2 de diciembre del 2019

Procesamiento de video

Profesor: Francisco Javier Hernández López

Proyecto: Implementación de detección y eliminación de sombras en movimiento para secuencias de tráfico

Integrantes:

Josue Moreno Moo

Antonio Maldonado Pinzón

Procesamiento de video

Introducción

Objetivo

Implementación y verificación del funcionamiento del método de eliminación de sombras descrito en el artículo: *Moving Shadow Detection and Removal for Traffic Sequences,* en el lenguaje de programación Python con las librerías de OpenCV y Numpy.

Herramientas

- Equipos de cómputo.
- Lenguaje de programación Python versión 3.7
- Librerías de Python: OpenCV, NumPy.
- Videos de secuencias de tráfico con sombras presentes.

Implementación

Luego, a través de un ciclo while, vamos recorriendo todos los frames (o imágenes) existentes en el video (ret es una variable que nos sirve para saber si hemos llegado al último frame del video):

```
while(1):
    ret, frame = cap.read()

    # Si hemos llegado al final del vídeo salimos
    if not ret:
        break
```

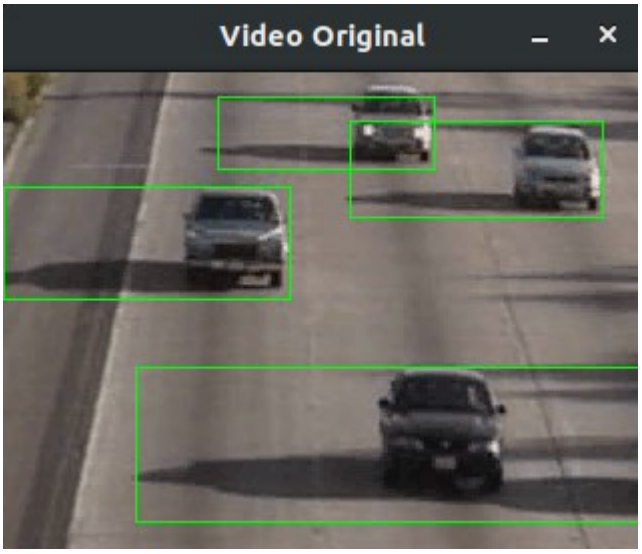
Después se reserva memoria para las imágenes que vamos a calcular

```
a,b,c=frame.shape #Obtenemos las d

#Creamos las matrices en que vamos
genes a calcular
fra2=np.zeros((a, b,c), np.uint8)
I1=np.zeros((a, b), np.uint8)
I2=np.zeros((a, b), np.uint8)
IFEt=np.zeros((a,b), np.uint8)
FRt=np.zeros((a,b), np.uint8)
IFt=np.zeros((a,b), np.uint8)
ISt=np.zeros((a, b), np.uint8)
HRt=np.zeros((a,b), np.uint8)
HRt=np.zeros((a,b), np.uint8)
VRt=np.zeros((a,b), np.uint8)
REt=np.zeros((a,b), np.uint8)
HREt=np.zeros((a,b), np.uint8)
VREt=np.zeros((a,b), np.uint8)
HFRt=np.zeros((a,b), np.uint8)
VFRt=np.zeros((a,b), np.uint8)
HREMt=np.zeros((a,b), np.uint8)
VREMt=np.zeros((a,b), np.uint8)
```

Procesamiento de video

Como primera instancia para empezar la implementación de este método, necesitamos videos de referencia en los cuales aparezcan autos en movimiento y que generen sombras en su trayecto. Para este caso, tenemos un video de referencia que cumple con las características necesarias para implementar el método y que nos servirá para obtener los resultados de los pasos intermedios hasta llegar al resultado final.

Primero se utilizó componentes conectados para detectar los objetos y dibujar su caja envolvente en la cual en esas cajas se aplicará el algoritmo para detectar sombras y no en toda la imagen.

