Universidad de Sevilla

Escuela Politécnica Superior de Sevilla

Trabajo Fin de Grado en Ingeniería Electrónica Industrial

Aforador de vehículos en carretera mediante *OpenCV* sobre plataforma local.

Autor: Javier Sánchez-Barbudo Martín

Tutor: Antonio García Delgado

Junio de 2022

Aforador de vehículos en carretera mediante *OpenCV* sobre plataforma local.

ANEXO IV: ALGORITMO COMENTADO

Departamento de Tecnología Electrónica

Escuela Politécnica Superior de Sevilla

Universidad de Sevilla

Junio de 2022

```
# Importación librerías:
import cv2   # Importa la librería OpenCV al completo.
import time  # Importa la librería Time al completo.
import numpy # Importa la librería Numpy al completo. Generalmente se emplea numpy as
np.
import math  # Importa la librería Time al completo.
# ------------------------------------------------------------------------------
fotogramas = ContadorPPSI1 = ContadorPPSI2 = ContadorPPSD1 = ContadorPPSD2 =
prev_frame_time = new_frame_time = med_fps = 0 # Definición de variables globales
empleadas en el aforador de vehículos. Inicialización a 0.
contador_vehi_izq = contador_vehi_der = 1       # Definición de variables globales
empleadas en el aforador de vehículos. Inicialización a 1.
seguimiento_izq_on = seguimiento_der_on = SI1_ATV_PRE = SI2_ATV_PRE = SD1_ATV_PRE =
SD2_ATV_PRE = False                             # Definición de variables globales
empleadas en el aforador de vehículos. Inicialización a False.
box_izq_1_anterior = box_izq_2_anterior = box_der_1_anterior = box_der_2_anterior =
CISI1 = CISI2 = CISD1 = CISD2 = [0, 0, 0, 0]      # Definición de variables globales
empleadas en el aforador de vehículos. Lista inicializada a 0.
SI1 = SI2 = SD1 = SD2 = PPSI1 = PPSI2 = PPSD1 = PPSD2 = PTSI1 = PTSI2 = PTSD1 = PTSD2
= SI1_OK = SI2_OK = SD1_OK = SD2_OK = ACT_SI1 = ACT_SI2 = ACT_SD1 = ACT_SD2 = False
                                                # Definición de variables globales
empleadas en el aforador de vehículos. Inicialización a False.
ISI1 = ISI2 = ISD1 = ISD2 = True                # Definición de variables globales
empleadas en el aforador de vehículos. Inicialización a True.
count_fps = []                                  #  Definición  de  variable  global
empleada en el aforador de vehículos. Inicialización sin tamaño definido.
# ------------------------------------------------------------------------------
# Captura del vídeo:
video_capture = cv2.VideoCapture('TFG_1.mp4')
#video_capture = cv2.VideoCapture('TFG_2.mp4')
#video_capture = cv2.VideoCapture('TFG_3.mp4')
fps_original = video_capture.get(cv2.CAP_PROP_FPS) # Obtención de FPS originales del
vídeo tratado.
# ------------------------------------------------------------------------------
# Función de detección:
def detect(subtraction_left, subtraction_right):
    global seguimiento_izq_on, seguimiento_der_on # Definición de variables globales
empleadas en la función.
    global ACT_SI1, ACT_SI2, ACT_SD1, ACT_SD2, SI1, SI2, SD1, SD2, ISI1, ISI2, ISD1,
ISD2 # Definición de variables globales empleadas en la función.
     global  CISI1,  CISI2,  CISD1,  CISD2,  SI1_ATV_PRE,  SI2_ATV_PRE,  SD1_ATV_PRE,
SD2_ATV_PRE # Definición de variables globales empleadas en la función.
# Reconocimiento de contornos:
      contours_left,  _  =  cv2.findContours(subtraction_left,  cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
      contours_right,  _  =  cv2.findContours(subtraction_right,  cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
    deteccion_izq = [] # Definición de variables locales empleadas en la función. Inicialización sin tamaño definido.
    deteccion_der = [] # Definición de variables locales empleadas en la función. Inicialización sin tamaño definido.
   pos_izq_SI1 = pos_izq_SI2 = pos_der_SD1 = pos_der_SD2 = 5 # Definición de variables locales empleadas en la función. Inicialización a 5.

    for contour_l in contours_left: # Bucle para recorrer el conjunto de contornos detectados.
        left_area = cv2.contourArea(contour_l) # Se obtiene el área del contorno.
       if left_area > 6000: # Si tal área es mayor de un determinado rango, el contorno se considera.
              x_i, y_i, w_i, h_i = cv2.boundingRect(contour_l) # Se obtienen las coordenadas en formato de array del contorno considerado.
             deteccion_izq.append([x_i, y_i, w_i, h_i]) # Se cargan las coordenadas anteriores en una lista.
    if len(deteccion_izq) == 1 and (deteccion_izq[0][0] < 60 or deteccion_izq[0][1] > 300) # Si hay una única detección y está fuera de rango...
           deteccion_izq.clear() # Borra la detección.
    elif len(deteccion_izq) == 2: # Si hay dos detecciones...
         if deteccion_izq[0][0] < 60 or deteccion_izq[0][1] > 300: # Si la primera detección está fuera de rango...
           del deteccion_izq[0] # Borra tal detección.
           if deteccion_izq[0][0] < 60 or deteccion_izq[0][1] > 300: # Si la segunda detección está por debajo de un determiando margen...
               del deteccion_izq[0] # Borra tal detección.
        elif deteccion_izq[1][0] < 60 or deteccion_izq[1][1] > 300: # Si la segunda detección está fuera de rango...
           del deteccion_izq[1] # Borra tal detección.
       if len(deteccion_izq) == 2: # Si ambas detecciones son adecuadas...
                vect_distance_1_2, _ = distance (deteccion_izq, deteccion_der, box_izq_1_anterior, box_izq_2_anterior, box_der_1_anterior, box_der_2_anterior) # Llamada a la función distance para la toma de decisiones.
           if vect_distance_1_2[4] < 60: # Si la distancia entre ambas detecciones es menor de un determinado rango, se están aplicando ambas detecciones al mismo vehículo. # Comparamos ambas cajas con las CISI1 y CISI2, nos quedamos con la que la distancia sea menor.
               if ((SI1 and vect_distance_1_2[6] >= vect_distance_1_2[7]) or (SI2 and vect_distance_1_2[8] >= vect_distance_1_2[9])): # Si la distancia existente entre la detección 0 y la boundingbox del SI1 o SI2 es mayor o igual que la distancia existente entre la detección 1 y la boundingbox del SI1 o SI2.
                   del deteccion_izq[0] # Borra la primera detección.
               elif ((SI1 and vect_distance_1_2[6] < vect_distance_1_2[7]) or (SI2 and vect_distance_1_2[8] < vect_distance_1_2[9])): # Si la distancia existente entre la detección 0 y la boundingbox del SI1 o SI2 es menor que la distancia existente entre la detección 1 y la boundingbox del SI1 o SI2.
                   del deteccion_izq[1] # Borra la segunda detección.
    elif len(deteccion_izq) > 2 and SI1 and SI2: # Si hay más de dos detecciones y ambos seguidores están activos...
```

```
                _,   vect_distance_3    =   distance   (deteccion_izq,   deteccion_der,
box_izq_1_anterior,   box_izq_2_anterior,   box_der_1_anterior,   box_der_2_anterior)   #
Llamada a la función distance para la toma de decisiones.
       distance_SI1 = vect_distance_3[0:2] # División de la lista devuelta. distance_SI1
es una nueva lista que contiene los tres primeros elementos.
       distance_SI2 = vect_distance_3[3:5] # División de la lista devuelta. distance_SI2
es una nueva lista que contiene los elementos 3,4 y 5 de vect_distance_3.
        pos_izq_SI1 = distance_SI1.index(min(distance_SI1)) # pos_izq_SI1 almacena el
índice donde se localiza la mínima distancia almacenada en distance_SI1.
        deteccion_izq[0] = deteccion_izq[pos_izq_SI1] # deteccion_izq[0] almacena las
primeras coordenadas adecuadas (distancia mínima) asociadas a la primera detección.
        pos_izq_SI2 = distance_SI2.index(min(distance_SI2)) # pos_izq_SI2 almacena el
índice donde se localiza la mínima distancia almacenada en distance_SI2.
        deteccion_izq[1] = deteccion_izq[pos_izq_SI2] # deteccion_izq[1] almacena las
segundas coordenadas adecuadas (distancia mínima) asociadas a la segunda detección.
         if pos_izq_SI1 == 5 or pos_izq_SI2 == 5: # Si no se ha almacenado ningún
índice...
             del deteccion_izq # Borrado de todas las detecciones.
        else: # Si se ha almacenado algún índice...
           del deteccion_izq[2:] # Borra todas las detecciones excepto las dos primeras.
                 vect_distance_1_2,  _  =  distance  (deteccion_izq,  deteccion_der,
box_izq_1_anterior,  box_izq_2_anterior,  box_der_1_anterior,  box_der_2_anterior)  #
Llamada a la función distance para la toma de decisiones.
            if vect_distance_1_2[4] == 0: # Si la distancia entre ambas detecciones es
nula...
                del deteccion_izq[1] # Borra la primera detección.

    if len(deteccion_izq) > 0: # Si hay alguna detección que tratar...
        seguimiento_izq_on = True # Activa el flag de seguimiento asociado al carril
izquierdo.
              vect_distance_1_2,   _   =   distance   (deteccion_izq,   deteccion_der,
box_izq_1_anterior,  box_izq_2_anterior,  box_der_1_anterior,  box_der_2_anterior)  #
Llamada a la función distance para la toma de decisiones.
        if (len(deteccion_izq) == 2): # Si existen 2 detecciones...
             if SI1 and not SI2_ATV_PRE: # Si el SI1 está activo y el SI2 no estaba
activo previamente...
                 if vect_distance_1_2[6] >= vect_distance_1_2[7]: # Si la distancia
existente entre la detección 0 y la boundingbox del SI1 es mayor o igual que la distancia
existente entre la detección 1 y la boundingbox del SI1...
                     CISI1 = deteccion_izq[1] # Las coordenadas iniciales del SI1 son
asignadas a las segundas coordenadas detectadas.
                     CISI2 = deteccion_izq[0] # Las coordenadas iniciales del SI2 son
asignadas a las primeras coordenadas detectadas.
                 else: # Si la distancia existente entre la detección 0 y la boundingbox
del SI1 es menor que la distancia existente entre la detección 1 y la boundingbox del
SI1...
                     CISI1 = deteccion_izq[0] # Las coordenadas iniciales del SI1 son
asignadas a las primeras coordenadas detectadas.
```

```
                    CISI2 = deteccion_izq[1] # Las coordenadas iniciales del SI2 son asignadas a las segundas coordenadas detectadas.
                SI1_ATV_PRE = True # Activa el flag para indicar que el SI1 está activo previamente al SI2.
            if not SI1 or SI2_ATV_PRE: # Si no está activo el SI1 o si está activo el flag para indicar que el SI2 está activo previamente al SI1...
                if vect_distance_1_2[8] >= vect_distance_1_2[9]: # Si la distancia existente entre la detección 0 y la boundingbox del SI2 es mayor o igual que la distancia existente entre la detección 1 y la boundingbox del SI2...
                    CISI1 = deteccion_izq[0] # Las coordenadas iniciales del SI1 son asignadas a las primeras coordenadas detectadas.
                    CISI2 = deteccion_izq[1] # Las coordenadas iniciales del SI2 son asignadas a las segundas coordenadas detectadas.
                else: # Si la distancia existente entre la detección 0 y la boundingbox del SI2 es menor que la distancia existente entre la detección 1 y la boundingbox del SI2...
