líneas dibujadas  
 return img  
  
 # Método para procesar un fotograma (frame) de video y detectar líneas  
 def process\_frame(self, img):  
 """  
 Método para procesar cada fotograma  
   
 Inputs:  
 -img: Fotograma de imagen  
   
 Outputs:  
 -img\_with\_lines: Imagen con líneas dibujadas  
 """  
   
 # Convertir la imagen a escala de grises  
 gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 # Aplicar desenfoque Gaussiano a la imagen en escala de grises  
 img\_blur = cv2.GaussianBlur(gray\_img, (3, 3), 0, 0)  
 # Detectar bordes usando el algoritmo de Canny  
 img\_canny = cv2.Canny(img\_blur, 40, 120)  
 # Obtener las dimensiones de la imagen  
 height, width = img.shape[:2]  
 # Definir los vértices de la región de interés  
 vertices = np.array([[(width\*0.1, height), (width\*0.4, height\*0.6), (width\*0.6, height\*0.6), (width\*0.9, height)]], dtype=np.int32)  
 # Aplicar la región de interés a la imagen con bordes detectados  
 img\_roi = self.region\_of\_interest(img\_canny, vertices)  
  
 # Definir parámetros para la transformada de Hough  
 rho = 2  
 theta = np.pi / 180  
 threshold = 17  
 min\_line\_len = 70  
 max\_line\_gap = 110  
 # Aplicar la transformada de Hough para detectar líneas  
 lines = cv2.HoughLinesP(img\_roi, rho, theta, threshold, np.array([]), minLineLength=min\_line\_len, maxLineGap=max\_line\_gap)  
  
 if lines is not None:  
 img\_with\_lines = self.draw\_lines(img, lines)  
 else:  
 img\_with\_lines = img  
  
 return img\_with\_lines  
  
 # Método para procesar un video completo y detectar líneas en cada fotograma (frame)  
 def process\_video(self, input\_video\_path, output\_video\_path):  
 """  
 Método para procesar el video  
   
 Inputs:  
 -input\_video\_path: Archivo de video de entrada  
   
 Outputs:  
 -output\_video\_path: Archivo de video de salida con las líneas detectadas y resaltadas en cada fotograma  
 """  
   
 # Leer el video de entrada  
 cap = cv2.VideoCapture(input\_video\_path)  
 # Obtener las dimensiones y la tasa de fotogramas del video  
 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))  
  
 # Definir el códec y crear el objeto VideoWriter para el video de salida  
 fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')  
 out = cv2.VideoWriter(output\_video\_path, fourcc, fps, (width, height))  
  
 # Leer y procesar cada fotograma (frame) del video de entrada  
 while cap.isOpened():  
 ret, frame = cap.read()  
 if ret:  
 # Procesar el fotograma actual y detectar líneas  
 result\_frame = self.process\_frame(frame)  
 # Escribir el fotograma procesado en el video de salida  
 out.write(result\_frame)  
 # Mostrar el fotograma procesado en una ventana  
 #cv2.imshow('Processed Frame', result\_frame)  
  
 # Si se presiona la tecla 'q', interrumpir el procesamiento del video  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
 # Si no hay más fotogramas para leer, terminar el procesamiento  
 else:  
 break  
  
 # Liberar recursos y cerrar ventanas  
 cap.release()  
 out.release()  
 cv2.destroyAllWindows()

input\_video\_path = 'input/test2\_low-res.mp4'  
output\_video\_path = 'output/test2\_low-res\_detect.mp4'  
lane\_detector = LaneDetector()  
lane\_detector.process\_video(input\_video\_path, output\_video\_path)