Registro y Control Vehicular

1st Santiago Márquez Álvarez

Pontificia Universidad Javeriana
Bogotá, Colombia
marquezsantigo@javeriana.edu.co

2nd Sergio Mora Pradilla

Pontificia Universidad Javeriana
Bogotá, Colombia
smora@javeriana.edu.co

Resumen: Se analizaron y procesaron un conjunto de imágenes de automotores de transporte público con el fin de obtener un registro de las placas de dichos vehículos y almacenarlas en una lista de caracteres. Para ello se usa Tesseract como herramienta OCR y distintas funciones de filtrado de imagen de open-cv.

I. Introducción

Dada la necesidad de mantener el servicio de transporte público SITP activo en un ámbito post-pandemia es necesario que los automotores dispuestos a prestar el servicio de transporte público sean sometidos a rigurosos controles y protocolos de limpieza diarios con el fin de disminuir la propagación del virus COVID-19. Para ello se lleva un control digitado a mano de los vehículos que han sido sometidos a los protocolos de limpieza y desinfección, anotando la placa y su número de identificación. Debido a la alta cantidad de vehículos que deben ser limpiados, el procedimiento de registro y control a mano ha demostrado ser ineficiente al requerir bastante tiempo y presentar discrepancias.

La solución propuesta procesa imágenes donde se aprecia con claridad las placas que identifican a cada vehículo y finalmente son almacenadas en una lista sin que se repitan. En este documento se explicará el uso de la herramienta Tesseract utilizada para el reconocimiento de caracteres y los filtros de pre-procesamiento para facilitar dicho reconocimiento, Se comparten resultados y conclusiones.

II. Herramientas Teóricas

A. Tesseract OCR

Tesseract es una herramienta óptica de reconocimiento de caracteres que transforma una imagen de dos dimensiones en texto sin importar si este es escrito por una persona o impreso por una maquina. Este OCR, (por sus siglas en inglés "Optical Character Recognition"), consta de varios sub procesos: pre-procesamiento de imagen, localización de texto, segmentación de caracteres, reconocimiento de caracteres y post procesado. Esta herramienta es usada mayormente para la recolección de información escrita en documentos para ser digitalizada y almacenada. Las ultimas generaciones de OCR se basan en herramientas de Deep-Learning dando con resultandos con una precisión bastante aceptable. Sin embargo, sigue teniendo dificultades cuando la imagen no se presenta en las mejores condiciones y es necesario un procesamiento que facilite al OCR el reconocimientos de caracteres.[1]

B. suavizado Gaussian

El suavizado Gaussiano es útil a la hora de eliminar ruido o información no deseada en los bordes presentes en la imagen. Esto se logra mediante la convolución de la imagen y una mascara con filtro pasa bajo, removiendo información de altas frecuencias dando el efecto de suavizado.[2]

C. Filtro Bilateral

El filtro bilateral es otro filtro para suavizar la imagen, similar al filtro gaussiano con la particularidad de que se aprecien más los bordes en la imagen.[2]

D. Detección de bordes con canny

El algoritmo canny para detección de bordes delimita los bordes a partir del cálculo del ángulo del gradiente en orientación horizontal y vertical de la imagen previamente suavizada. Los bordes siempre son perpendiculares a la dirección del gradiente. Con los valores de dirección y magnitud del gradiente se recorre la imagen para eliminar cualquier píxel que no haga parte de un borde.[3]

E. Umbralización

El filtrado por umbralización toma una imagen en escala de grises, si el valor de un píxel es menor al umbral especificado, se hace igual a 0, de lo contrario toma el valor máximo especificado como parámetro.[4]

III. Solución propuesta

Para la obtención de los caracteres, las imágenes deben ser filtradas para encontrar la placa. Una vez identificada la placa se buscan los caracteres que contiene,

A. Filtrado de imagen completa

Las imágenes se filtran con dos alternativas por si la primera no arroja resultados. El primer filtrado se hace con la imagen en escala de grises, se aplica un su avisado gaussiando (Gaussian blur) y luego una umbralizacion (Thresholding) para dejar el contorno de la placa en blanco y su exterior en negro. Hay imágenes en las cuales lo anterior no basta para diferenciar el contorno de la placa, por lo que la segunda alternativa de filtrado es aplicar un filtro bilateral (Bilateral filter)a la imagen en escala de grises y luego detección de bordes mendicante canny.

```python
def pre_procesamiento(self, image):
    img = cv2.imread(image)
    self.image_draw = img.copy()
    self.img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    N = 7
    if self.metodo == 0:
        image_gauss_lp = cv2.GaussianBlur(self.img_gray, (N, N), 1.5, 1.5)
        ret2, self.bordes = cv2.threshold(image_gauss_lp, 100, 150, cv2.THRESH_BINARY)
    else:
        img_bil = cv2.bilateralFilter(self.img_gray, 25, 25, 50)
        self.bordes = cv2.Canny(img_bil, 70, 200)
```

Fig. 1. Alternativas de filtrado

Hay imágenes en donde funciona la primer alternativa, otras la segunda y en otras ambas. Para el ultimo caso, el resultado encontrado es el mismo.