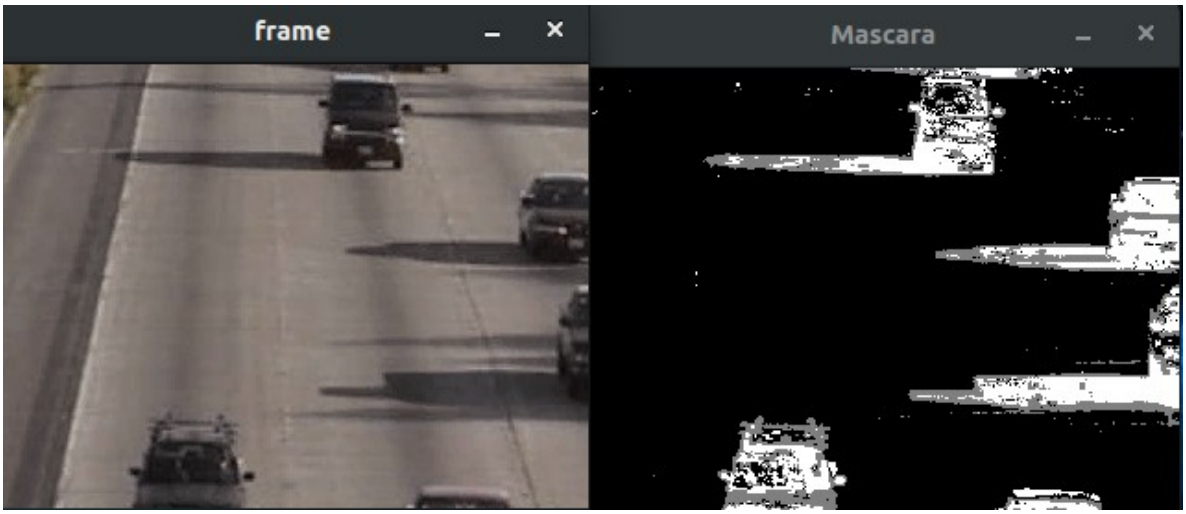


Para esta implementación en particular, decidimos utilizar el sustractor de fondo KNN, en lugar de MOG2, debido a que nos obtenía una mejor máscara del foreground y contenía menor ruido.

Código:

```
fgbg = cv2.createBackgroundSubtractorKNN()
fgmask = fgbg.apply(frame)
```

Imagen:



Esta fue la salida del método KNN. Lo que requeríamos es que la mascara estuviera binarizada y que no tuviera ruido.

Por lo que se tuvo que implementar una función limpiar(I1,I2) la cual hace uso de la función de la librería opencv llamada componentsWihStats. Utilizamos esa función para eliminar los pequeños blobs, es decir el ruido, esto es lo que resultó luego de aplicar la función limpiar.

Código:

```
def limpiar(Iin,Iout):
   objects_stats = [] #Para almacenar los stats de los objetos grandes
   areas = [] #Para almacenar unicamente areas de los objetos
   nlabels, labels, stats, centroids = cv2.connectedComponentsWithStats(Iin, 4, cv2.CV_32S)
   flag=0
   for stat2 in stats:
     if stat2[4] < 100 and flag > 0: #Para no dibujar el rectangulo del background y elimina
       objects_stats.append(stat2)
       areas.append(stat2[4])
       Iout[stat2[1]:stat2[1]+stat2[3], stat2[0]:stat2[0]+stat2[2]]=0
     flag = flag + 1
```

Lo que hace la función es eliminar los blobs que contengan menos de 100 pixeles, esto es para eliminar el ruido que por lo usual tienen pocos pixeles.

Imagen luego de aplicar la función limpiar

Procesamiento de video

Aún falta binarizar la imagen y rellenar los huecos internos que tienen los objetos, por lo que hay que aplicar un threshold y aplicar un suavizado para rellenar los huecos.

Código:

```
Ct = cv2.medianBlur(fgmask,7)
Ct = cv2.blur(Ct,(3,3))
th, Ct = cv2.threshold(Ct,30, 255, cv2.THRESH_BINARY)
```

Imagen:

Esta es la imagen Ct la cual es la entrada para nuestro método. Aunque aún hay huecos en los objetos no es tanto como antes.

Luego procedemos a calcular el apartado ECt con la imagen binaria Ct y el B1.

$$EC_t = C_t - (C_t \ominus B_1)$$

B1: Kernel 3x3 con 1’s en toda la matriz.

Para ello, utilizamos la imagen Ct y le aplicamos una erosión con el kernel B1. Luego a la misma imagen Ct le restamos esta operación morfológica, obteniendo así la imagen ECt

Código:

```
aux = cv2.erode(Ct, kernel, iterations = 1)
ECt = Ct - aux
th, ECt = cv2.threshold(ECt, 180, 255, cv2.THRESH_BINARY)#
```

Resultado:

A continuación, obtenemos el apartado DEt a través de una dilatación entre ECt y el kernel 3x3:

$$DE_t = EC_t \oplus B_2$$

En el artículo se recomendaba usar un elemento estructural de tamaño5x5 a 9x9 pero nosotros decidimos usar uno de 3x3 porque haciendo pruebas es el que funcionó mejor.

Código:

```
DEt = cv2.dilate(ECt, kernel, iterations = 1)
```

Resultado:

Procesamiento de video

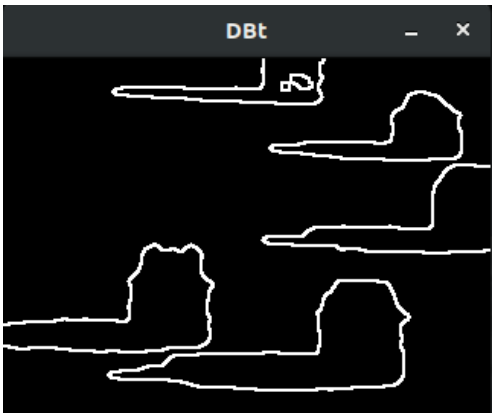
Para el siguiente apartado, DBt, lo calculamos a partir de un AND lógico entre la imagen DEt y Ct como lo establece el artículo:

$$DB_t(x,y) = \begin{cases} 1, & DE_t(x,y) = 1 \text{ and } C_t(x,y) = 1 \\ 0, & \text{otherwise} \end{cases}$$

Código:

```
DBt = cv2.bitwise_and(DEt, Ct)
```

Imagen:



Antes de calcular el apartado IEt, calculamos los bordes de la imagen Ct con la función Canny definida en la librería de openCV y la guardamos en la variable Et:

Ahora pasamos a calcular IEt, y se calcula como:

$$IE_t(x,y) = \begin{cases} 1, & E_t(x,y) = 1 \text{ and } DB_t(x,y) = 1 \\ 0, & \text{otherwise} \end{cases}$$

DBt ya lo calculamos, por lo que solo falta calcular Et y éste se calcula con los bordes Canny sobre el frame actual.

