

Pontificia universidad Javeriana

Facultad de ingeniería

Departamento de electrónica

Sergio Ivan Otero Reyes  
[Sergio\\_oter@javeriana.edu.co](mailto:Sergio_oter@javeriana.edu.co)  
Aaron Santiago Pedraza Cárdenas  
[Aaronpedrza@javeriana.edu.co](mailto:Aaronpedrza@javeriana.edu.co)  
Carlos Humberto Diaz De Luque  
[Diazcarlos@javeriana.edu.co](mailto:Diazcarlos@javeriana.edu.co)



Procesamiento de imágenes y visión

Ing. Francisco Calderón

Proyecto final

“Detección de vehículos”

29/11/2020

Bogotá D.C.

## ***Introducción:***

El procesamiento de imágenes digitales es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información. Este filtrado de imágenes es el conjunto de técnicas englobadas dentro del preprocesamiento de imágenes cuyo objetivo fundamental es obtener, a partir de una imagen origen, otra final cuyo resultado sea más adecuado para una aplicación específica mejorando ciertas características de la misma que posibiliten efectuar operaciones del procesado sobre ella. Dichas aplicaciones pueden ir desde detección de bordes hasta detección de características específicas en la imagen. Haciendo así posible el reconocimiento de objetos mediante la extracción de las características de la imagen y la búsqueda de objetos basada en dichas características.

## ***Objetivo general:***

Diseñar un algoritmo de detección de vehículos mediante el uso herramientas de procesamiento de video.

Objetivos específicos:

- Realizar un conteo de los vehículos en la estación de servicio.
- Generar datos secuenciados de los vehículos en la estación.

## ***Desarrollo:***

Este proyecto esta enfocado en el diseño de un sistema que permita calcular el número de vehículos presentes dentro de una estación de gasolina, de acuerdo con lo anterior se busca implementar técnicas de procesamiento de imágenes y video que permitan brindar esta información.



Fig 1. Diagrama de bloque

En la figura 1 se observa el diagrama de bloques que se implementó para poder realizar el conteo de vehículos. Donde el bloque inicial se encarga de obtener características del video como tamaño, duración, cuadros por segundo y definición.

El bloque de aplicación de operaciones morfológicas tiene como objetivo simplificar los datos de una imagen (o bien sea video), preservando las características esenciales como la estructura y marcado de objetos, como también la extracción de esqueleto. Permitiendo así que la fijación de contornos para los vehículos sea libre de ruido existente y eliminando aspectos irrelevantes.

Una vez los contornos de los vehículos se hallan creado y dibujado, se procede a calcular los centroides que van a permitir encontrar el centro del contorno y así ir aumentando el número de vehículos presentes dentro de la estación de servicio y generar los datos del conteo.

### *Características del video:*

```
import numpy as np
import cv2
import pandas as pd

cap = cv2.VideoCapture("/Users/aaronsantiagopedrazacardenas/Desktop/procesamiento de video /proyecto/videos proyecto/77.3gp")

frames_count, fps, width, height = cap.get(cv2.CAP_PROP_FRAME_COUNT), cap.get(cv2.CAP_PROP_FPS), cap.get(
    cv2.CAP_PROP_FRAME_WIDTH), cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
width = int(width)
height = int(height)
print(frames_count, fps, width, height)

# crear un marco de datos de pandas con el número de filas de la misma longitud que el recuento de marcos
df = pd.DataFrame(index=range(int(frames_count)))
df.index.name = "Frames"

framenumber = 0 # realiza un seguimiento del cuadro actual
carscrossedup = 0 # realiza un seguimiento de los coches que cruzaron
#carscrosseddown = 0 # keeps track of cars that crossed down
carids = [] # lista en blanco para agregar identificadores de automóviles
caridscrossed = [] # lista en blanco para agregar identificadores de automóviles que se han cruzado
totalcars = 0 # realiza un seguimiento del total de autos
```

Fig 2. Características del video de entrada.