                    CISI1 = deteccion_izq[1] # Las coordenadas iniciales del SI1 son asignadas a las segundas coordenadas detectadas.
                    CISI2 = deteccion_izq[0] # Las coordenadas iniciales del SI2 son asignadas a las primeras coordenadas detectadas.
                SI2_ATV_PRE = True # Activa el flag para indicar que el SI2 está activo previamente al SI1.
            if SI1_ATV_PRE or SI2_ATV_PRE: # Si cualquier seguidor izquierdo estaba activo previamente al otro...
                SI1 = ACT_SI1 = ISI1 = SI2 = ACT_SI2 = ISI2 = True # Activa todos los flags relacionados con los seguidores del carril izquierdo.
        elif (not SI1_ATV_PRE and not SI2_ATV_PRE) or PPSI1: # Si no está activo ningún flag asociado a la activación previa de ningún seguidor o se ha producido una PP en el SI1...
            if not SI2 and CISI1[0] == 0 and CISI1[1] == 0: # Si no está activo el SI2 y no hay coordenadas iniciales en el SI1 (tanto en la coordenada x como en la y)...
                CISI1 = deteccion_izq[0] # Las coordenadas iniciales del SI1 son asignadas a las coordenadas detectadas.
                SI1 = ACT_SI1 = ISI1 = True # Activa todos los flags relacionados con el SI1.
            else: # Si está activo el SI2 y hay coordenadas iniciales en el SI1 (en la coordenada x o en la coordenada y)...
                if vect_distance_1_2[1] == 0 and vect_distance_1_2[0] <= 100: # Si la distancia existente entre la detección izquierda y la boundingbox del SI2 es nula y la distancia existente entre la detección izquierda y la boundingbox del SI1 está por debajo o es igual a un rango...
                    CISI1 = deteccion_izq[0] # Las coordenadas iniciales del SI1 son asignadas a las coordenadas detectadas.
                    SI1 = ACT_SI1 = ISI1 = True # Activa todos los flags relacionados con el SI1.
                else: # Si la distancia existente entre la detección izquierda y la boundingbox del SI2 no es nula o la distancia existente entre la detección izquierda y la boundingbox del SI1 no está por debajo o no es igual a un rango...
```

```
                    CISI1 = box_izq_1_anterior # Las coordenadas iniciales del SI1 son asignadas a las últimas coordenadas proporcionadas por el SI1.
                    CISI2 = deteccion_izq[0] # Las coordenadas iniciales del SI2 son asignadas a las coordenadas detectadas.
                    SI1 = ACT_SI1 = SI2 = ACT_SI2 = ISI1 = ISI2 = True # Activa todos los flags relacionados con el SI1 y el SI2.
                    SI1_ATV_PRE = True # Activa el flag para indicar que el SI1 está activo previamente al SI2.
        elif (SI2 or PPSI2 or SI2_ATV_PRE): # Si está activo el SI2 o se ha producido una PP en el el SI2 o el flag de activación previa del SI2 está activo ...
            if not SI1 and CISI2[0] == 0 and CISI2[1] == 0: # Si no está activo el SI1 y no hay coordenadas iniciales en el SI2 (tanto en la coordenada x como en la y)...
                CISI2 = deteccion_izq[0] # Las coordenadas iniciales del SI2 son asignadas a las coordenadas detectadas.
                SI2 = ACT_SI2 = ISI2 = True # Activa todos los flags relacionados con el SI2.
            else: # Si no está activo el SI2 y no se ha producido una PP en el el SI2 y el flag de activación previa del SI2 no está activo ...
                if vect_distance_1_2[0] == 0 and vect_distance_1_2[1] <= 100: # Si la distancia existente entre la detección izquierda y la boundingbox del SI1 es nula y la distancia existente entre la detección izquierda y la boundingbox del SI2 está por debajo o es igual a un rango...
                    CISI2 = deteccion_izq[0] # Las coordenadas iniciales del SI2 son asignadas a las coordenadas detectadas.
                    SI2 = ACT_SI2 = ISI2 = True # Activa todos los flags relacionados con el SI2.
                    SI2_ATV_PRE = True # Activa el flag para indicar que el SI2 está activo previamente al SI1.
                else: # Si la distancia existente entre la detección izquierda y la boundingbox del SI1 no es nula o la distancia existente entre la detección izquierda y la boundingbox del SI2 no está por debajo o no es igual a un rango...
                    if vect_distance_1_2[0] >= vect_distance_1_2[1]: # Si la distancia existente entre la detección izquierda y la boundingbox del SI1 es mayor o igual a la distancia existente entre la detección izquierda y la boundingbox del SI2...
                        CISI1 = box_izq_1_anterior # Las coordenadas iniciales del SI1 son asignadas a las últimas coordenadas proporcionadas por el SI1.
                        CISI2 = deteccion_izq[0] # Las coordenadas iniciales del SI2 son asignadas a las coordenadas detectadas.
                    else: # Si la distancia existente entre la detección izquierda y la boundingbox del SI1 es menor a la distancia existente entre la detección izquierda y la boundingbox del SI2...
                        CISI1 = deteccion_izq[0] # Las coordenadas iniciales del SI1 son asignadas a las coordenadas detectadas.
                        CISI2 = box_izq_2_anterior # Las coordenadas iniciales del SI2 son asignadas a las últimas coordenadas proporcionadas por el SI2.
                    SI1 = ACT_SI1 = SI2 = ACT_SI2 = ISI1 = ISI2 = True # Activa todos los flags relacionados con los seguidores del carril izquierdo.
```

```
for contour_r in contours_right: # Bucle para recorrer el conjunto de contornos detectados.
        right_area = cv2.contourArea(contour_r) # Se obtiene el área del contorno.
       if right_area > 6000: # Si tal área es mayor de un determinado rango, el contorno se considera.
               x_d, y_d, w_d, h_d = cv2.boundingRect(contour_r) # Se obtienen las coordenadas en formato de array del contorno considerado.
             deteccion_der.append([x_d, y_d, w_d, h_d]) # Se cargan las coordenadas anteriores en una lista.
    if len(deteccion_der) == 1 and (deteccion_der[0][0] < 30 or deteccion_der[0][1] < 30): # Si hay una única detección y está fuera de rango...
            deteccion_der.clear() # Borra la detección.
    elif len(deteccion_der) == 2: # Si hay dos detecciones...
         if deteccion_der[0][0] < 30 or deteccion_der[0][1] < 30: # Si la primera detección está fuera de rango...
            del deteccion_der[0] # Borra tal detección.
          if deteccion_der[0][0] < 30 or deteccion_der[0][1] < 30: # Si la segunda detección está fuera de rango...
               del deteccion_der[0] # Borra tal detección.
         elif deteccion_der[1][0] < 30 or deteccion_der[1][1] < 30: # Si la segunda detección está fuera de rango...
            del deteccion_der[1] # Borra tal detección.
        if len(deteccion_der) == 2: # Si ambas detecciones son adecuadas...
                vect_distance_1_2, _ = distance (deteccion_izq, deteccion_der, box_izq_1_anterior, box_izq_2_anterior, box_der_1_anterior, box_der_2_anterior) # Llamada a la función distance para la toma de decisiones.
            if vect_distance_1_2[5] < 60: # Si ocurre esto, está aplicando ambas cajas al mismo vehículo.
#Comparamos ambas cajas con las CISD1 y CISD2, nos quedamos con la que la distancia sea menor.
                if ((SD1 and vect_distance_1_2[10] >= vect_distance_1_2[11]) or (SD2 and vect_distance_1_2[12] >= vect_distance_1_2[13])): # Si SD1 activo y la distancia entre la detección0 y la caja del SD1 es mayor que la distancia entre la detección1 y la caja del SD1...
                    del deteccion_der[0] # Borra la primera detección.
                elif ((SD1 and vect_distance_1_2[10] < vect_distance_1_2[11]) or (SD2 and vect_distance_1_2[12] < vect_distance_1_2[13])): # Si SD1 activo y la distancia entre la detección0 y la caja del SD1 es menor que la distancia entre la detección1 y la caja del SD1...
                    del deteccion_der[1] # Borra la segunda detección.
    elif len(deteccion_der) > 2 and SD1 and SD2: # Si hay más de dos detecciones y ambas detectores están activos.
                _, vect_distance_3 = distance (deteccion_izq, deteccion_der, box_izq_1_anterior, box_izq_2_anterior, box_der_1_anterior, box_der_2_anterior) # Llamada a la función distance para la toma de decisiones.
       distance_SD1 = vect_distance_3[6:9] # División de la lista devuelta. distance_SD1 es una nueva lista que contiene los elementos 6,7 y 8 de vect_distance_3.
```

```
            distance_SD2 = vect_distance_3[9:12] # División de la lista devuelta. distance_SD2 es una nueva lista que contiene los elementos 9,10 y 11 de vect_distance_3.
        pos_der_SD1 = distance_SD1.index(min(distance_SD1)) # pos_der_SD1 almacena el índice donde se localiza la mínima distancia almacenada en distance_SD1.
         deteccion_der[0] = deteccion_der[pos_der_SD1 # deteccion_der[0] almacena las primeras coordenadas adecuadas (distancia mínima) asociadas a la primera detección.
        pos_der_SD2 = distance_SD2.index(min(distance_SD2)) # pos_der_SD2 almacena el índice donde se localiza la mínima distancia almacenada en distance_SD2.
        deteccion_der[1] = deteccion_der[pos_der_SD2] # deteccion_der[1] almacena las segundas coordenadas adecuadas (distancia mínima) asociadas a la segunda detección.
         if pos_der_SD1 == 5 or pos_der_SD2 == 5: # Si no se ha almacenado ningún índice...
             del deteccion_der # Borrado de todas las detecciones.
        else: # Si se ha almacenado algún índice...
           del deteccion_der[2:] # Borra todas las detecciones excepto las dos primeras.
                vect_distance_1_2, _ = distance (deteccion_izq, deteccion_der, box_izq_1_anterior, box_izq_2_anterior, box_der_1_anterior, box_der_2_anterior) # Llamada a la función distance para la toma de decisiones.
            if vect_distance_1_2[5] == 0: # Si la distancia entre ambas detecciones es nula...
                del deteccion_der[1] # Borra la primera detección.

    if len(deteccion_der) > 0: # Si hay alguna detección que tratar...
         seguimiento_der_on = True # Activa el flag de seguimiento asociado al carril derecho.
               vect_distance_1_2, _ = distance (deteccion_izq, deteccion_der, box_izq_1_anterior, box_izq_2_anterior, box_der_1_anterior, box_der_2_anterior) # Llamada a la función distance para la toma de decisiones.
        if (len(deteccion_der) == 2): # Si existen 2 detecciones...
             if SD1 and not SD2_ATV_PRE: # Si el SD1 está activo y el SD2 no estaba activo previamente...
                 if vect_distance_1_2[10] >= vect_distance_1_2[11]: # Si la distancia existente entre la detección 0 y la boundingbox del SD1 es mayor o igual que la distancia existente entre la detección 1 y la boundingbox del SD1...
                     CISD1 = deteccion_der[1] # Las coordenadas iniciales del SD1 son asignadas a las segundas coordenadas detectadas.
                     CISD2 = deteccion_der[0] # Las coordenadas iniciales del SD2 son asignadas a las primeras coordenadas detectadas.
                 else: # Si la distancia existente entre la detección 0 y la boundingbox del SD1 es menor que la distancia existente entre la detección 1 y la boundingbox del SD1...