Código:

```
Et = Bordes(frame)
```

Imagen:

Procesamiento de video

El artículo dice que para calcular Iet es hacer una AND entre Et y DBt, sin embargo lo que se quiere calcular son los bordes del objeto pero sin el contorno por lo que al hacer la AND entre DBt y Et practicamente solo va a quedar Et que no es lo que queremos. Tomando en cuenta lo anterior se propuso otra forma de calcular IEt, la cual es:

IEt = Et – DBt

Código:

```
IEt = Et - DBt
th, IEt = cv2.threshold(IEt, 10, 255, cv2.THRESH_BINARY)#
```

Imagen:

Ahora ya que tenemos IEt procedemos a calcular HRt y VRt

Antes de calcular HRt y VRt hay que calcular IHt y IVt

Para calcular IHt y IVt se implementó una funcion para aplicar la operación horizontal y otra funcion para aplicar la operación vertical.

Las operaciones solo se aplica para la caja envolvente del blob.

Código:

```
def Horizontal(I, I1):
    vert, hor = I.shape
    for i in range(0, vert):
        bandera = 0
        for j in range(0, hor):
            if(I[i,j] != 0):
                if(bandera == 0):
                    y_inicial = j
                    y_final = j
                    bandera = 1
                else:
                    y_final = j
        if(bandera == 1):
            I1[i,y_inicial:y_final] = 255
```

Lo que hace la función Horizontal es recibir una caja envolvente de un blob y va recorriendo la caja fila por fila tomando el primer pixel de la fila y el ultimo pixel de la fila y asigna el valor de 255 a todos los pixeles que estén en medio del primer pixel y el último pixel.

```
def Vertical(I, I2):
    vert, hor = I.shape
    for i in range(0, hor):
        bandera = 0
        for j in range(0, vert):
            if(I[j,i] != 0):
                if(bandera == 0):
                    x_inicial = j
                    x_final = j
                    bandera = 1
                else:
                    x_final = j
        if(bandera == 1):
            I2[x_inicial:x_final,i] = 255
```

La función vertical es muy parecida a la función horizontal la diferencia es que en esta función asigna el valor de 255 al primer pixel de la columna y el último pixel de la columna.

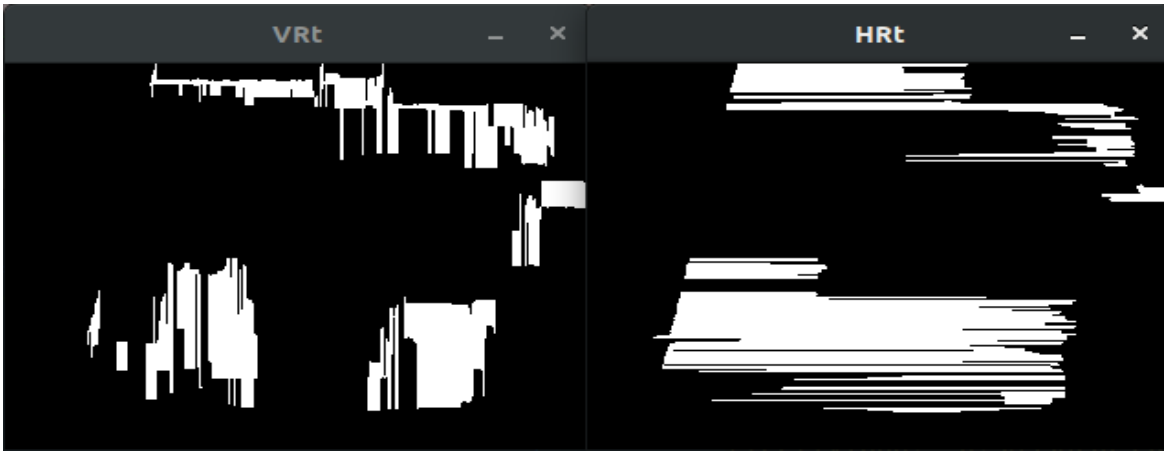
Luego de que se calculan IHt y VRt se utiliza nuestra funcion limpiar() la cual elimina los pequeños blobs y el resultado HRt y VRt respectivamente.