En la figura 2, se observa las líneas de código que permiten obtener las características del video como: fps, conteo de frames, ancho y alto. También se toma el numero entero de los ancho y alto del video que después son visualizado en la consola de salida, así permitiendo al usuario observar las características del video.

En este mismo bloque también se crea un dataframe el cual va tener el mismo número de filas que el recuento de frames, este dataframe va a cumplir la función de poder almacenar en que frame se detectó un vehículo en la estación de servicio. Además, se crean tres variables que van a almacenar el seguimiento del cuadro actual, los vehículos que cruzan por la línea que se tiene dentro del video y el número total de vehículos detectados, dos vectores en blanco que van almacenar los id de los vehículos detectados y lo que cruzan por línea de salida.

### *Aplicación de operaciones morfológicas:*

```

48     while True:
49
50         ret, frame = cap.read() # import image
51
52         if ret: # if there is a frame continue with code
53
54             image = cv2.resize(frame, (0, 0), None, ratio, ratio) # resize image
55
56             gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # converts image to gray
57
58             fgbg = fgbg.apply(gray) # uses the background subtraction
59
60             # applies different thresholds to fgmask to try and isolate cars
61             # just have to keep playing around with settings until cars are easily identifiable
62             kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5)) # kernel to apply to the morphology
63             closing = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)
64             opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
65             dilation = cv2.dilate(opening, kernel)
66             ret, mask = cv2.threshold(dilation, 228, 255, cv2.THRESH_BINARY) # removes the shadows
67
68
69

```

Fig 3. Aplicación de operaciones morfológicas:

El objetivo de esto es aplicar una máscara que permita separar el fondo de los vehículos, los cuales son el objetivo para identificar del plano completo. Por lo anterior en el algoritmo substraer el fondo mediante la función `cv2.createBackgroundSubtractorMOG2()` “`fgbg.apply(gray)`” la cual será la máscara, la continuación mediante la función Cv2. Método GetStructuringElement La cual nos establece el kernel deseado para aplicar las transformaciones morfológicas apertura (erosión + dilatación) y cierre (dilatación + erosión), estas transformaciones morfológicas mediante el Kernel establecido suaviza los contornos y recupera elementos permitiendo redondear contornos, mediante el cierre se unen los elementos cercanos. A continuación, aplicamos una dilatación adicional con el fin de unir posibles partes rotas de la imagen; todo lo anterior en escala de grises lo cual al comienzo se pasa todo a escala de grises.

Por ultimo y como objetivo se implementa una umbralización para eliminar las sombras resultantes después de realizadas las diferentes transformaciones morfológicas mencionadas.

#### Fijación de contornos:

Para hacer la determinación de contornos, vemos que con `findContours`, encontramos los contornos una lista con puntos límites alrededor de cada objeto y cómo se relacionan entre sí mismos para poder posteriormente usar cascinos convexos para cada uno de los contornos con la función `convexHull`. Posterior a esto, dibujamos dichos cascinos convexos, pero teniendo en cuenta que un casco convexo por sí solo es solo un contorno. Por lo que, usaremos la función `drawContours`. Todo este proceso se ve ilustrado en la figura 4.

```

# creates contours
contours, hierarchy = cv2.findContours(bins, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# use convex hull to create polygon around contours
hull = [cv2.convexHull(c) for c in contours]

# draw contours
cv2.drawContours(image, hull, -1, (0, 255, 0), 3)

```

Fig 4 . Creacion de contorno

Ahora, procedemos a poner líneas en la imagen que ayudarán a poder delimitar cuándo se dejarán de contar contornos, necesitados debido a que los carros a ciertas distancias incrementan sus contornos. Luego de fijar las líneas, procedemos a acotar el área del contorno, para poder tener una aproximación más precisa. Esto se necesita para filtrar contornos generados por ruido, así como también poner contornos de tamaños grandes para vehículos como buses.

```
# line created to stop counting contours, needed as cars in distance become one big contour
#lineypos = 390
lineypos = 170
cv2.line(image, (0, lineypos), (width, lineypos), (255, 0, 0), 5)

# line y position created to count contours
#lineypos2 = 5
lineypos2 = 400
cv2.line(image, (0, lineypos2), (width, lineypos2), (0, 255, 0), 5)

# min area for contours in case a bunch of small noise contours are created
minarea = 600

# max area for contours, can be quite large for buses
maxarea = 50000

# vectors for the x and y locations of contour centroids in current frame
cxx = np.zeros(len(contours))
cyy = np.zeros(len(contours))
```