B. Filtrado de Placa

Una vez diferenciado el contorno de la placa en la imagen, se procede a encontrar dicho contorno.

```python
def contornos(self):
    contourss = cv2.findContours(self.bordes, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    self.contours = imutils.grab_contours(contourss)
    self.contours = sorted(self.contours, key=cv2.contourArea, reverse=True)[:10]
    self.screenCnt = None
```

Fig. 2. obtención de contornos

Para descartar el contorno de la placa del resto de contornos se buscan entre todos los contornos en la imagen un contorno que se aproxime a un polígono de 4 vértices.

```python
def approx_Rectangular(self,contorno):
    cx, cy, self.w, self.h = cv2.boundingRect(contorno)
    peri = cv2.arcLength(contorno, True)
    self.approx = cv2.approxPolyDP(contorno, 0.018 * peri, True)
    self.mask = np.zeros(self.img_gray.shape, np.uint8)
```

Fig. 3. Aproximación a un polígono de 4 lados

Estos contornos de 4 vértices se filtran de nuevo mediante el cálculo de su relación de aspecto. Dado que las placas en las imágenes tiene distorsionada su orientación y perspectiva respecto a la cámara de formas distintas, se filtran los contornos entre un rango de relación de aspecto de entre 0.8 y 1.6. En este rango de valores se filtran los contornos de las placas, sin embargo, se pueden filtrar otros contornos que no pertenezcan a estas.

```python
def rel_aspecto(self, contorno):
    cx, cy, w, h = cv2.boundingRect(contorno)
    self.aspect_Ratio = float(w) / h
```

Fig. 4. Filtrado por relación de aspecto

Con el contorno de la placa encontrado, se procede a corregirle la orientación y el ángulo de perspectiva mediante transformaciones Homógrafas.

Finalmente se procesa la placa usando la herramienta Tesseract -OCR para identificar los caracteres en su interior.

```python
def homografia(self, screenCnt):

    # obtencion pts transfromacion
    points1 = screenCnt
    if screenCnt[1][0][0] > screenCnt[0][0][0] and screenCnt[1][0][1] > screenCnt[0][0][1]:
        if screenCnt[0][0][0] < screenCnt[2][0][0]:
            points2 = [(0, 0), (100, 0), (100, 49), (0, 49)]
        else:
            points2 = [(100, 0), (100, 49), (0, 49), (0, 0)]

    elif screenCnt[1][0][0] < screenCnt[0][0][0] and screenCnt[1][0][1] > screenCnt[0][0][1]:
        if screenCnt[0][0][0] < screenCnt[2][0][0]:
            points2 = [(0, 0), (0, 49), (100, 49), (100, 0)]
        else:
            points2 = [(100, 0), (0, 0), (0, 49), (100, 49)]

    # transformacion perspectiva
    N = 4
    pts1 = np.array(points1[:N])
    pts2 = np.array(points2[:N])
    H, _ = cv2.findHomography(pts1, pts2, method=cv2.RANSAC)
    self.plate_solo = cv2.warpPerspective(self.image_draw, H, (100, 49))
```

Fig. 5. Transformaciones de la placa

Dada la configuración del OCR, en un banco de imágenes de 20 placas, se identifica su contenido en todas sin error, Los contornos donde no se encontraron caracteres, son contornos que no pudieron ser filtrados y al no tener información alguna relevante, son ignorados. La información encontrada es almacenada en una lista donde se encuentran todas las placas sin duplicados dentro de la lista.

```python
def OCR(self):
    custom_config = r' --psm 6 -c tessedit_char_whitelist=1234567890QWERTYUIOPLKJHGFDSAZXCVBNM'
    self.final = pytesseract.image_to_string(self.plate_solo, config=custom_config)

def guardar_placa(self):
    self.Finish = True
    self.placas = np.append(self.placas, self.final[0:6])

def filtrar_duplicados(self):
    for plate in self.placas:
        if plate not in self.placas_final:
            self.placas_final.append(plate)
```

Fig. 6. Obtención de caracteres y ordenamiento

IV. Resultados

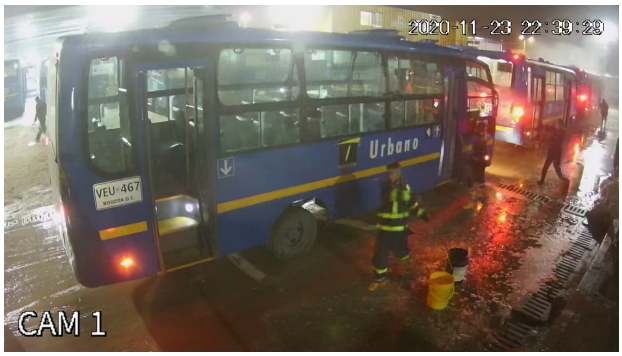

Fig. 7. Imagen original.