                     CISD1 = deteccion_der[0] # Las coordenadas iniciales del SD1 son asignadas a las primeras coordenadas detectadas.
                     CISD2 = deteccion_der[1] # Las coordenadas iniciales del SD2 son asignadas a las segundas coordenadas detectadas.
                 SD1_ATV_PRE = True # Activa el flag para indicar que el SD1 está activo previamente al SD2.
             if not SD1 or SD2_ATV_PRE: # Si no está activo el SD1 o si está activo el flag para indicar que el SD2 está activo previamente al SD1...
```

```
                if vect_distance_1_2[12] >= vect_distance_1_2[13]: # Si la distancia existente entre la detección 0 y la boundingbox del SD2 es mayor o igual que la distancia existente entre la detección 1 y la boundingbox del SD2...
                    CISD1 = deteccion_der[0] # Las coordenadas iniciales del SD1 son asignadas a las primeras coordenadas detectadas.
                    CISD2 = deteccion_der[1] # Las coordenadas iniciales del SD2 son asignadas a las segundas coordenadas detectadas.
                else: # Si la distancia existente entre la detección 0 y la boundingbox del SD2 es menor que la distancia existente entre la detección 1 y la boundingbox del SD2...
                    CISD1 = deteccion_der[1] # Las coordenadas iniciales del SD1 son asignadas a las segundas coordenadas detectadas.
                    CISD2 = deteccion_der[0] # Las coordenadas iniciales del SI2 son asignadas a las primeras coordenadas detectadas.
                SD2_ATV_PRE = True # Activa el flag para indicar que el SD2 está activo previamente al SD1.
            if SD1_ATV_PRE or SD2_ATV_PRE: # Si cualquier seguidor derecho estaba activo previamente al otro...
                SD1 = ACT_SD1 = ISD1 = SD2 = ACT_SD2 = ISD2 = True # Activa todos los flags relacionados con los seguidores del carril izquierdo.
        elif (not SD1_ATV_PRE and not SD2_ATV_PRE) or PPSD1: # Si no está activo ningún flag asociado a la activación previa de ningún seguidor o se ha producido una PP en el SD1...
            if not SD2 and CISD1[0] == 0 and CISD1[1] == 0: # Si no está activo el SD2 y no hay coordenadas iniciales en el SD1 (tanto en la coordenada x como en la y)...
                CISD1 = deteccion_der[0] # Las coordenadas iniciales del SD1 son asignadas a las coordenadas detectadas.
                SD1 = ACT_SD1 = ISD1 = True # Activa todos los flags relacionados con el SD1.
            else: # Si está activo el SD2 y hay coordenadas iniciales en el SD1 (en la coordenada x o en la coordenada y)...
                if  vect_distance_1_2[3] == 0 and vect_distance_1_2[2] <= 100: # Si la distancia existente entre la detección derecha y la boundingbox del SD2 es nula y la distancia existente entre la detección derecha y la boundingbox del SD1 está por debajo o es igual a un rango...
                    CISD1 = deteccion_der[0] # Las coordenadas iniciales del SD1 son asignadas a las coordenadas detectadas.
                    SD1 = ACT_SD1 = ISD1 = True # Activa todos los flags relacionados con el SD1.
                else: # Si la distancia existente entre la detección derecha y la boundingbox del SD2 no es nula o la distancia existente entre la detección derecha y la boundingbox del SD1 no está por debajo o no es igual a un rango...
                    CISD1 = box_der_1_anterior # Las coordenadas iniciales del SD1 son asignadas a las últimas coordenadas proporcionadas por el SD1.
                    CISD2 = deteccion_der[0] # Las coordenadas iniciales del SD2 son asignadas a las coordenadas detectadas.
                    SD1 = ACT_SD1 = SD2 = ACT_SD2 = ISD1 = ISD2 = True # Activa todos los flags relacionados con el SD1 y el SD2.
```

```
                    SD1_ATV_PRE = True # Activa el flag para indicar que el SD1 está activo previamente al SD2.
        elif (SD2 or PPSD2 or SD2_ATV_PRE): # Si está activo el SD2 o se ha producido una PP en el el SD2 o el flag de activación previa del SD2 está activo ...
            if not SD1 and CISD2[0] == 0 and CISD2[1] == 0: # Si no está activo el SD1 y no hay coordenadas iniciales en el SD2 (tanto en la coordenada x como en la y)...
                CISD2 = deteccion_der[0] # Las coordenadas iniciales del SD2 son asignadas a las coordenadas detectadas.
                SD2 = ACT_SD2 = ISD2 = True # Activa todos los flags relacionados con el SD2.
            else: # Si no está activo el SD2 y no se ha producido una PP en el el SD2 y el flag de activación previa del SD2 no está activo ...
                if vect_distance_1_2[2] == 0 and vect_distance_1_2[3] <= 100: # Si la distancia existente entre la detección derecha y la boundingbox del SD1 es nula y la distancia existente entre la detección derecha y la boundingbox del SD2 está por debajo o es igual a un rango...
                    CISD2 = deteccion_der[0] # Las coordenadas iniciales del SD2 son asignadas a las coordenadas detectadas.
                    SD2 = ACT_SD2 = ISD2 = True # Activa todos los flags relacionados con el SI2.
                    SD2_ATV_PRE = True # Activa el flag para indicar que el SD2 está activo previamente al SD1.
                else: # Si la distancia entre la detección y la caja del SI2 está por encima de un determinado margen (implica que trabajan ambos seguidores)...
                    if vect_distance_1_2[2] >= vect_distance_1_2[3]: # Si la distancia existente entre la detección derecha y la boundingbox del SD1 es mayor o igual a la distancia existente entre la detección derecha y la boundingbox del SD2...
                        CISD1 = box_der_1_anterior # Las coordenadas iniciales del SD1 son asignadas a las últimas coordenadas proporcionadas por el SD1.
                        CISD2 = deteccion_der[0] # Las coordenadas iniciales del SD2 son asignadas a las coordenadas detectadas.
                    else: # Si la distancia existente entre la detección derecha y la boundingbox del SD1 es menor a la distancia existente entre la detección derecha y la boundingbox del SD2...
                        CISD1 = deteccion_der[0] # Las coordenadas iniciales del SD1 son asignadas a las coordenadas detectadas.
                        CISD2 = box_der_2_anterior # Las coordenadas iniciales del SD2 son asignadas a las últimas coordenadas proporcionadas por el SD2.
                    SD1 = ACT_SD1 = SD2 = ACT_SD2 = ISD1 = ISD2 = True # Activa todos los flags relacionados con los seguidores del carril derecho.

#Función de seguimiento:
def tracking( roi_lane_left_end, roi_lane_right_end):

    global fotogramas, SI1, SI2, SD1, SD2, PPSI1, PPSI2, PPSD1, PPSD2, ContadorPPSI1, ContadorPPSI2, ContadorPPSD1, ContadorPPSD2, SI1_OK, SI2_OK, SD1_OK, SD2_OK, ACT_SI1, ACT_SI2, ACT_SD1, ACT_SD2 # Definición de variables globales empleadas en la función.
```

```
        global box_izq_1_anterior, box_izq_2_anterior, box_der_1_anterior, box_der_2_anterior, CISI1, CISI2, CISD1, CISD2 # Definición de variables globales empleadas en la función.
        global tracker_izq_1, tracker_izq_2, tracker_der_1, tracker_der_2, seguimiento_izq_on, seguimiento_der_on, contador_vehi_izq, contador_vehi_der, PTSI1, PTSI2, PTSD1, PTSD2 # Definición de variables globales empleadas en la función.
    global ISI1, ISI2, ISD1, ISD2, SI1_ATV_PRE, SI2_ATV_PRE, SD1_ATV_PRE, SD2_ATV_PRE # Definición de variables globales empleadas en la función.

    FlagRSI1 = FlagRSI2 = FlagRSD1 = FlagRSD2 = IncrLeft = IncrRight = False # Definición de variables locales empleadas en la función. Inicialización a False.
     box_izq_1 = box_izq_2 = box_der_1 = box_der_2 = [0, 0, 0, 0] # Definición de variables locales empleadas en la función. Inicialización con tamaño y valor (0) definidos.

    if fotogramas == 5: # Si el contador de fotogramas es igual a 5...
        detect(subtraction_left, subtraction_right) # Llamada a la función detect para refrescar la detección.
        fotogramas = 0 # Reinicialización del contador.

    if (SI1 and seguimiento_izq_on): # Si el primer seguidor izquierdo está activo y existe detección en el carril izquierdo...
        if ISI1 or PPSI1: # Si la inicialización del primer seguidor izquierdo está activa o se ha producido una pérdida parcial en el mismo...
            tracker_izq_1 = cv2.TrackerKCF_create() # Creación del primer seguidor izquierdo.
            retval = tracker_izq_1.init(roi_lane_left_end, CISI1) # Inicialización del primer seguidor izquierdo.
            PPSI1 = False # Inabilitación de la pérdida parcial en el primer seguidor izquierdo.
            if retval is None: # Si la inicialización del primer seguidor izquierdo fue adecuada...
                SI1_OK, box_izq_1 = tracker_izq_1.update(roi_lane_left_end) # Actualización del primer seguidor izquierdo en el carril correspondiente.
                box_izq_1_anterior = box_izq_1 # Guardado de coordenadas proporcionadas por la actualización del primer seguidor izquierdo.
                ACT_SI1 = True # Activación del flag asociado a la actualización del primer seguidor izquierdo.
                ISI1 = False # Desactivación del flag asociado a la inicialización del primer seguidor izquierdo.
            else: # Si la inicialización del primer seguidor izquierdo no fue adecuada...
                SI1 = ISI1 = True # Primer seguidor izquierdo e inicialización del mismo habilitados.
                ACT_SI1 = PPSI1 = False # Se desactivan los flags asociados a la actualización del primer seguidor izquierdo y a la pérdida parcial del mismo.
        elif ACT_SI1: # Si el primer seguidor izquierdo se tiene que actualizar...
            SI1_OK, box_izq_1 = tracker_izq_1.update(roi_lane_left_end) # Actualización del primer seguidor izquierdo en el carril correspondiente.
```

```
            if SI1_OK and ContadorPPSI1 < 2 and box_izq_1[1] < 300: # Si la actualización del primer seguidor izquierdo fue adecuada y no se han producido más de dos pérdidas parciales en el mismo y la coordenada Y de la boundingbox proporcionada está por debajo de un determinado rango...
                box_izq_1_anterior = box_izq_1 # Guardado de coordenadas proporcionadas por la actualización del primer seguidor izquierdo.
                 FlagRSI1 = True # Activación del flag de representación asociado al primer seguidor izquierdo.
                draw (box_izq_1[:], box_izq_2[:], box_der_1[:], box_der_2[:], FlagRSI1, FlagRSI2, FlagRSD1, FlagRSD2) # Llamada a la función draw para la representación visual del seguidor correspondiente.
             else: # Si la actualización del primer seguidor izquierdo no fue adecuada o se han producido más de dos pérdidas parciales en el mismo o la coordenada Y de la boundingbox proporcionada es igual o mayor a un determinado rango...
                 if (ContadorPPSI1 >= 2 or (box_izq_1_anterior[1]>=300)): # Si existen dos o más pérdidas parciales en el primer seguidor izquierdo o la coordernada Y de la última boundingbox proporcionada es mayor o igual a un determinado rango...
                     PTSI1 = True # Se produce la pérdida total del primer seguidor izquierdo.
                  ACT_SI1 = False # Desactivación del flag asociado a la actualización del primer seguidor izquierdo.
                else: # Si no existen dos o más pérdidas parciales en el primer seguidor izquierdo y la coordernada Y de la última boundingbox proporcionada es menor a un determinado rango...