Fig 5. Creacion maximos y minimos contornos

### Cálculos de centroides:

Una vez se encuentran los contornos de los vehículos, se procede a calcular los centroides de los contornos mediante los momentos. Los momentos son una medida particular que indica la dispersión de una nube de punto, y matemáticamente se definen como:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

Donde x,y son las coordenadas de un pixel y la función I(x,y) indica su intensidad y como se aplicó una máscara binaria, por lo tanto la función I(x,y) solo puede ver 0 o 1. Hay tres momentos que nos interesan: M00,M01 y M10. Por lo cual la ecuación para M00, se transforma en:

$$M_{00} = \sum_x \sum_y x^0 y^0 I(x, y) = \sum_x \sum_y I(x, y)$$

Como I(x,y) solo pueda dar 0 o 1, la fórmula de M00 es equivalente al número de pixeles cuyo valor es 1. Por tanto, M00 es el área en pixeles de la región blanca.

Para M10, la formula original se transforma en:

$$M_{10} = \sum_x \sum_y x^1 y^0 I(x, y) = \sum_x \sum_y xI(x, y)$$

Esta ecuación es igual a la suma de las coordenadas de x de los pixeles cuya intensidad es 1, por lo cual dividiendo M10 por M00 se obtendría el centroide para la componente x:

$$\frac{M_{10}}{M_{00}} = \frac{\sum_x \sum_y xI(x, y)}{\sum_x \sum_y I(x, y)} = \frac{x_1 + x_2 + \dots + x_n}{n} = \bar{x}$$

Y para el momento M01 es:

$$\frac{M_{01}}{M_{00}} = \frac{\sum_x \sum_y yI(x, y)}{\sum_x \sum_y I(x, y)} = \frac{y_1 + y_2 + \dots + y_n}{n} = \bar{y}$$

```
# vectores para las ubicaciones X y Y de los centroides de contorno en el cuadro actual
cxx = np.zeros(len(contours))
cyy = np.zeros(len(contours))

for i in range(len(contours)): # recorre todos los contornos en el cuadro actual

    if hierarchy[0, i, 3] == -1: # usar la jerarquía para contar solo los contornos principales (los contornos no están dentro de otros)

        area = cv2.contourArea(contours[i]) # área del contorno

        if minarea < area < maxarea: # umbral de área para contorno

            # calcular centroides de contornos
            cnt = contours[i]
            M = cv2.moments(cnt)
            cx = int(M['m10'] / M['m00'])
            cy = int(M['m01'] / M['m00'])

            if cy > lineypos: # iltra los contornos que están por encima de la línea (y comienza en la parte superior)

                # obtiene puntos delimitadores del contorno para crear un rectángulo
                # x, y es la esquina superior izquierda y w, h es ancho y alto
                x, y, w, h = cv2.boundingRect(cnt)

                # crea un rectángulo alrededor del contorno
                cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)

                # Imprime el texto del centroide para verificarlo más adelante
                cv2.putText(image, str(cx) + "," + str(cy), (cx + 10, cy + 10), cv2.FONT_HERSHEY_SIMPLEX,
                           .3, (0, 0, 255), 1)

                cv2.drawMarker(image, (cx, cy), (0, 0, 255), cv2.MARKER_STAR, markerSize=5, thickness=1,
                               line_type=cv2.LINE_AA)

            # agrega centroides que pasaron los criterios anteriores a la lista de centroides
            cxx[i] = cx
            cyy[i] = cy
```

Fig 6 . Cálculo de centroides

En la figura 6, se observa las líneas de código que crean dos vectores para poder ubicar los centroides de los contornos existentes, además recorre todos los contornos en el cuadro actual y se calcula el área del contorno, con el fin poder delimitar el mínimo y máximo del área, así evitando detectar persona como vehículos ya que el área del contorno de una persona es menor al de un vehículo.