Código:

Procesamiento de video

```
connectivity = 8
nlabels, labels, stats, centroids = cv2.connectedComponentsWithStats(Ct, connectivity, cv2.CV_32S) #Calculamos componentes conecte
t (mascara del objeto)
objects_stats = []
areas = []
flag = 0
#Recorremos todos los blobs
for stat in stats:
    if stat[4] > 105 and flag > 0: #Los blobs que tengan mas de 155 pixeles
        objects_stats.append(stat)
        areas.append(stat[4])
        #Aplicamos operacion horizontal a IEt para generar HRt
        Horizontal(IEt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]], HRt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]])
        #Aplicamos operacion vertical a IEt para generar VRt
        Vertical(IEt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]], VRt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]])
        #Dibujamos la caja envolvente de los blobs sobre el frame
        cv2.rectangle(frame,(stat[0],stat[1]),(stat[0]+stat[2],stat[1]+stat[3]),(0,255,0),1)
    flag = flag + 1

#Se eliminan los blobs pequenos
limpiar(HRt,HRt)
limpiar(VRt,VRt)
```

Imagenes:



Ahora procedemos a calcular Ret, el artículo dice que:

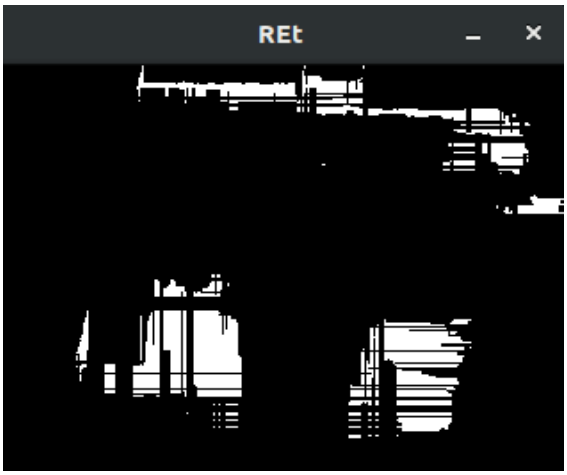
$$RE_t(x,y) = \begin{cases} 1, & HR_t(x,y) = 1 \text{ and} \\ & VR_t(x,y) = 1 \text{ and} \\ & E_t(x,y) = 1 \\ 0, & \text{otherwise} \end{cases}$$

Pero no tiene sentido que se haga una AND con Et por lo que solo se realizó la AND entre HRt y Vrt.

Código:

```
REt = cv2.bitwise_and(HRt, VRt)
```

Imagen:



Ahora se procede a calcular HFRt y VFRt, para ello primero se calulan las imágenes HREt y VREt esto se hace aplicando las operaciones horizontales y verticales respectivamente.

Código:

Procesamiento de video

```
nlabels, labels, stats, centroids = cv2.connectedComponentsWithStats(Ct, connectivity, cv2.CV_32S)
objects_stats = []
areas = []
flag = 0

for stat in stats:
    if stat[4] > 155 and flag > 0:
        objects_stats.append(stat)
        areas.append(stat[4])
        #Aplicamos la operacion Horizontal a REt para generar HREt
        Horizontal(REt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]], HREt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]])
        #Aplicamos la operacion Vertical a REt para generar VREt
        Vertical(REt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]], VREt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]])
    flag = flag + 1
```

Primero se calculan los componentes conectados para encontrar las dimensiones de las cajas de los blobs y luego en un ciclo for se recorren todos los blobs mayores a 105 pixeles y se les calcula las operaciones horizontal y vertical para generar HREt y VREt.

Imagen:

$$T_d = \alpha \cdot \min\{\min_{i=1,2,\cdots,M_1}(HL^i), \min_{j=1,2,\cdots,N_1}(VC^j)\}$$

$$\alpha \in (0, 1]$$

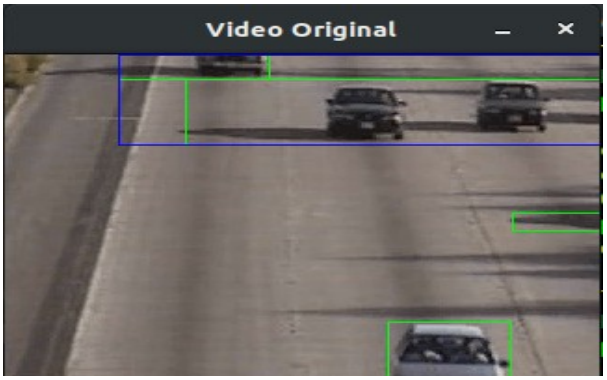
Ahora se tiene que etiquetar los componetnes conectados en HREt y VREt, para esto en el artículo propone esto:

donde HL es el número de filas del blob en HREt

y VC es el número de columnas de blob en VREt

Pero nosotros decidimos hacer algo diferente. Lo que propusimos es calcular la distanica entre las cajas y si se cumple un cierto umbral se unirán las cajas envolventes de los 2 blobs. Lo anterior se hace con la funcion unionCajas, la funcion es un poco larga y hace uso de otras funciones, básicamente lo que hace la función unionCajas es calcular la distancia entre blobs para diferentes casos y si se cumple el umbral entonces se unirán las cajas envolventes.

Lo que hicimos es dibujar un rectángulo azul el cual indica que se unieron al menos 2 cajas envolventes de los blobs



Y en esos rectángulos azules aplicamos las operaciones horizontales y verticales y luego se aplica una OR para calcular HREMt y VREMt.