                    ContadorPPSI1 += 1 # Incremento en una unidad el contador de pérdidas parciales del primer seguidor izquierdo.
                     ACT_SI1 = PPSI1 = False # Inabilitación de la actualización del primer seguidor izquierdo y la pérdida parcial en el mismo.
         else: # Si el primer seguidor izquierdo no se tiene que actualizar y está en uso...
             if PTSI1: # Si existe pérdida total del primer seguidor izquierdo...
                SI1 = SI1_OK = ACT_SI1 =  PTSI1 = PPSI1 = FlagRSI1 = False # Se desactiva el primer seguidor izquierdo y todos los flags asociados al mismo.
                 tracker_izq_1 = cv2.TrackerKCF_create() # Creación del primer seguidor izquierdo.
                  box_izq_1 = box_izq_1_anterior = CISI1 = [0, 0, 0, 0] # Anulación de todas las coordenadas asociadas al primer seguidor izquierdo.
                contador_vehi_izq += 1 # Incremento en una unidad el número de vehículos detectados en el carril izquierdo.
                 ContadorPPSI1 = 0 # Reinicialización del contador de pérdidas parciales asociado al primer seguidor izquierdo.
                ISI1 = IncrLeft = True # Activación del flag asociado a la inicialización del primer seguidor izquierdo y del flag asociado a la escritura en el reporte de detecciones.
                  if SI1_ATV_PRE: # Si el flag para indicar que el primer seguidor izquierdo está activo previamente al segundo seguidor izquierdo está activo...
                     SI1_ATV_PRE = False# Desactivación del flag para indicar que el primer seguidor izquierdo está activo previamente al segundo seguidor izquierdo.
```

```
                    SI2_ATV_PRE = True # Activación del flag para indicar que el segundo seguidor izquierdo está activo previamente al primer seguidor izquierdo.
                if not SI2: # Si el segundo seguidor izquierdo no está activo...
                        seguimiento_izq_on = False # Seguimiento inactivo en el carril izquierdo.
              elif not PPSI1: # Si no existe pérdida parcial en el primer seguidor izquierdo...
                    detect( subtraction_left, subtraction_right) # Llamada a la función detect para refrescar la detección.
                if ACT_SI1: # Si el flag asociado a la actualización del primer seguidor izquierdo está activo...
                        PPSI1 = True # Habilitación de la pérdida parcial en el primer seguidor izquierdo.
                    ACT_SI1 = False # Desactivación del flag asociado a la actualización del primer seguidor izquierdo.
                else: # Si el flag asociado a la actualización del primer seguidor izquierdo está inhabilitado...
                        ISI1 = ACT_SI1 = PPSI1 = False # Se desactiva el primer seguidor izquierdo y parte de los flags asociados al mismo.
                    ContadorPPSI1 += 1 # Incremento en una unidad el contador de pérdidas parciales del primer seguidor izquierdo.
                    if ContadorPPSI1 >= 2: # Si el contador de pérdidas parciales del primer seguidor izquierdo es mayor o igual a dos...
                            PTSI1 = True # Habilitación de la pérdida total del primer seguidor izquierdo.

    if (SI2 and seguimiento_izq_on): # Si el segundo seguidor izquierdo está activo y existe detección en el carril izquierdo...
        if ISI2 or PPSI2: # Si la inicialización del segundo seguidor izquierdo está activa o se ha producido una pérdida parcial en el mismo...
             tracker_izq_2 = cv2.TrackerKCF_create() # Creación del segundo seguidor izquierdo.
            retval = tracker_izq_2.init(roi_lane_left_end, CISI2) # Inicialización del segundo seguidor izquierdo.
            PPSI2 = False # Inabilitación de la pérdida parcial en el segundo seguidor izquierdo.
             if retval is None: # Si la inicialización del segundo seguidor izquierdo fue adecuada...
                    SI2_OK, box_izq_2 = tracker_izq_2.update(roi_lane_left_end) # Actualización del segundo seguidor izquierdo en el carril correspondiente.
                box_izq_2_anterior = box_izq_2 # Guardado de coordenadas proporcionadas por la actualización del segundo seguidor izquierdo.
                 ACT_SI2 = True # Activación del flag asociado a la actualización del segundo seguidor izquierdo.
                ISI2 = False # Desactivación del flag asociado a la inicialización del segundo seguidor izquierdo.
            else: # Si la inicialización del segundo seguidor izquierdo no fue adecuada...
```

```
                SI2 = ISI2 = True # Segundo seguidor izquierdo e inicialización del mismo habilitados.
                ACT_SI2 = PPSI2 = False # Se desactivan los flags asociados a la actualización del segundo seguidor izquierdo y a la pérdida parcial del mismo.
        elif ACT_SI2: # Si el segundo seguidor izquierdo se tiene que actualizar...
            SI2_OK, box_izq_2 = tracker_izq_2.update(roi_lane_left_end) # Actualización del segundo seguidor izquierdo en el carril correspondiente.
            if SI2_OK and ContadorPPSI2 < 2 and box_izq_2[1] < 300: # Si la actualización del segundo seguidor izquierdo fue adecuada y no se han producido más de dos pérdidas parciales en el mismo y la coordenada Y de la boundingbox proporcionada está por debajo de un determinado rango...
                box_izq_2_anterior = box_izq_2 # Guardado de coordenadas proporcionadas por la actualización del segundo seguidor izquierdo.
                FlagRSI2 = True # Activación del flag de representación asociado al segundo seguidor izquierdo.
                draw (box_izq_1[:], box_izq_2[:], box_der_1[:], box_der_2[:], FlagRSI1, FlagRSI2, FlagRSD1, FlagRSD2) # Llamada a la función draw para la representación visual del seguidor correspondiente.
            else: # Si la actualización del segundo seguidor izquierdo no fue adecuada o se han producido más de dos pérdidas parciales en el mismo o la coordenada Y de la boundingbox proporcionada es igual o mayor a un determinado rango...
                if (ContadorPPSI2 >= 2 or (box_izq_2_anterior[1]>=300)): # Si existen dos o más pérdidas parciales en el segundo seguidor izquierdo o la coordernada Y de la última boundingbox proporcionada es mayor o igual a un determinado rango...
                    PTSI2 = True # Se produce la pérdida total del segundo seguidor izquierdo.
                    ACT_SI2 = False # Desactivación del flag asociado a la actualización del segundo seguidor izquierdo.
                else: # Si no existen dos o más pérdidas parciales en el segundo seguidor izquierdo y la coordernada Y de la última boundingbox proporcionada es menor a un determinado rango...
                    ContadorPPSI2 += 1 # Incremento en una unidad el contador de pérdidas parciales del segundo seguidor izquierdo.
                    ACT_SI2 = PPSI2 = False # Inabilitación de la actualización del segundo seguidor izquierdo y la pérdida parcial en el mismo.
        else: # Si el segundo seguidor izquierdo no se tiene que actualizar y está en uso...
            if PTSI2: # Si existe pérdida total del segundo seguidor izquierdo...
                SI2 = SI2_OK = ACT_SI2 =  PTSI2 = PPSI2 = FlagRSI2 = False # Se desactiva el segundo seguidor izquierdo y todos los flags asociados al mismo.
                tracker_izq_2 = cv2.TrackerKCF_create() # Creación del segundo seguidor izquierdo.
                box_izq_2 = box_izq_2_anterior = CISI2 = [0, 0, 0, 0] # Anulación de todas las coordenadas asociadas al segundo seguidor izquierdo.
                contador_vehi_izq += 1 # Incremento en una unidad el número de vehículos detectados en el carril izquierdo.
                ContadorPPSI2 = 0 # Reinicialización del contador de pérdidas parciales asociado al segundo seguidor izquierdo.
```

```
                ISI2 = IncrLeft = True # Activación del flag asociado a la inicialización del segundo seguidor izquierdo y del flag asociado a la escritura en el reporte de detecciones.
                    if SI2_ATV_PRE: # Si el flag para indicar que el segundo seguidor izquierdo está activo previamente al primer seguidor izquierdo está activo...
                        SI1_ATV_PRE = SI2_ATV_PRE = False # Inabilitación de flags de indicación de seguidores activos previamente.
                 if not SI1: # Si el primer seguidor izquierdo no está activo...
                        seguimiento_izq_on = False # Seguimiento inactivo en el carril izquierdo.
                elif not PPSI2: # Si no existe pérdida parcial en el segundo seguidor izquierdo...
                    detect( subtraction_left, subtraction_right) # Llamada a la función detect para refrescar la detección.
                 if ACT_SI2: # Si el flag asociado a la actualización del segundo seguidor izquierdo está activo...
                        PPSI2 = True # Habilitación de la pérdida parcial en el segundo seguidor izquierdo.
                    ACT_SI2 = False # Desactivación del flag asociado a la actualización del segundo seguidor izquierdo.
                    else: # Si el flag asociado a la actualización del segundo seguidor izquierdo está inhabilitado...
                        ISI2 = ACT_SI2 = PPSI2 = False # Se desactiva el segundo seguidor izquierdo y parte de los flags asociados al mismo.
                    ContadorPPSI2 += 1 # Incremento en una unidad el contador de pérdidas parciales del segundo seguidor izquierdo.
                        if ContadorPPSI2 >= 2: # Si el contador de pérdidas parciales del segundo seguidor izquierdo es mayor o igual a dos...
                            PTSI2 = True # Habilitación de la pérdida total del segundo seguidor izquierdo.

        if (SD1 and seguimiento_der_on): # Si el primer seguidor derecho está activo y existe detección en el carril derecho...
            if ISD1 or PPSD1: # Si la inicialización del primer seguidor derecho está activa o se ha producido una pérdida parcial en el mismo...
                tracker_der_1 = cv2.TrackerKCF_create() # Creación del primer seguidor derecho.
              retval = tracker_der_1.init(roi_lane_right_end, CISD1) # Inicialización del primer seguidor derecho.
                PPSD1 = False # Inabilitación de la pérdida parcial en el primer seguidor derecho.
                if retval is None: # Si la inicialización del primer seguidor derecho fue adecuada...
                        SD1_OK, box_der_1 = tracker_der_1.update(roi_lane_right_end) # Actualización del primer seguidor derecho en el carril correspondiente.
                    box_der_1_anterior = box_der_1 # Guardado de coordenadas proporcionadas por la actualización del primer seguidor derecho.
```

```
                ACT_SD1 = True # Activación del flag asociado a la actualización del primer seguidor derecho.
                ISD1 = False # Desactivación del flag asociado a la inicialización del primer seguidor derecho.
            else: # Si la inicialización del primer seguidor derecho no fue adecuada...
                SD1 = ISD1 = True # Primer seguidor derecho e inicialización del mismo habilitados.
                ACT_SD1 = PPSD1 = False # Se desactivan los flags asociados a la actualización del primer seguidor derecho y a la pérdida parcial del mismo.
        elif ACT_SD1: # Si el primer seguidor derecho se tiene que actualizar...
            SD1_OK, box_der_1 = tracker_der_1.update(roi_lane_right_end) # Actualización del primer seguidor derecho en el carril correspondiente.
            if SD1_OK and ContadorPPSD1 < 2 and box_der_1[1] > 30: # Si la actualización del primer seguidor derecho fue adecuada y no se han producido más de dos pérdidas parciales en el mismo y la coordenada Y de la boundingbox proporcionada está por encima de un determinado rango...
                box_der_1_anterior = box_der_1 # Guardado de coordenadas proporcionadas por la actualización del primer seguidor derecho.