Una vez se calculan los centroides de x, y basado en las fórmulas que anteriores, se utilizan las funciones de cv2.boundingRect y cv2.rectangle, permitiendo crear un rectángulo de color rojo alrededor del contorno, también imprime en texto en donde está ubicado el centroide para un posterior análisis. Por último, se guardan los valores de los centroides calculados en los vectores cxx y cyy.

```

currentcars = 0 # coches actuales en pantalla
currentcarsindex = [] # coches actuales en la pantalla índice carid

for i in range(len(carids)): # recorre todo carids

    if df.at[int(framenumber), str(carids[i])] != '':
        # comprueba el marco actual para ver qué ID de coche están activos
        # al verificar que el centroide existe en el marco actual para cierta identificación de automóvil

        currentcars = currentcars + 1 # agrega otro a los autos actuales en pantalla
        currentcarsindex.append(i) # agrega identificadores de automóviles a los automóviles actuales en la pantalla

for i in range(currentcars): #recorre todos los identificadores de automóviles actuales en la pantalla

    # toma el centroide de cierto carid para el marco actual
    curcent = df.iloc[int(framenumber)][str(carids[currentcarsindex[i]])]

    # toma el centroide de cierto carid para el cuadro anterior
    oldcent = df.iloc[int(framenumber - 1)][str(carids[currentcarsindex[i]])]

    if curcent: # si hay un centroide actual

        # Texto en pantalla para el centroide actual
        cv2.putText(image, "Centroid" + str(curcent[0]) + "," + str(curcent[1]),
                   (int(curcent[0]), int(curcent[1])), cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 255, 255), 2)

        cv2.putText(image, "ID:" + str(carids[currentcarsindex[i]]), (int(curcent[0]), int(curcent[1] - 15)),
                   cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 255, 255), 2)

        cv2.drawMarker(image, (int(curcent[0]), int(curcent[1])), (0, 0, 255), cv2.MARKER_STAR, markerSize=5,
                      thickness=1, line_type=cv2.LINE_AA)

        if oldcent: # comprueba si existe un centroide antiguo
            # agrega el cuadro de radio del centroide anterior al centroide actual para la visualización
            xstart = oldcent[0] - maxrad
            ystart = oldcent[1] - maxrad
            xwidth = oldcent[0] + maxrad
            yheight = oldcent[1] + maxrad
            cv2.rectangle(image, (int(xstart), int(ystart)), (int(xwidth), int(yheight)), (0, 125, 0), 1)

            # comprueba si el centroide antiguo está en la línea o debajo de ella y si la curva está en la línea o arriba
            # para contar automóviles y ese automóvil aún no se ha contado
            if oldcent[1] >= lineypos2 and curcent[1] <= lineypos2 and carids[
                currentcarsindex[i]] not in carscrossed:

                carscrossedup = carscrossedup + 1
                cv2.line(image, (0, lineypos2), (width, lineypos2), (0, 0, 255), 5)
                carscrossed.append(
                    currentcarsindex[i]) # agrega la identificación del automóvil a la lista de automóviles de recuento para evitar el

            # comprueba si el centroide antiguo está en la línea o por encima de ella y si la curva está en la línea o debajo de ella
            # para contar automóviles y ese automóvil aún no se ha contado
            elif oldcent[1] <= lineypos2 and curcent[1] >= lineypos2 and carids[
                currentcarsindex[i]] not in carscrossed:

                carscrosseddown = carscrosseddown + 1
                cv2.line(image, (0, lineypos2), (width, lineypos2), (0, 0, 125), 5)
                carscrossed.append(currentcarsindex[i])

        # TTexto en pantalla de la esquina superior izquierda
        cv2.rectangle(image, (0, 0), (250, 100), (255, 0, 0), -1) # rectángulo de fondo para texto en pantalla

        cv2.putText(image, "Cars in Area: " + str(currentcars), (0, 15), cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 170, 0), 1)

        cv2.putText(image, "Cars Crossed Up: " + str(carscrossedup), (0, 30), cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 170, 0),
                   1)

        cv2.putText(image, "Cars Crossed Down: " + str(carscrosseddown), (0, 45), cv2.FONT_HERSHEY_SIMPLEX, .5,
                   (0, 170, 0), 1)

        cv2.putText(image, "Total Cars Detected: " + str(len(carids)), (0, 60), cv2.FONT_HERSHEY_SIMPLEX, .5,
                   (0, 170, 0), 1)

        cv2.putText(image, "Frame: " + str(framenumber) + ' of ' + str(frames_count), (0, 75), cv2.FONT_HERSHEY_SIMPLEX,
                   .5, (0, 170, 0), 1)

        cv2.putText(image, 'Time: ' + str(round(framenumber / fps, 2)) + ' sec of ' + str(round(frames_count / fps, 2))
                   + ' sec', (0, 90), cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 170, 0), 1)