Código:

```
resuk = unionCajas(objects_stats, 0)

for stat in resuk:
    cv2.rectangle(frame,(stat[0],stat[1]),(stat[2],stat[3]),(255,0,0),1) #Dibujamos a las cajas
    #Aplicamos la operacion horizontal a HREt para generar HREMt
    Horizontal(HREt[stat[1]:stat[3], stat[0]:stat[2]], HREMt[stat[1]:stat[3], stat[0]:stat[2]])
    #Aplicamos la operacion horizontal a VREt para generar VREMt
    Vertical(VREt[stat[1]:stat[3], stat[0]:stat[2]], VREMt[stat[1]:stat[3], stat[0]:stat[2]])

#Aplicamos una OR para unir los blobs
HREMt = cv2.bitwise_or(HREt, HREMt)
VREMt = cv2.bitwise_or(VREt, VREMt)
```

Imagen:

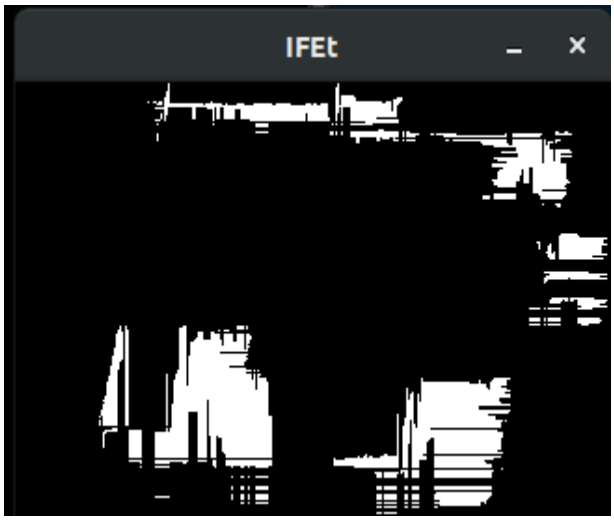
Procesamiento de video

Ya que calculamos estas 2 imágenes, ahora calculamos IFEt para lo cual se hizo una AND entre HREMt y VREMt ya que se requería separar los blobs que se hayan unido en el paso anterior.

Código:

```
IFEt=cv2.bitwise_and(HREMt,VREMt)
```

Imagen:



En este proyecto se consideró que en una máscara de objeto solo hay un objeto. Para IFEkt se consideró siempre k=1.

Ahora calculamos FRt, para eso primero necesitamos calcular HFRt y VFRt que no son más que aplicar las operaciones horizontal y vertical a IFEt.

Código:

Procesamiento de video

```
nlabels, labels, stats, centroids = cv2.connectedComponentsWithStats(IFEt, connectivity, cv2.CV_32S)
objects_stats = []
areas = []
flag = 0

for stat in stats:
    if stat[4] > 100 and flag > 0: #Si el blob es mayor de 100 pixeles
        objects_stats.append(stat)
        areas.append(stat[4])
        #Aplicamos la operacion horizontal a IFEt para generar HFRt
        Horizontal(IFEt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]], HFRt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]])
        #Aplicamos la operacion horizontal a IFEt para generar VFRt
        Vertical(IFEt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]], VFRt[stat[1]:stat[1]+stat[3], stat[0]:stat[0]+stat[2]])
    flag = flag + 1
```

Imagen:

Ya que tenemos VFRt y HFRt ya podemos calcular Frt:

$$FR_t^k(x,y) = \begin{cases} 1, & HFR_t^k(x,y) = 1 \text{ or} \\ & VFR_t^k(x,y) = 1 \\ 0, & \text{otherwise} \end{cases}$$

Procesamiento de video

Código:

```
FRt=cv2.bitwise_or(HFRt,VFRt)
```

Imagen:

Ahora calculamos IFt, y se calcula:

$$IF_t(x,y) = \begin{cases} 1, & FR_t^k(x,y) = 1,\ \forall k = 1,2,\cdots,q \\ 0, & \text{otherwise} \end{cases}$$

Ya que nosotros tomamos k=1 IFt es igual a Frt para nuestro caso.

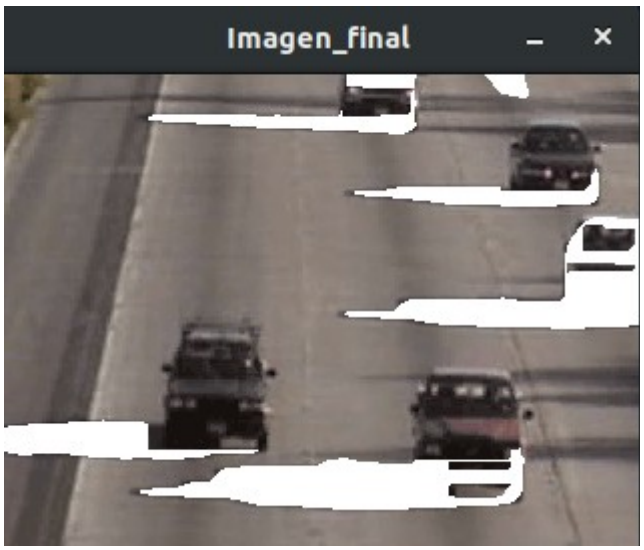
Llegando a la parte final del método, procedemos a calcular la imagen ISt, la cual surge a partir de una diferencia entre Ct, DBt y FRt (todas ellas calculadas previamente). Luego, aplicamos un cierto umbral y la función que definimos para eliminación de ruido. Convertimos la imagen ISt a color y obtenemos la imagen final del método con una operación OR entre ISt y el frame original del video.