                FlagRSD1 = True # Activación del flag de representación asociado al primer seguidor derecho.
                draw (box_izq_1[:], box_izq_2[:], box_der_1[:], box_der_2[:], FlagRSI1, FlagRSI2, FlagRSD1, FlagRSD2) # Llamada a la función draw para la representación visual del seguidor correspondiente.
            else: # Si la actualización del primer seguidor derecho no fue adecuada o se han producido más de dos pérdidas parciales en el mismo o la coordenada Y de la boundingbox proporcionada es igual o menor a un determinado rango...
                if (ContadorPPSD1 >= 2 or (box_der_1_anterior[1]<=30)): # Si existen dos o más pérdidas parciales en el primer seguidor derecho o la coordernada Y de la última boundingbox proporcionada es menor o igual a un determinado rango...
                    PTSD1 = True # Se produce la pérdida total del primer seguidor derecho.
                    ACT_SD1 = False # Desactivación del flag asociado a la actualización del primer seguidor derecho.
                else: # Si no existen dos o más pérdidas parciales en el primer seguidor derecho y la coordernada Y de la última boundingbox proporcionada es mayor a un determinado rango...
                    ContadorPPSD1 += 1 # Incremento en una unidad el contador de pérdidas parciales del primer seguidor derecho.
                    ACT_SD1 = PPSD1 = False # Inabilitación de la actualización del primer seguidor derecho y la pérdida parcial en el mismo.
        else: # Si el primer seguidor derecho no se tiene que actualizar y está en uso...
            if PTSD1: # Si existe pérdida total del primer seguidor derecho...
                SD1 = SD1_OK = ACT_SD1 =  PTSD1 = PPSD1 = FlagRSD1 = False # Se desactiva el primer seguidor derecho y todos los flags asociados al mismo.
                tracker_der_1 = cv2.TrackerKCF_create() # Creación del primer seguidor derecho.
```

```
                box_der_1 = box_der_1_anterior = CISD1 = [0, 0, 0, 0] # Anulación de todas las coordenadas asociadas al primer seguidor derecho.
                contador_vehi_der += 1 # Incremento en una unidad el número de vehículos detectados en el carril derecho.
                ContadorPPSD1 = 0 # Reinicialización del contador de pérdidas parciales asociado al primer seguidor derecho.
                ISD1 = IncrRight = True # Activación del flag asociado a la inicialización del primer seguidor derecho y del flag asociado a la escritura en el reporte de detecciones.
                if SD1_ATV_PRE: # Si el flag para indicar que el primer seguidor derecho está activo previamente al segundo seguidor derecho está activo...
                    SD1_ATV_PRE = False # Desactivación del flag para indicar que el primer seguidor derecho está activo previamente al segundo seguidor derecho.
                    SD2_ATV_PRE = True # Activación del flag para indicar que el segundo seguidor derecho está activo previamente al primer seguidor derecho.
                if not SD2: # Si el segundo seguidor derecho no está activo...
                    seguimiento_der_on = False # Seguimiento inactivo en el carril derecho.
            elif not PPSD1: # Si no existe pérdida parcial en el primer seguidor derecho...
                detect(subtraction_left, subtraction_right) # Llamada a la función detect para refrescar la detección.
                if ACT_SD1: # Si el flag asociado a la actualización del primer seguidor derecho está activo...
                    PPSD1 = True # Habilitación de la pérdida parcial en el primer seguidor derecho.
                    ACT_SD1 = False # Desactivación del flag asociado a la actualización del primer seguidor derecho.
                else: # Si el flag asociado a la actualización del primer seguidor derecho está inhabilitado...
                    ISD1 = ACT_SD1 = PPSD1 = False # Se desactiva el primer seguidor derecho y parte de los flags asociados al mismo.
                    ContadorPPSD1 += 1 # Incremento en una unidad el contador de pérdidas parciales del primer seguidor derecho.
                    if ContadorPPSD1 >= 2: # Si el contador de pérdidas parciales del primer seguidor derecho es mayor o igual a dos...
                        PTSD1 = True # Habilitación de la pérdida total del primer seguidor derecho.

    if (SD2 and seguimiento_der_on): # Si el segundo seguidor derecho está activo y existe detección en el carril derecho...
        if ISD2 or PPSD2: # Si la inicialización del segundo seguidor derecho está activa o se ha producido una pérdida parcial en el mismo...
            tracker_der_2 = cv2.TrackerKCF_create() # Creación del segundo seguidor derecho.
            retval = tracker_der_2.init(roi_lane_right_end, CISD2) # Inicialización del segundo seguidor derecho.
            PPSD2 = False # Inabilitación de la pérdida parcial en el segundo seguidor derecho.
```

```
            if retval is None: # Si la inicialización del segundo seguidor derecho fue adecuada...
                    SD2_OK, box_der_2 = tracker_der_2.update(roi_lane_right_end) # Actualización del segundo seguidor derecho en el carril correspondiente.
                box_der_2_anterior = box_der_2 # Guardado de coordenadas proporcionadas por la actualización del segundo seguidor derecho.
                 ACT_SD2 = True # Activación del flag asociado a la actualización del segundo seguidor derecho.
                ISD2 = False # Desactivación del flag asociado a la inicialización del segundo seguidor derecho.
            else: # Si la inicialización del segundo seguidor derecho no fue adecuada...
                SD2 = ISD2 = True # Segundo seguidor derecho e inicialización del mismo habilitados.
                   ACT_SD2 = PPSD2 = False # Se desactivan los flags asociados a la actualización del segundo seguidor derecho y a la pérdida parcial del mismo.
        elif ACT_SD2: # Si el segundo seguidor derecho se tiene que actualizar...
            SD2_OK, box_der_2 = tracker_der_2.update(roi_lane_right_end) # Actualización del segundo seguidor derecho en el carril correspondiente.
            if SD2_OK and ContadorPPSD2 < 2 and box_der_2[1] > 30: # Si la actualización del segundo seguidor derecho fue adecuada y no se han producido más de dos pérdidas parciales en el mismo y la coordenada Y de la boundingbox proporcionada está por encima de un determinado rango...
                box_der_2_anterior = box_der_2 # Guardado de coordenadas proporcionadas por la actualización del segundo seguidor derecho.
                  FlagRSD2 = True # Activación del flag de representación asociado al segundo seguidor derecho.
                draw (box_izq_1[:], box_izq_2[:], box_der_1[:], box_der_2[:], FlagRSI1, FlagRSI2, FlagRSD1, FlagRSD2) # Llamada a la función draw para la representación visual del seguidor correspondiente.
            else: # Si la actualización del segundo seguidor derecho no fue adecuada o se han producido más de dos pérdidas parciales en el mismo o la coordenada Y de la boundingbox proporcionada es igual o menor a un determinado rango...
                 if (ContadorPPSD2 >= 2 or (box_der_2_anterior[1]<=30)): # Si existen dos o más pérdidas parciales en el segundo seguidor derecho o la coordernada Y de la última boundingbox proporcionada es menor o igual a un determinado rango...
                    PTSD2 = True # Se produce la pérdida total del segundo seguidor derecho.
                   ACT_SD2 = False # Desactivación del flag asociado a la actualización del segundo seguidor derecho.
                else: # Si no existen dos o más pérdidas parciales en el segundo seguidor derecho y la coordernada Y de la última boundingbox proporcionada es mayor a un determinado rango...
                    ContadorPPSD2 += 1 # Incremento en una unidad el contador de pérdidas parciales del segundo seguidor derecho.
                    ACT_SD2 = PPSD2 = False # Inabilitación de la actualización del segundo seguidor derecho y la pérdida parcial en el mismo.
        else: # Si el segundo seguidor derecho no se tiene que actualizar y está en uso...
            if PTSD2: # Si existe pérdida total del segundo seguidor derecho...
```

```
                SD2 = SD2_OK = ACT_SD2 =  PTSD2 = PPSD2 = FlagRSD2 = False # Se desactiva el segundo seguidor derecho y todos los flags asociados al mismo.
                tracker_der_2 = cv2.TrackerKCF_create() # Creación del segundo seguidor derecho.
                 box_der_2 = box_der_2_anterior = CISD2 = [0, 0, 0, 0] # Anulación de todas las coordenadas asociadas al segundo seguidor derecho.
                contador_vehi_der += 1 # Incremento en una unidad el número de vehículos detectados en el carril derecho.
                ContadorPPSD2 = 0 # Reinicialización del contador de pérdidas parciales asociado al segundo seguidor derecho.
               ISD2 = IncrRight = True # Activación del flag asociado a la inicialización del segundo seguidor derecho y del flag asociado a la escritura en el reporte de detecciones.
                  if SD2_ATV_PRE: # Si el flag para indicar que el segundo seguidor derecho está activo previamente al primer seguidor derecho está activo...
                      SD1_ATV_PRE = SD2_ATV_PRE = False # Inabilitación de flags de indicación de seguidores activos previamente.
                if not SD1: # Si el primer seguidor derecho no está activo...
                      seguimiento_der_on = False # Seguimiento inactivo en el carril derecho.
              elif not PPSD2: # Si no existe pérdida parcial en el segundo seguidor derecho...
                  detect(subtraction_left, subtraction_right) # Llamada a la función detect para refrescar la detección.
                if ACT_SD2: # Si el flag asociado a la actualización del segundo seguidor derecho está activo...
                     PPSD2 = True # Habilitación de la pérdida parcial en el segundo seguidor derecho.
                   ACT_SD2 = False # Desactivación del flag asociado a la actualización del segundo seguidor derecho.
                 else: # Si el flag asociado a la actualización del segundo seguidor derecho está inhabilitado...
                    ISD2 = ACT_SD2 = PPSD2 = False # Se desactiva el segundo seguidor derecho y parte de los flags asociados al mismo.
                      ContadorPPSD2 += 1  # Incremento en una unidad el contador de pérdidas parciales del segundo seguidor derecho.
                    if ContadorPPSD2 >= 2: # Si el contador de pérdidas parciales del segundo seguidor derecho es mayor o igual a dos...
                        PTSD2 = True # Habilitación de la pérdida total del segundo seguidor derecho.

    if IncrRight or IncrLeft: # Si está habilitado cualquier flag asociado a la escritura en el reporte de detecciones...
        writefile(IncrLeft ,IncrRight, contador_vehi_izq, contador_vehi_der) # Llamada a la función writefile para la escritura en el reporte de detecciones.
         IncrLeft = IncrRight = False # Inhabilitación de los flags asociados a la escritura en el reporte de detecciones.
```

```
    fotogramas += 1 # Incremento en una unidad el contador de fotogramas.

# Función de escritura en fichero:
def writefile ( IncrLeft ,IncrRight, contador_vehi_izq, contador_vehi_der ):

    global video_capture # Definición de variable global empleada en la función.