```

Fig 7 . Etiquetado de centroides

La figura 7 se observa la líneas de código que encargan de realizar el etiquetado de los centroides en la pantalla. Inicia mediante un for recorriendo el vector carids, donde con una condición se comprueba si el frame actual hay un vehículo activo y así ir aumentando el contador de vehículos actuales en la pantalla, pero para poder visualizar los resultados en la pantalla, hay dos condiciones en la cual la primera consta si el centroide es del frame actual se encarga de poner en texto de pantalla el centroide y le ID del vehículo, mediante la función de put.Text. La segunda condición es si el centroide es de frame anterior, se calcula el radio del centroide para poder visualizarlo, además pregunta si el centroide esta por debajo o por encima la línea que esta presente en el video y así ir aumentando el contador de vehículos que cruzaron la línea, este contador tiene el nombre carscrossedup, por el ultimo se escriben las líneas que permite visualizar todos resultados calculados en la izquierda superior de la pantalla de salida.

```

295     + sec, (0, 90), cv2.FONT_HERSHEY_SIMPLEX, .5, (0, 170, 0), 1)
296
297     # muestra imágenes y transformaciones
298     cv2.imshow("countours", image)
299     cv2.moveWindow("countours", 0, 0)
300
301     cv2.imshow("fgmask", fgmask)
302     cv2.moveWindow("fgmask", int(width * ratio), 0)
303
304     cv2.imshow("closing", closing)
305     cv2.moveWindow("closing", width, 0)
306
307     cv2.imshow("opening", opening)
308     cv2.moveWindow("opening", 0, int(height * ratio))
309
310     cv2.imshow("dilation", dilation)
311     cv2.moveWindow("dilation", int(width * ratio), int(height * ratio))
312
313     cv2.imshow("binary", bins)
314     cv2.moveWindow("binary", width, int(height * ratio))
315
316     video.write(image) # guardar la imagen actual en el archivo de video de antes
317
318     # se suma al recuento de cuadros
319     framenumber = framenumber + 1
320
321     k = cv2.waitKey(int(1000/fps)) & 0xff # int(1000/fps) velocidad normal ya que la tecla de espera está en ms
322     if k == 27:
323         break
324
325 else: # si el video está terminado, interrumpe el ciclo
326     break
327
328 cap.release()
329 cv2.destroyAllWindows()
330
331 # guarda el marco de datos en un archivo csv para su posterior análisis
332 df.to_csv('traffic.csv', sep=',')

```

Fig 8. Visualización

Las ultimas líneas de código se pueden visualizar en la figura 8, se encargan de mostrar todas las operaciones morfológicas que se realizan, además la máscara binaria, la sustracción de fondo de primer plano y mostrar el video de entrada con los resultados que de conteo de vehículos dentro de la estación. Por último se crea un archivo csv, el cual almacena en qué frame se detectó el vehículo y la ubicación del centroide dentro del video.