Código:

```
ISt=Ct - DBt-FRt

#Aplicamos un umbral a ISt
th, ISt = cv2.threshold(ISt, 50, 255, cv2.THRESH_BINARY)

#Eliminamos los pequenos blobs
limpiar(ISt,ISt)

#Convertimos ISt a color
ISt=cv2.cvtColor(ISt,cv2.COLOR_GRAY2BGR)

#Le aplicamos las sombras calculadas al frame
Imagen_final=cv2.bitwise_or(ISt,fra2)
```

Procesamiento de video

Imagen:



Curvas ROC

A continuación se va a evaluar el algoritmo con las siguiente métricas:

- Positive Detection Rate (PDR):

$$PDR = \frac{TP}{TP + FN}$$

- Negative Detection Rate (NDR):

$$NDR = \frac{TN}{TN + FP}$$

- Presición:

$$P = \frac{TP}{TP + FP}$$

- Medida de efectividad:

$$F = \frac{2 * PDR * P}{PDR + P}$$

donde:

- True Positives (TP) → Pixeles bien clasificados como FG
- True Negatives (TN) → Pixeles bien clasificados como BG
- False Positives (FP) → Pixeles mal clasificados como FG
- False Negatives (FN) → Pixeles mal clasificados como BG

Para evaluar el Video ‘highwayI_raw.avi’ solo se tienen 8 imagenes del GT. Estos son los 8 GT:

Procesamiento de video

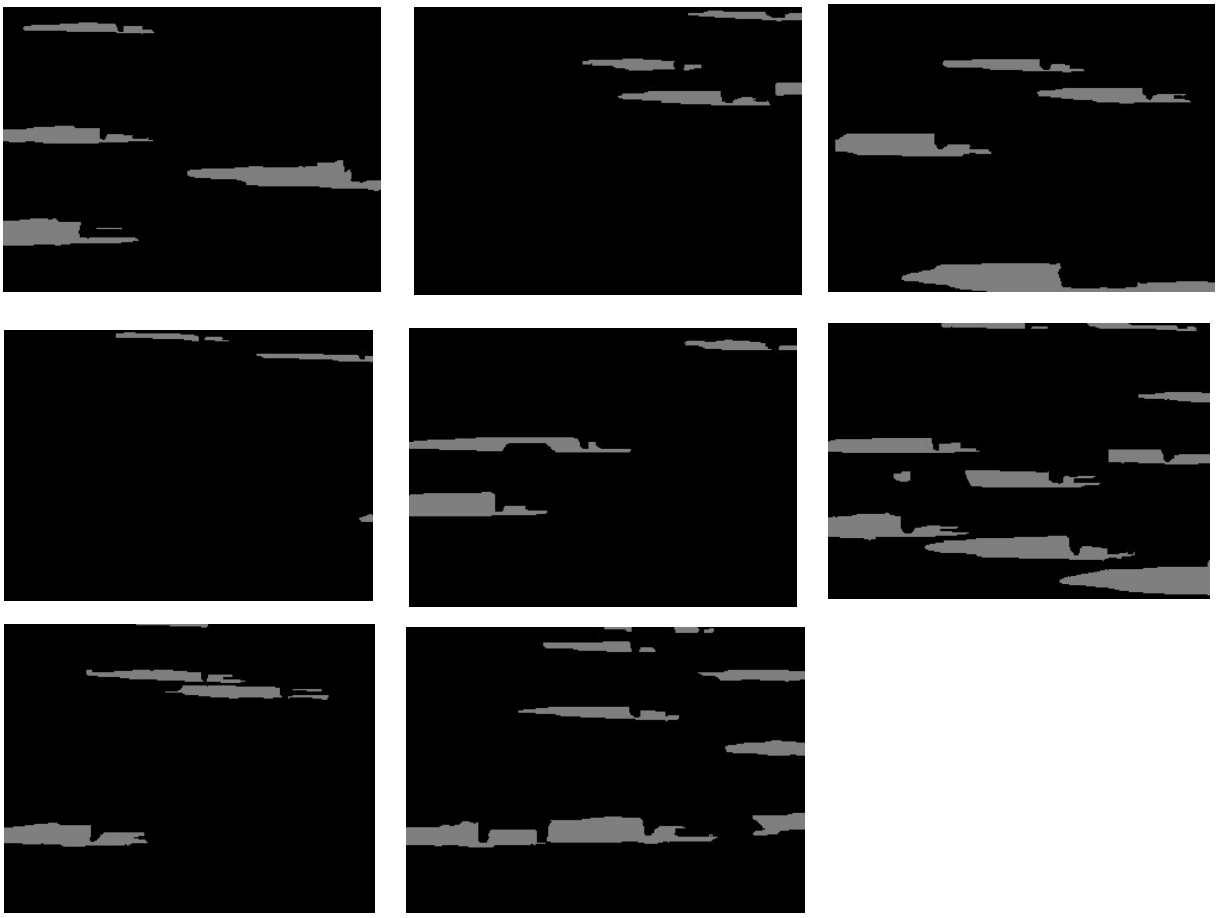
Ya que solo queremos evaluar la sombra hay que eliminar el auto y quedarnos con la sombra, para esto se implementó esta función:

```
def sombras(I):
    vert,hor=I.shape
    for i in range(vert):
        for j in range (hor):
            if(I[i,j]>=180):
                I[i,j]=0
```

Lo que hace la función sombras es eliminar el auto y se queda con la sombra, simplemente aplicando un umbral.