     with open('DATA_DETECTION_TFG_JSBM.txt', 'a', encoding = 'utf-8') as file: # Apertura de fichero con nombre DATA_DETECTION_TFG_JSBM.txt en la variable file y en modo append (añadido).
        if IncrLeft: # Si está habilitado el flag asociado a la escritura en el reporte de detecciones en el carril izquierdo...
             left_pos = video_capture.get(cv2.CAP_PROP_POS_MSEC)*0.001 # Obtención de la marca temporal del frame tratado y conversión a segundos.
            if left_pos >= 60: # Si la marca temporal obtenida es mayor o igual a un determinado rango...
                left_pos = divmod(left_pos, 60) # Obtención de cociente y el resto de la división entre la marca temporal y 60.
                  if left_pos[0] == 1: # Si el cociente de la división realizada anteriormente es igual a uno...
                    file.write(' LEFT Detection number ' + str(contador_vehi_izq-1) + '. Time detection: ' + str(left_pos[0]) + ' minute and ' +  str(round(left_pos[1], 3)) + ' seconds\r') # Escritura de una línea en el reporte de detecciones.
                else: # Si el cociente de la división realizada anteriormente es distinto a uno...
                    file.write(' LEFT Detection number ' + str(contador_vehi_izq-1) + '. Time detection: ' + str(left_pos[0]) + ' minutes and ' +  str(round(left_pos[1], 3)) + ' seconds\r') # Escritura de una línea en el reporte de detecciones.
            else: # Si la marca temporal obtenida es menor a un determinado rango...
                file.write(' LEFT Detection number ' + str(contador_vehi_izq-1) + '. Time detection: ' + str(round(left_pos, 3)) + ' seconds\r') # Escritura de una línea en el reporte de detecciones.
        if IncrRight: # Si está habilitado el flag asociado a la escritura en el reporte de detecciones en el carril derecho...
            right_pos = video_capture.get(cv2.CAP_PROP_POS_MSEC)*0.001 # Obtención de la marca temporal del frame tratado y conversión a segundos.
            if right_pos >= 60: # Si la marca temporal obtenida es mayor o igual a un determinado rango...
                right_pos = divmod(right_pos, 60) # Obtención de cociente y el resto de la división entre la marca temporal y 60.
                  if right_pos[0] == 1: # Si el cociente de la división realizada anteriormente es igual a uno...
                    file.write(' RIGHT Detection number ' + str(contador_vehi_der-1) + '. Time detection: ' + str(right_pos[0]) + ' minute and ' +  str(round(right_pos[1], 3)) + ' seconds\r') # Escritura de una línea en el reporte de detecciones.
                else: # Si el cociente de la división realizada anteriormente es distinto a uno...
```

```
                    file.write(' RIGHT Detection number ' + str(contador_vehi_der-1) +
'. Time detection: ' + str(right_pos[0]) + ' minutes and ' +  str(round(right_pos[1],
3)) + ' seconds\r') # Escritura de una línea en el reporte de detecciones.
            else: # Si la marca temporal obtenida es menor a un determinado rango...
                file.write('RIGHT Detection number ' + str(contador_vehi_der-1) + '.
Time detection: ' + str(round(right_pos, 3)) + ' seconds\r') # Escritura de una línea
en el reporte de detecciones.
        file.write('*--------------------* \r*') # Escritura de una línea separatoria
en el reporte de detecciones.

# Función de representación:
def draw ( box_izq_1, box_izq_2, box_der_1, box_der_2, FlagRSI1, FlagRSI2, FlagRSD1,
FlagRSD2 ):
    if FlagRSI1: # Si está activo del flag de representación asociado al primer seguidor
izquierdo...
                        cv2.rectangle(roi_lane_left_end,  (box_izq_1[0],box_izq_1[1]),
(box_izq_1[0]+box_izq_1[2],   box_izq_1[1]+box_izq_1[3]),   (255,   0,   0),   3)   #
Representación en el carril izquierdo de la boundingbox proporcionada por el primer
seguidor izquierdo.
                            cv2.putText(roi_lane_left_end,   "SI1",   (box_izq_1[0],
box_izq_1[1]),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, lineType= cv2.LINE_AA)
# Representación en el carril izquierdo del identificador del primer seguidor izquierdo.
                            cv2.putText(roi_lane_left_end,   str(contador_vehi_izq),
(box_izq_1[0]+(box_izq_1[2]-50),  box_izq_1[1]),cv2.FONT_HERSHEY_SIMPLEX,  1,  (0,  0,
255), 2, lineType= cv2.LINE_AA) # Representación del conteo de vehículos actual en el
carril izquierdo.
    if FlagRSI2: # Si está activo del flag de representación asociado al segundo seguidor
izquierdo...
                        cv2.rectangle(roi_lane_left_end,  (box_izq_2[0],box_izq_2[1]),
(box_izq_2[0]+box_izq_2[2],   box_izq_2[1]+box_izq_2[3]),   (255,   0,   0),   3)   #
Representación en el carril izquierdo de la boundingbox proporcionada por el segundo
seguidor izquierdo.
                            cv2.putText(roi_lane_left_end,   "SI2",   (box_izq_2[0],
box_izq_2[1]),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, lineType= cv2.LINE_AA)
# Representación en el carril izquierdo del identificador del segundo seguidor
izquierdo.
                            cv2.putText(roi_lane_left_end,   str(contador_vehi_izq),
(box_izq_2[0]+(box_izq_2[2]-50),  box_izq_2[1]),cv2.FONT_HERSHEY_SIMPLEX,  1,  (0,  0,
255), 2, lineType= cv2.LINE_AA) # Representación del conteo de vehículos actual en el
carril izquierdo.
    if FlagRSD1: # Si está activo del flag de representación asociado al primer seguidor
derecho...
                        cv2.rectangle(roi_lane_right_end,  (box_der_1[0],box_der_1[1]),
(box_der_1[0]+box_der_1[2],   box_der_1[1]+box_der_1[3]),   (255,   0,   0),   3)   #
Representación en el carril derecho de la boundingbox proporcionada por el primer
seguidor derecho.
                            cv2.putText(roi_lane_right_end,   "SD1",   (box_der_1[0],
box_der_1[1]),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, lineType= cv2.LINE_AA)
# Representación en el carril derecho del identificador del primer seguidor derecho.
```

```
                            cv2.putText(roi_lane_right_end,  str(contador_vehi_der),
(box_der_1[0]+(box_der_1[2]-50),  box_der_1[1]),cv2.FONT_HERSHEY_SIMPLEX,  1,  (0,  0,
255), 2, lineType= cv2.LINE_AA) # Representación del conteo de vehículos actual en el
carril derecho.
    if FlagRSD2: # Si está activo del flag de representación asociado al segundo seguidor
derecho...
                        cv2.rectangle(roi_lane_right_end,  (box_der_2[0],box_der_2[1]),
(box_der_2[0]+box_der_2[2],   box_der_2[1]+box_der_2[3]),   (255,   0,   0),   3)   #
Representación en el carril derecho de la boundingbox proporcionada por el segundo
seguidor derecho.
                            cv2.putText(roi_lane_right_end,  "SD2",  (box_der_2[0],
box_der_2[1]),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, lineType= cv2.LINE_AA)
# Representación en el carril derecho del identificador del segundo seguidor derecho.
                            cv2.putText(roi_lane_right_end,  str(contador_vehi_der),
(box_der_2[0]+(box_der_2[2]-50),  box_der_2[1]),cv2.FONT_HERSHEY_SIMPLEX,  1,  (0,  0,
255), 2, lineType= cv2.LINE_AA) # Representación del conteo de vehículos actual en el
carril derecho.

# Función de cálculo de distancias:
def distance (deteccion_izq, deteccion_der, box_izq_1_anterior, box_izq_2_anterior,
box_der_1_anterior, box_der_2_anterior):

    dist_dect_SI1 = dist_dect_SI2 = dist_dect_SD1 = dist_dect_SD2 = dist_dect_izq =
dist_dect_der = dist_dect_0_SI1 = dist_dect_1_SI1 = dist_dect_0_SI2 = dist_dect_1_SI2
= dist_dect_0_SD1 = dist_dect_1_SD1 = dist_dect_0_SD2 = dist_dect_1_SD2 = 0 # Definición
de variables locales empleadas en la función. Inicialización a cero.
     dist_dect_SI1_0  =  dist_dect_SI1_1  =  dist_dect_SI1_2  =  dist_dect_SI2_0  =
dist_dect_SI2_1 = dist_dect_SI2_2 = dist_dect_SD1_0 = dist_dect_SD1_1 = dist_dect_SD1_2
= dist_dect_SD2_0 = dist_dect_SD2_1 = dist_dect_SD2_2 = 0 # Definición de variables
locales empleadas en la función. Inicialización a cero.

    if ((len(deteccion_izq) == 1 and deteccion_izq[0][2] != 0) or (len(deteccion_der)
== 1 and deteccion_der[0][2] != 0)): # Si existe una única detección en el carril
izquierdo y la coordenada Y de tal detección es distinta de cero o si existe una única
detección en el carril derecho y la coordenada Y de tal detección es distinta de cero...
        if SI1 and len(deteccion_izq) == 1 and box_izq_1_anterior[2] != 0: # Si el
primer seguidor izquierdo está activo y existe una única detección en el carril izquierdo
y la coordenada Y de la última boundingbox es distinta de cero...
            centroid_det_izq = [((deteccion_izq[0][0] + deteccion_izq[0][2]) * 0.5),
((deteccion_izq[0][1] + deteccion_izq[0][3]) * 0.5)] # Se obtiene el centroide de las
coordenadas proporcionadas en la detección del carril izquierdo.
            centroid_box_izq_1 = [((box_izq_1_anterior[0] + box_izq_1_anterior[2]) *
0.5), ((box_izq_1_anterior[1] + box_izq_1_anterior[3]) * 0.5)] # Se obtiene el centroide
de la última boundingbox proporcionada por el primer seguidor izquierdo.
          dist_dect_SI1 = math.dist(centroid_det_izq, centroid_box_izq_1) # Se obtiene
la distancia entre el centroide de la detección en el carril izquierdo y la última
boundingbox proporcionada por el primer seguidor izquierdo.
```

```
        if SI2 and len(deteccion_izq) == 1 and box_izq_2_anterior[2] != 0: # Si el segundo seguidor izquierdo está activo y existe una única detección en el carril izquierdo y la coordenada Y de la última boundingbox es distinta de cero...
            centroid_det_izq = [((deteccion_izq[0][0] + deteccion_izq[0][2]) * 0.5), ((deteccion_izq[0][1] + deteccion_izq[0][3]) * 0.5)] # Se obtiene el centroide de las coordenadas proporcionadas en la detección del carril izquierdo.
            centroid_box_izq_2 = [((box_izq_2_anterior[0] + box_izq_2_anterior[2]) * 0.5), ((box_izq_2_anterior[1] + box_izq_2_anterior[3]) * 0.5)] # Se obtiene el centroide de la última boundingbox proporcionada por el segundo seguidor izquierdo.
            dist_dect_SI2 = math.dist(centroid_det_izq, centroid_box_izq_2) # Se obtiene la distancia entre el centroide de la detección en el carril izquierdo y la última boundingbox proporcionada por el segundo seguidor izquierdo.
        if SD1 and len(deteccion_der) == 1 and box_der_1_anterior[2] != 0: # Si el primer seguidor derecho está activo y existe una única detección en el carril derecho y la coordenada Y de la última boundingbox es distinta de cero...
            centroid_det_der = [((deteccion_der[0][0] + deteccion_der[0][2]) * 0.5), ((deteccion_der[0][1] + deteccion_der[0][3]) * 0.5)] # Se obtiene el centroide de las coordenadas proporcionadas en la detección del carril derecho.
            centroid_box_der_1 = [((box_der_1_anterior[0] + box_der_1_anterior[2]) * 0.5), ((box_der_1_anterior[1] + box_der_1_anterior[3]) * 0.5)] # Se obtiene el centroide de la última boundingbox proporcionada por el primer seguidor derecho.
            dist_dect_SD1 = math.dist(centroid_det_der, centroid_box_der_1) # Se obtiene la distancia entre el centroide de la detección en el carril derecho y la última boundingbox proporcionada por el primer seguidor derecho.
        if SD2 and len(deteccion_der) == 1 and box_der_2_anterior[2] != 0: # Si el segundo seguidor derecho está activo y existe una única detección en el carril derecho y la coordenada Y de la última boundingbox es distinta de cero...
            centroid_det_der = [((deteccion_der[0][0] + deteccion_der[0][2]) * 0.5), ((deteccion_der[0][1] + deteccion_der[0][3]) * 0.5)] # Se obtiene el centroide de las coordenadas proporcionadas en la detección del carril derecho.
            centroid_box_der_2 = [((box_der_2_anterior[0] + box_der_2_anterior[2]) * 0.5), ((box_der_2_anterior[1] + box_der_2_anterior[3]) * 0.5)] # Se obtiene el centroide de la última boundingbox proporcionada por el segundo seguidor derecho.
            dist_dect_SD2 = math.dist(centroid_det_der, centroid_box_der_2) # Se obtiene la distancia entre el centroide de la detección en el carril derecho y la última boundingbox proporcionada por el segundo seguidor derecho.