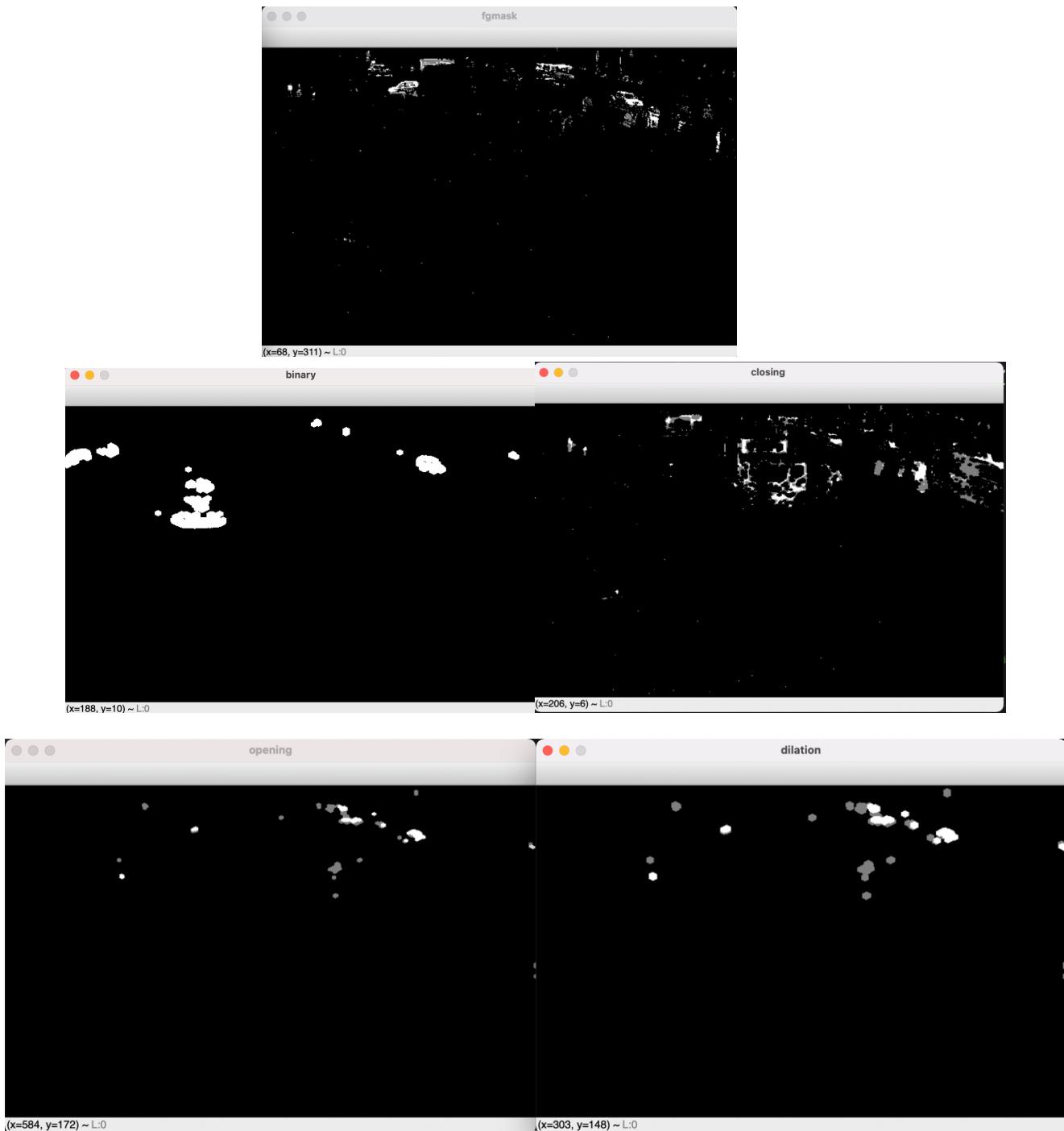
**Resultados:**

Fig 9. Resultados

La figura 9, se observa los resultados de salida en el algoritmo , en el cual podemos visualizar las operaciones morfológicas que fueron aplicadas para poder realizar una detección de vehículos mas sencillas y evitando el ruido en el video, también observan la capa de binaniracion que se aplico para poder identificar los contornos y la salida de la sustracción de primer plano con respecto a la mascara que se aplico.



Fig 10. Resultado final.

El resultado final se presenta en la figura 10, la cual en la parte superior izquierda se muestran los resultados calculados como vehículos en el área, numero total de vehículos, frames y tiempo total de ejecución, también se observa como el algoritmo detecta cuando el vehículo de color de amarillo lo detecta y muestra su contorno, además el contador de vehículos totales esta en uno, lo cual indica que algoritmo esta detectado los vehículos de manera correcta.