Fig. 8. Primer alternativa de filtrado.

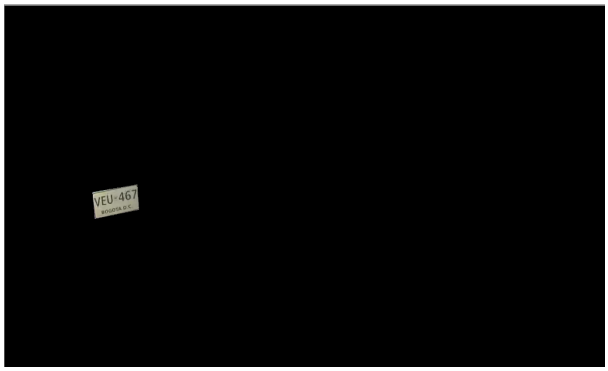

Fig. 9. Mascara únicamente de la placa.

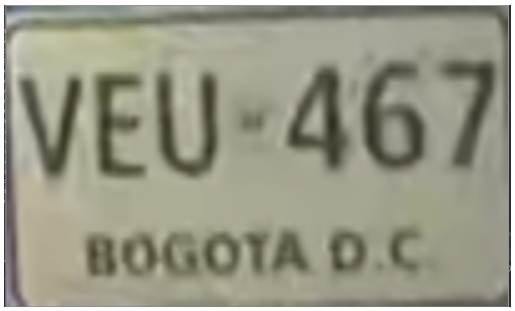

Fig. 10. Placa con perspectiva y ángulo corregido

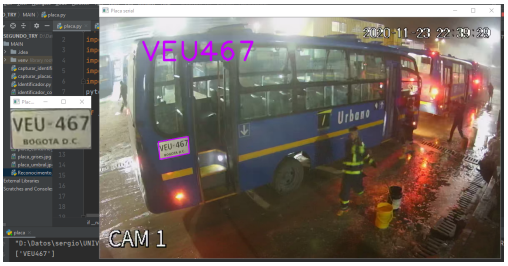

Fig. 11. Resultado Final

En las imágenes anteriores se aprecia el procedimiento de filtrado para obtener el contorno de la placa y su información. Este procedimiento se lleva acabo con un banco de 20 imágenes todas con resultados satisfactorios.

Este banco de imágenes Fue filtrado previamente porque hay casos en los que el análisis para encontrar el contorno con los métodos planteados no es viable. Las imágenes no están en las mejores condiciones para ser analizadas dado que las condiciones donde están siendo capturadas son desfavorables.

V. CONCLUSIONES

- La herramienta Tesseract es muy eficiente y da resultados satisfactorios siempre y cuando la imagen donde yace el texto sea legible y no tenga distorsión de ningún tipo.
- El banco de imágenes original es mayor a 20, sin embargo muchas imágenes fueron descartadas debido a que para capturarlas se usaron cámaras de vigilancia cuya finalidad no es la de analizar placas. También debido a cómo está organizado el sitio de lavado, no siempre las cámaras capturan las placas bien sea por que están empañadas por el agua o por que la placa no alcanza a salir en el campo visual de la cámara. También si el vehículo está en movimiento las letras de la placa se verán borrosas haciendo imposible reconocerlas para Tesserat OCR
- Puede darse el caso de que algunas veces esté mas de una placa en la imagen. Esto a futuro puede ser una mejora a implementar pues solo se esta tomando una placa por imagen.
- La implementación para vídeo no se llevó acabo dado que el sistema de grabación no permite la transferencia de archivos de vídeo a dispositivos externos, solo imágenes. Por lo que dadas las circunstancias de captura de imagen y las dificultades que presenta es necesario recurrir a otro tipo de herramientas de análisis mas avanzadas como redes neuronales o entrenamientos de algoritmos que puedan llevar acabo un mejor trabajo en estas condiciones
- Dado que los buses tienen una placa e identificador único, Se opta por la opción de hacer un reconocimiento óptico solo de las placas, con esta información, comparar en una base de datos y obtener el identificador del bus.

VI. REFERENCIAS

[1]“A comprehensive guide to OCR with Tesseract, OpenCV and Python”. 2020. En línea Disponible en: https://nanonets.com/blog/ocr-with-tesseract/introduction

[2]Documentación open cv. “smoothing images”. En línea Disponible en: docs.opencv.org/master/d4/d13/tutorial$_{p}y_{f}iltering.html$

[3]Documentación open cv.”Canny edge detection”En línea Disponible en: https://docs.opencv.org/master/da/d22/tutorial$_{p}y_{c}anny.html$

[4]Documentación open cv.”image thresholding”En línea Disponible en: https://docs.opencv.org/master/d7/d4d/tutorial$_{p}y_{t}hresholding.html$