    elif len(deteccion_izq) == 2 or len(deteccion_der) == 2: # Si existen dos detecciones en el carril izquierdo o existen dos detecciones en el carril derecho...
        if (SI1 or SI2) and len(deteccion_izq) != 0: # Si el primer o el segundo seguidor izquierdo está activo y existe alguna detección en el carril izquierdo...
            centroid_det_izq_1 = [((deteccion_izq[0][0] + deteccion_izq[0][2]) * 0.5), ((deteccion_izq[0][1] + deteccion_izq[0][3]) * 0.5)] # Se obtiene el centroide de las primeras coordenadas proporcionadas en la detección del carril izquierdo.
            centroid_det_izq_2 = [((deteccion_izq[1][0] + deteccion_izq[1][2]) * 0.5), ((deteccion_izq[1][1] + deteccion_izq[1][3]) * 0.5)] # Se obtiene el centroide de las segundas coordenadas proporcionadas en la detección del carril izquierdo.
```

```
            dist_dect_izq = math.dist(centroid_det_izq_1, centroid_det_izq_2) # Se obtiene la distancia entre el centroide de la primera detección y la segunda relacionadas con el carril izquierdo.
        if (SD1 or SD2) and len(deteccion_der) != 0: # Si el primer o el segundo seguidor derecho está activo y existe alguna detección en el carril derecho...
            centroid_det_der_1 = [((deteccion_der[0][0] + deteccion_der[0][2]) * 0.5), ((deteccion_der[0][1] + deteccion_der[0][3]) * 0.5)] # Se obtiene el centroide de las primeras coordenadas proporcionadas en la detección del carril derecho.
            centroid_det_der_2 = [((deteccion_der[1][0] + deteccion_der[1][2]) * 0.5), ((deteccion_der[1][1] + deteccion_der[1][3]) * 0.5)] # Se obtiene el centroide de las segundas coordenadas proporcionadas en la detección del carril derecho.
            dist_dect_der = math.dist(centroid_det_der_1, centroid_det_der_2) # Se obtiene la distancia entre el centroide de la primera detección y la segunda relacionadas con el carril derecho.
        if SI1 and box_izq_1_anterior[2] != 0 and len(deteccion_izq) != 0: # Si el primer seguidor izquierdo está activo y la coordenada Y de la última boundingbox es distinta de cero y existe alguna detección en el carril izquierdo...
            centroid_box_izq_1 = [((box_izq_1_anterior[0] + box_izq_1_anterior[2]) * 0.5), ((box_izq_1_anterior[1] + box_izq_1_anterior[3]) * 0.5)] # Se obtiene el centroide de la última boundingbox proporcionada por el primer seguidor izquierdo.
            dist_dect_0_SI1 = math.dist(centroid_det_izq_1, centroid_box_izq_1) # Se obtiene la distancia entre el centroide de la primera detección en el carril izquierdo y la última boundingbox proporcionada por el primer seguidor izquierdo.
            dist_dect_1_SI1 = math.dist(centroid_det_izq_2, centroid_box_izq_1) # Se obtiene la distancia entre el centroide de la segunda detección en el carril izquierdo y la última boundingbox proporcionada por el primer seguidor izquierdo.
        if SI2 and box_izq_2_anterior[2] != 0 and len(deteccion_izq) != 0: # Si el segundo seguidor izquierdo está activo y la coordenada Y de la última boundingbox es distinta de cero y existe alguna detección en el carril izquierdo...
            centroid_box_izq_2 = [((box_izq_2_anterior[0] + box_izq_2_anterior[2]) * 0.5), ((box_izq_2_anterior[1] + box_izq_2_anterior[3]) * 0.5)] # Se obtiene el centroide de la última boundingbox proporcionada por el segundo seguidor izquierdo.
            dist_dect_0_SI2 = math.dist(centroid_det_izq_1, centroid_box_izq_2) # Se obtiene la distancia entre el centroide de la primera detección en el carril izquierdo y la última boundingbox proporcionada por el segundo seguidor izquierdo.
            dist_dect_1_SI2 = math.dist(centroid_det_izq_2, centroid_box_izq_2) # Se obtiene la distancia entre el centroide de la segunda detección en el carril izquierdo y la última boundingbox proporcionada por el segundo seguidor izquierdo.
        if SD1 and box_der_1_anterior[2] != 0 and len(deteccion_der) != 0: # Si el primer seguidor derecho está activo y la coordenada Y de la última boundingbox es distinta de cero y existe alguna detección en el carril derecho...
            centroid_box_der_1 = [((box_der_1_anterior[0] + box_der_1_anterior[2]) * 0.5), ((box_der_1_anterior[1] + box_der_1_anterior[3]) * 0.5)] # Se obtiene el centroide de la última boundingbox proporcionada por el primer seguidor derecho.
            dist_dect_0_SD1 = math.dist(centroid_det_der_1, centroid_box_der_1) # Se obtiene la distancia entre el centroide de la primera detección en el carril derecho y la última boundingbox proporcionada por el primer seguidor derecho.
```

```
            dist_dect_1_SD1 = math.dist(centroid_det_der_2, centroid_box_der_1) # Se obtiene la distancia entre el centroide de la segunda detección en el carril derecho y la última boundingbox proporcionada por el primer seguidor derecho.
        if SD2 and box_der_2_anterior[2] != 0 and len(deteccion_izq) != 0: # Si el segundo seguidor derecho está activo y la coordenada Y de la última boundingbox es distinta de cero y existe alguna detección en el carril derecho...
            centroid_box_der_2 = [((box_der_2_anterior[0] + box_der_2_anterior[2]) * 0.5), ((box_der_2_anterior[1] + box_der_2_anterior[3]) * 0.5)] # Se obtiene el centroide de la última boundingbox proporcionada por el segundo seguidor derecho.
            dist_dect_0_SD2 = math.dist(centroid_det_der_1, centroid_box_der_2) # Se obtiene la distancia entre el centroide de la primera detección en el carril derecho y la última boundingbox proporcionada por el segundo seguidor derecho.
            dist_dect_1_SD2 = math.dist(centroid_det_der_2, centroid_box_der_2) #  Se obtiene la distancia entre el centroide de la segunda detección en el carril derecho y la última boundingbox proporcionada por el segundo seguidor derecho.

    elif ((len(deteccion_izq) == 3 and SI1 and SI2) or (len(deteccion_der) == 3 and SD1 and SD2)): # Si existen tres detecciones en el carril izquierdo y los dos seguidores asociados al carril izquierdo están activos o si existen tres detecciones en el carril derecho y los dos seguidores asociados al carril derecho están activos...
        if len(deteccion_izq) == 3: # Si existen tres detecciones en el carril izquierdo...
            centroid_det_izq_0 = [((deteccion_izq[0][0] + deteccion_izq[0][2]) * 0.5), ((deteccion_izq[0][1] + deteccion_izq[0][3]) * 0.5)] # Se obtiene el centroide de las primeras coordenadas proporcionadas en la detección del carril izquierdo.
            centroid_det_izq_1 = [((deteccion_izq[1][0] + deteccion_izq[1][2]) * 0.5), ((deteccion_izq[1][1] + deteccion_izq[1][3]) * 0.5)] # Se obtiene el centroide de las segundas coordenadas proporcionadas en la detección del carril izquierdo.
            centroid_det_izq_2 = [((deteccion_izq[2][0] + deteccion_izq[2][2]) * 0.5), ((deteccion_izq[2][1] + deteccion_izq[2][3]) * 0.5)] # Se obtiene el centroide de las terceras coordenadas proporcionadas en la detección del carril izquierdo.
        if len(deteccion_der) == 3:
            centroid_det_der_0 = [((deteccion_der[0][0] + deteccion_der[0][2]) * 0.5), ((deteccion_der[0][1] + deteccion_der[0][3]) * 0.5)] # Se obtiene el centroide de las primeras coordenadas proporcionadas en la detección del carril derecho.
            centroid_det_der_1 = [((deteccion_der[1][0] + deteccion_der[1][2]) * 0.5), ((deteccion_der[1][1] + deteccion_der[1][3]) * 0.5)] # Se obtiene el centroide de las segundas coordenadas proporcionadas en la detección del carril derecho.
            centroid_det_der_2 = [((deteccion_der[2][0] + deteccion_der[2][2]) * 0.5), ((deteccion_der[2][1] + deteccion_der[2][3]) * 0.5)] # Se obtiene el centroide de las terceras coordenadas proporcionadas en la detección del carril derecho.
        if box_izq_1_anterior[2] != 0 or box_izq_2_anterior[2] != 0: # Si la coordenada Y de la última boundingbox proporcionada por el primer seguidor izquierdo es distinta de cero o si la coordenada Y de la última boundingbox proporcionada por el segundo seguidor izquierdo es distinta de cero...
            if box_izq_1_anterior[2] != 0: # Si la coordenada Y de la última boundingbox proporcionada por el primer seguidor izquierdo es distinta de cero...
```

```
                centroid_box_izq_1 = [((box_izq_1_anterior[0] + box_izq_1_anterior[2])
* 0.5), ((box_izq_1_anterior[1] + box_izq_1_anterior[3]) * 0.5)] # Se obtiene el
centroide de la última boundingbox proporcionada por el primer seguidor izquierdo.
                dist_dect_SI1_0 = math.dist(centroid_det_izq_0, centroid_box_izq_1) #
Se obtiene la distancia entre el centroide de la primera detección en el carril izquierdo
y la última boundingbox proporcionada por el primer seguidor izquierdo.
                dist_dect_SI1_1 = math.dist(centroid_det_izq_1, centroid_box_izq_1) #
Se obtiene la distancia entre el centroide de la segunda detección en el carril izquierdo
y la última boundingbox proporcionada por el primer seguidor izquierdo.
                dist_dect_SI1_2 = math.dist(centroid_det_izq_2, centroid_box_izq_1) #
Se obtiene la distancia entre el centroide de la tercera detección en el carril izquierdo
y la última boundingbox proporcionada por el primer seguidor izquierdo.
            if box_izq_2_anterior[2] != 0: # Si la coordenada Y de la última boundingbox
proporcionada por el segundo seguidor izquierdo es distinta de cero...
                centroid_box_izq_2 = [((box_izq_2_anterior[0] + box_izq_2_anterior[2])
* 0.5), ((box_izq_2_anterior[1] + box_izq_2_anterior[3]) * 0.5)] # Se obtiene el
centroide de la última boundingbox proporcionada por el segundo seguidor izquierdo.
                dist_dect_SI2_0 = math.dist(centroid_det_izq_0, centroid_box_izq_2) #
Se obtiene la distancia entre el centroide de la primera detección en el carril izquierdo
y la última boundingbox proporcionada por el segundo seguidor izquierdo.
                dist_dect_SI2_1 = math.dist(centroid_det_izq_1, centroid_box_izq_2) #
Se obtiene la distancia entre el centroide de la segunda detección en el carril izquierdo
y la última boundingbox proporcionada por el segundo seguidor izquierdo.