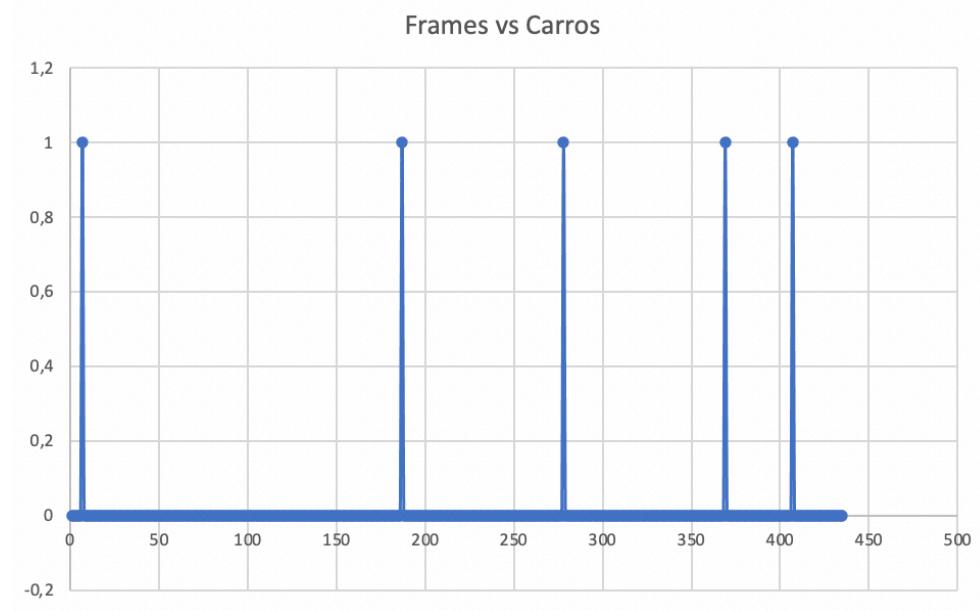


Fig 11. Frames vs carros

La grafica frames vs carros, esta dada donde en el eje x se tiene el total de frame del video de la estación de servicio, y en el eje y hay valor de 1 o 0, donde 1 si detecto en vehículo y 0 cuando no hay detección

de algún vehículo, por lo cual en este video de prueba el algoritmo detecto 5 vehículos en diferentes lapsos de tiempo.

**Conclusiones:**

- Se pudo determinar que este algoritmo es efectivo, pero debemos fijar condiciones para un funcionamiento eficaz de este mismo.
- El cálculo de centroides, en conjunto con la fijación de áreas máximas y mínimas para así determinar qué tamaño de vehículos serán seguidos, para luego poner una línea en el video que nos ayude a determinar si pasó o no, y contarlos, para que no se cuenten carros que circulen por fuera de la estación es una muestra de cuánto acotamiento necesita este algoritmo.