Resultados:

Procesamiento de video



Ahora ya se puede calcular las métricas, se implementaron las funciones para calcular las métricas:

```
def TrueNegatives(I,GT):
    q,w=I.shape
    acumulador=0
    for i in range(q):
        for j in range(w):
            if(I[i,j]==0 and GT[i,j]==0):
                acumulador+=1
    return   acumulador
```

```
def FalseNegatives(I,GT):
    q,w=I.shape
    acumulador=0
    for i in range(q):
        for j in range(w):
            if(I[i,j]==0 and GT[i,j]!=0):
                acumulador+=1
    return   acumulador
```

```
def FalsePositives(I,GT):
    q,w=I.shape
    acumulador=0
    for i in range(q):
        for j in range(w):
            if(I[i,j]!=0 and GT[i,j]==0):
                acumulador+=1
    return    acumulador
```

```
def TruePositives(I,GT):
    q,w=I.shape
    acumulador=0
    for i in range(q):
        for j in range(w):
            if(I[i,j]!=0 and GT[i,j]!=0):
                acumulador+=1
    return    acumulador
```

Ya que implementamos las métricas, ahora hay que utilizarlas para evaluar el algoritmo, los 8 GT que tenemos son los frames 45,65,85,105,125,145,165,185. Por lo que solo en esos frames vamos a evaluar el algoritmo. Así que se definió esto:

```
cont+=1

if(cont==45):
    TP+=TruePositives(ISt,h45)
    TN+=TrueNegatives(ISt,h45)
    FP+=FalsePositives(ISt,h45)
    FN+=FalseNegatives(ISt,h45)
     cv2.imshow('ISt45',ISt)

elif(cont==65):
    TP+=TruePositives(ISt,h65)
    TN+=TrueNegatives(ISt,h65)
    FP+=FalsePositives(ISt,h65)
    FN+=FalseNegatives(ISt,h65)
     cv2.imshow('ISt65',ISt)

elif(cont==85):
    TP+=TruePositives(ISt,h85)
    TN+=TrueNegatives(ISt,h85)
    FP+=FalsePositives(ISt,h85)
    FN+=FalseNegatives(ISt,h85)
     cv2.imshow('ISt85',ISt)

elif(cont==105):
    TP+=TruePositives(ISt,h105)
    TN+=TrueNegatives(ISt,h105)
    FP+=FalsePositives(ISt,h105)
    FN+=FalseNegatives(ISt,h105)
     cv2.imshow('ISt105',ISt)
```

```
elif(cont==125):
  TP+=TruePositives(ISt,h125)
  TN+=TrueNegatives(ISt,h125)
  FP+=FalsePositives(ISt,h125)
  FN+=FalseNegatives(ISt,h125)
   cv2.imshow('ISt125',ISt)

elif(cont==145):
  TP+=TruePositives(ISt,h145)
  TN+=TrueNegatives(ISt,h145)
  FP+=FalsePositives(ISt,h145)
  FN+=FalseNegatives(ISt,h145)
   cv2.imshow('ISt145',ISt)

elif(cont==165):
  TP+=TruePositives(ISt,h165)
  TN+=TrueNegatives(ISt,h165)
  FP+=FalsePositives(ISt,h165)
  FN+=FalseNegatives(ISt,h165)
   cv2.imshow('ISt165',ISt)

elif(cont==185):
  TP+=TruePositives(ISt,h185)
  TN+=TrueNegatives(ISt,h185)
  FP+=FalsePositives(ISt,h185)
  FN+=FalseNegatives(ISt,h185)
   cv2.imshow('ISt185',ISt)
```

A las funciones se les manda la imagen ISt que es lo que arroja el algoritmo, es decir la sombra del auto y también se le manda su correspondiente GT para evaluar ese frame,

Ya que hayamos calculado TP,TN,FP,FN, lo que sigue es calcular F,PDR,NDR,P.

Primero sacamos el promedio de los 8 GT:

```
TP/=8
TN/=8
FP/=8
FN/=8
```

Ahora calculamos F,PDR,NDR,P.

```
PDR=TP/(TP+FN) #Pc
NDR=TN/(TN+FP) #Ne
P=TP/(TP+FP)   #Pr
F=2*PDR*P/(PDR+P)
```

Resultados:

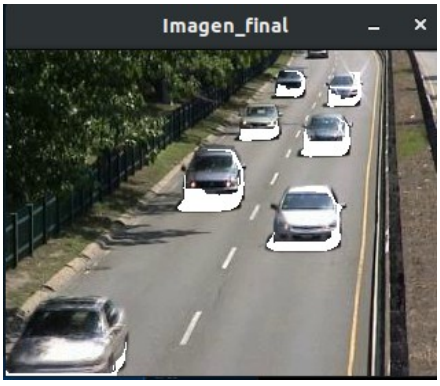
```
Efectividad:  0.6880931354867597
Tasa de deteccion positiva:  0.6909031871315816
Tasa de deteccion negativa:  0.9778851042196238
Precision:  0.6853058494165448
```

Tasa de detección positiva significa con que porcentaje se detectó la sombra y la tasa de detección negativa el porcentaje con la que se detectó el fondo. Esto quiere decir que detectamos la sombra en un 69% y el fondo en un 97.7%

Probando con otros videos

Ahora probemos el algoritmo con el video Highway.avi.