                dist_dect_SI2_2 = math.dist(centroid_det_izq_2, centroid_box_izq_2) #
Se obtiene la distancia entre el centroide de la tercera detección en el carril izquierdo
y la última boundingbox proporcionada por el segundo seguidor izquierdo.
        if box_der_1_anterior[2] != 0 or box_der_2_anterior[2] != 0: # Si la coordenada
Y de la última boundingbox proporcionada por el primer seguidor derecho es distinta de
cero o si la coordenada Y de la última boundingbox proporcionada por el segundo seguidor
derecho es distinta de cero...
            if box_der_1_anterior[2] != 0: # Si la coordenada Y de la última boundingbox
proporcionada por el primer seguidor derecho es distinta de cero...
                centroid_box_der_1 = [((box_der_1_anterior[0] + box_der_1_anterior[2])
* 0.5), ((box_der_1_anterior[1] + box_der_1_anterior[3]) * 0.5)] # Se obtiene el
centroide de la última boundingbox proporcionada por el primer seguidor derecho.
                dist_dect_SD1_0 = math.dist(centroid_det_der_0, centroid_box_der_1) #
Se obtiene la distancia entre el centroide de la primera detección en el carril derecho
y la última boundingbox proporcionada por el primer seguidor derecho.
                dist_dect_SD1_1 = math.dist(centroid_det_der_1, centroid_box_der_1) #
Se obtiene la distancia entre el centroide de la segunda detección en el carril derecho
y la última boundingbox proporcionada por el primer seguidor derecho.
                dist_dect_SD1_2 = math.dist(centroid_det_der_2, centroid_box_der_1) #
Se obtiene la distancia entre el centroide de la tercera detección en el carril derecho
y la última boundingbox proporcionada por el primer seguidor derecho.
            if box_der_2_anterior[2] != 0: # Si la coordenada Y de la última boundingbox
proporcionada por el segundo seguidor derecho es distinta de cero...
                centroid_box_der_2 = [((box_der_2_anterior[0] + box_der_2_anterior[2])
* 0.5), ((box_der_2_anterior[1] + box_der_2_anterior[3]) * 0.5)] # Se obtiene el
centroide de la última boundingbox proporcionada por el segundo seguidor derecho.
```

```
                dist_dect_SD2_0 = math.dist(centroid_det_der_0, centroid_box_der_2) #
Se obtiene la distancia entre el centroide de la primera detección en el carril derecho
y la última boundingbox proporcionada por el segundo seguidor derecho.
                dist_dect_SD2_1 = math.dist(centroid_det_der_1, centroid_box_der_2) #
Se obtiene la distancia entre el centroide de la segunda detección en el carril derecho
y la última boundingbox proporcionada por el segundo seguidor derecho.
                dist_dect_SD2_2 = math.dist(centroid_det_der_2, centroid_box_der_2) #
Se obtiene la distancia entre el centroide de la tercera detección en el carril derecho
y la última boundingbox proporcionada por el segundo seguidor derecho.

    vect_distance_1_2 = [dist_dect_SI1, dist_dect_SI2, dist_dect_SD1, dist_dect_SD2,
dist_dect_izq,  dist_dect_der,  dist_dect_0_SI1,  dist_dect_1_SI1,  dist_dect_0_SI2,
dist_dect_1_SI2, dist_dect_0_SD1, dist_dect_1_SD1, dist_dect_0_SD2, dist_dect_1_SD2] #
Montaje de la primera lista para su retorno.
         vect_distance_3    =    [dist_dect_SI1_0,dist_dect_SI1_1,    dist_dect_SI1_2,
dist_dect_SI2_0,  dist_dect_SI2_1,  dist_dect_SI2_2,  dist_dect_SD1_0,dist_dect_SD1_1,
dist_dect_SD1_2, dist_dect_SD2_0, dist_dect_SD2_1, dist_dect_SD2_2] # Montaje de la
segunda lista para su retorno.
    return(vect_distance_1_2, vect_distance_3)

#Bucle "infinito":
while (video_capture.isOpened()): # Mientras existan frames en el vídeo a tratar...
    ret, frame = video_capture.read() # Captura de un frame del vídeo a tratar.
    if not ret: # Si no se ha capturado un frame del vídeo a tratar...
         print('No hay captura de frame. Fin ejecución') # Impresión en consola de la
finalización de la ejecución.
        break # Ruptura del bucle "infinito".
    else: # Si se ha capturado un frame del vídeo a tratar...
# 1 PASO. REESCALADO EN 1/2 COMO OPTIMIZACIÓN (DE 1920x1080 A 960x540)
         scale_percent = 50 # Porcentaje de reescalado, se mantiene la relación de
aspecto.
        width = int(frame.shape[1] * scale_percent / 100) # Definición del nuevo ancho
del frame reescalado.
        height = int(frame.shape[0] * scale_percent / 100) # Definición del nuevo alto
del frame reescalado.
       dim = (width, height) # Definición de una tupla con los parámetros de reescalado.
        frame_resized = cv2.resize(frame, dim, interpolation = cv2.INTER_AREA) # Frame
reescalado.
# 2 PASO. OBTENCIÓN DE LAS ROI ASOCIADAS A LOS CARRILES (ROI = Region Of Interest).
         roi_lane_left_end = frame_resized[10:530, 190:490] # Definición de la ROI
asociada al carril izquierdo.
         roi_lane_right_end = frame_resized[10:530, 480:780] # Definición de la ROI
asociada al carril derecho.
# 3 PASO. APLICACIÓN DE LA SUSTRACCIÓN DE FONDO A CADA ROI.
        subtraction_left = mask_MOG2_left.apply(roi_lane_left_end, learningRate = -1)
# Sustracción de fondo aplicada a la ROI del carril izquierdo.
        subtraction_right = mask_MOG2_right.apply(roi_lane_right_end, learningRate = -
1) # Sustracción de fondo aplicada a la ROI del carril derecho.
# 4 PASO. APLICACIÓN DE OPERACIONES MORFOLÓGICAS TRAS LA SUSTRACCIÓN DE FONDO.
```

```
        kernel_open = numpy.ones((3,3), numpy.uint8) # Definición del primer núcleo empleado en la operación de apertura.
        kernel_close = numpy.ones((5,5), numpy.uint8) # Definición del segundo núcleo empleado en la operación de cierre.
        subtraction_left_open = cv2.morphologyEx(subtraction_left, cv2.MORPH_OPEN, kernel_open, iterations = 2) # Aplicación de la operación de apertura a la ROI del carril izquierdo.
        subtraction_right_open = cv2.morphologyEx(subtraction_right, cv2.MORPH_OPEN, kernel_open, iterations = 2) # Aplicación de la operación de apertura a la ROI del carril derecho.
        subtraction_left_close = cv2.morphologyEx(subtraction_left_open, cv2.MORPH_CLOSE, kernel_close, iterations = 2) # Aplicación de la operación de cierre a la ROI del carril izquierdo.
        subtraction_right_close = cv2.morphologyEx(subtraction_right_open, cv2.MORPH_CLOSE, kernel_close, iterations = 2) # Aplicación de la operación de cierre a la ROI del carril derecho.
# 5 PASO. OBTENCIÓN DE FPS DEL PROCESO (FPS = Frames Per Second)
        new_frame_time = time.time() # Obtención del tiempo de ejecución actual.
        fps = (1/(new_frame_time-prev_frame_time)) # Cálculo de los FPS.
        prev_frame_time = new_frame_time # Guardado del tiempo de ejecución actual.
        count_fps.append(int(fps)) # Guardado de los FPS actuales en una lista.
        cv2.putText(frame_resized, "FPS: " + (str(int(fps))), (10, 30),cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, lineType= cv2.LINE_AA) # Representación del número de FPS actual.
# 6 PASO. TRATAMIENTO DEL FRAME (SEGUIMIENTO O DETECCIÓN)
        if (seguimiento_izq_on or seguimiento_der_on): # Si existe detección en el carril izquierdo o en el carril derecho...
            tracking(roi_lane_left_end[:], roi_lane_right_end[:]) # Llamada a la función de seguimiento.
        else: # Si no existe detección en el carril izquierdo o en el carril derecho...
            detect(subtraction_left_close[:], subtraction_right_close[:]) # Llamada a la función de detección.

        cv2.imshow('N-433 KM 89 CTRA. CIRCUNVALACION SEVILLA-HUELVA D HUELVA', frame_resized) # Representación visual del vídeo tratado.

        if cv2.waitKey(5) & 0xFF == ord('q'): # Si se cumple la condición de salida en el tiempo determinado...
            break # Ruptura del bucle "infinito".

video_capture.release() # Liberación los recursos hardware y software empleados durante la ejecución.
cv2.destroyAllWindows() # Cierre de la representación visual.
print('Fin ejecución') # Impresión en consola de la finalización de la ejecución.
print('Detecciones carril izquierdo: '+ str((contador_vehi_izq -1))) # Impresión en consola de las detecciones realizadas en el carril izquierdo.
print('Detecciones carril derecho: '+ str((contador_vehi_der - 1))) # Impresión en consola de las detecciones realizadas en el carril derecho.
```

```
sum_fps = sum(count_fps) # Obtención del número de FPS tratados durante toda la ejecución.
if (len(count_fps)) != 0: # Si el número de FPS tratados es distintos de cero...
    med_fps = (sum_fps / (len(count_fps))) # Obtención de la media de FPS tratados a lo largo de toda la ejecución.
    max_fps = max(count_fps) # Obtención del número máximo de FPS tratados a lo largo de toda la ejecución.
     new_list = list(set(count_fps)) # Obtención de una nueva lista sin elementos repetidos.
    pos_zero_fps = new_list.index(0) # Determinación de elementos nulos. No se pueden dar FPS nulos.
    del(count_fps[pos_zero_fps]) # Eliminación del elemento nulo.
    min_fps = min(count_fps) # Obtención del número mínimo de FPS tratados a lo largo de toda la ejecución.

     with open ('DATA_DETECTION_TFG_JSBM.txt', 'a', encoding = 'utf-8') as file: # Apertura de fichero con nombre DATA_DETECTION_TFG_JSBM.txt en la variable file y en modo append (añadido).
            file.write('*-----------------------------------------------------------------*\n\r') # Escritura de una línea separatoria en el reporte de detecciones.
            file.write('*-----------------------------------------------------------------*\n\r') # Escritura de una línea separatoria en el reporte de detecciones.
        file.write('FINAL REPORT OF TRACKING: \n\r') # Escritura del encabezado final del reporte de detecciones.
         file.write('Original frames: %.3f\r\n' % round(fps_original, 3)) # Escritura del número de frames originales del vídeo tratado en el reporte de detecciones.
       file.write('Left lane detections : %d\r\n' % (contador_vehi_izq - 1)) # Escritura del número de detecciones realizadas en el carril izquierdo en el reporte de detecciones.
          file.write('Right lane detections : %d\r\n' % (contador_vehi_der - 1)) # Escritura del número de detecciones realizadas en el carril derecho en el reporte de detecciones.
         file.write('FPS process: %d\r\n' % (sum_fps + 1)) # Escritura del número de frames trarados en el reporte de detecciones.
         file.write('Mean FPS process : %.3f\r\n' % round(med_fps,3)) # Escritura del número medio de FPS en toda la ejecución en el reporte de detecciones.
         file.write('Highest FPS process : %.3f\r\n' % max_fps) # Escritura del número máximo de FPS en toda la ejecución en el reporte de detecciones.
         file.write('Minimum FPS process : %.3f\r\n' % min_fps) # Escritura del número mínimo de FPS en toda la ejecución en el reporte de detecciones.
```