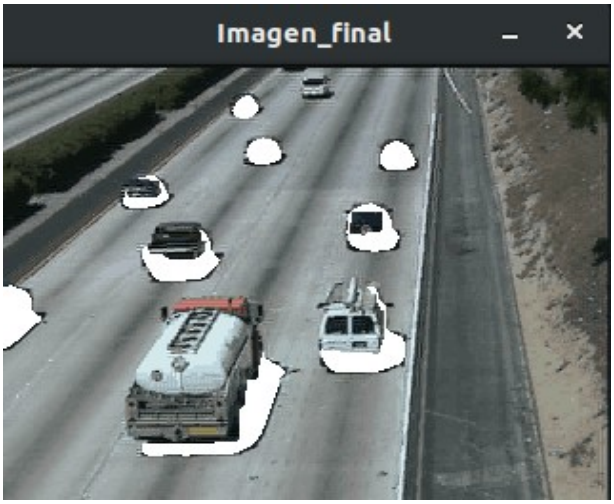
Resultados:



Aqui vemos como en este video se detecta muy bien la sombra de los autos a excepción de los autos que se encuentran lejos, esto lo decía en el artículo.

Ahora probemos el algoritmo con el video ‘HighwayII_raw.avi’

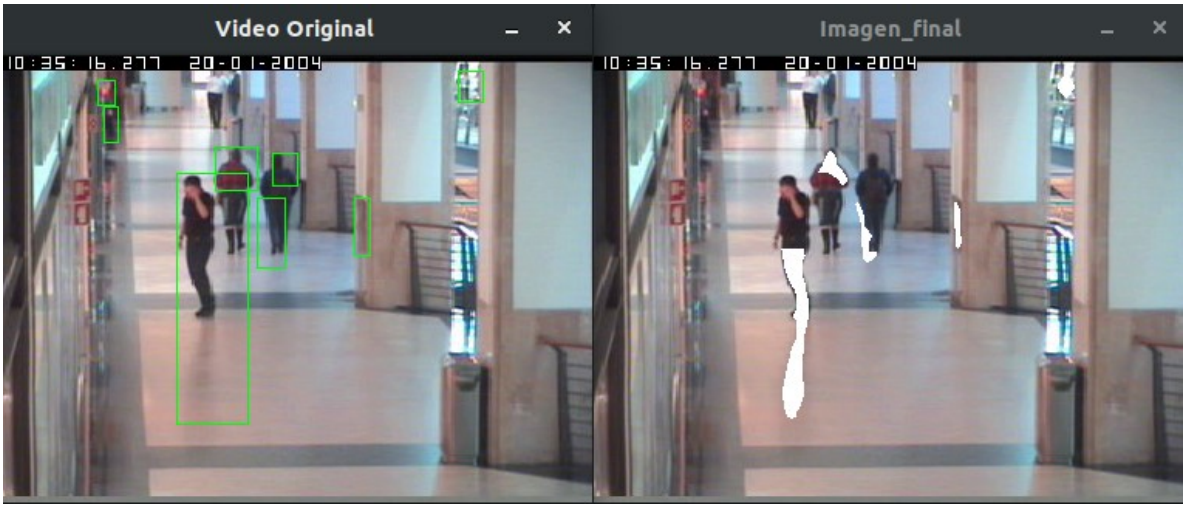
Resultado:



En este video tiene el mismo problema que el video anterior, el cual es que para los autos lejanos no detecta su s sombras y para los autos cercanos si lo hace.

Ahora probemos el algoritmo con un video que ya no tenga vehículos sino con personas.

El video se llama ‘mall.mpg’



Vemos que el algoritmo no es bueno con la detección de sombras de personas como lo dice el artículo.

Conclusiones

El algoritmo no es perfecto pero se pueden tener buenos resultados si es que se aplica todo correctamente, el algoritmo no es tan robusto y solo funciona para vehículos.

En algunas ocaciones se detectaba como sombra parte del vehículo, quizás si hubieramos probado con otros parámetros a nuestras funciones y si no nos hubieramos saltado algunos pasos el resultado sería mejor aunque con las curvas ROC verificamos que nuestros resultados fueron buenos.

El algoritmo sufría cuando había varios autos juntos y arrojaba resultados incorrectos, también cuando los autos estaban lejos, sin embargo, cuando los autos estaban separados los resultados fueron muy buenos.

Este algoritmo es muy gráfico ya que se tenían que calcular muchas imágenes para poder llegar al resultado, pero por el otro lado el algoritmo no es tan costoso computacionalmente hablando así que es una buena alternativa si no se tiene una gran máquina.