
Reconocimiento automático de placas vehiculares

Jorge Ramos

Jonatan Romo

Kenny Méndez

Adrián Rodríguez

Centro de Investigación en Matemáticas. Unidad Monterrey.

Resumen—El problema del reconocimiento automático de placas vehiculares se ha convertido en una fuente de métodos muy variados para intentar resolverlo, desde métodos basados en conceptos básicos usados en el procesamiento digital de imágenes hasta métodos de aprendizaje automático. En este trabajo se exploran algunas de las soluciones a dicho problema con base en diferentes enfoques, uno de ellos basados en machine learning y otros basados en métodos de procesamiento de imágenes digitales. Se utilizaron tres métodos para la detección de placas: Global Threshold, Adaptative threshold - Detección de bordes de Canny, y Cascada de Haar. Para el reconocimiento de los caracteres en las placas vehiculares se exploraron dos métodos basados en machine learning: KNN y Tesseract (LSTM estable).

I. INTRODUCCIÓN

El reconocimiento automático de placas vehiculares es un problema importante debido a que el crecimiento del parque vehicular en los países desarrollados y en desarrollo va en aumento, por lo que mantener control de velocidades para evitar accidentes y la capacidad de encontrar vehículos robados se han convertido en problemas comunes y que pueden prevenirse con un buen uso de esta aplicación.

Se han establecido varias soluciones a este problema, pero no existe una solución universal debido a que se pueden presentar problemas como vehículos en movimiento, cambios de iluminación, movimientos de la cámara, obstrucciones, cercanía-lejanía de la foto al vehículo, fondo cambiante. Por esta razón se han presentado diversos métodos para intentar resolver un problema muy específico.

En este trabajo se presenta un enfoque simple para realizar el reconocimiento de las placas vehiculares desde la detección de la placa en el vehículo, hasta la lectura de los caracteres que contiene la placa. Un diagrama básico del proceso seguido para el reconocimiento de las placas se muestra en el anexo.

II. DETECCIÓN DE PLACAS

La detección de placas es el primer paso del proceso del reconocimiento de placas vehiculares (sin considerar el

preprocesamiento previo), este implica obtener los pixeles específicos dentro de la imagen en los que se encuentra una placa vehicular. Para este trabajo se consideraron tres posibles formas de obtener dichas coordenadas: Detector de bordes de Canny con Adaptative threshold, umbrales globales y una cascada de Haar.

Los primeros dos métodos son métodos estándar en la literatura de procesamiento de imágenes digitales y el tercero corresponde a un método desarrollado inicialmente para la detección de rostros humanos basado en técnicas de machine learning que puede adaptarse para la detección de cualquier otro objeto.

II-A. *Detector de bordes de Canny*

El algoritmo de detección de bordes de Canny permite obtener bordes limpios y delgados que están bien conectados con los bordes cercanos, se trata de un detector multietapas que consiste de cuatro pasos:

1 **Preprocesamiento:** Antes de iniciar con el proceso de detección de bordes debe considerarse que un problema al que regularmente se enfrentan los detectores de borde es que son sensibles al ruido. Debido a esto, es aconsejable aplicar un poco de suavizado con un desenfoque (blur), usualmente un desenfoque gaussiano suele ayudar.

2 **Cálculo de gradientes:** En esta etapa, se calculan las magnitudes y direcciones en cada punto de la imagen. Donde la magnitud del gradiente en cada punto determina si posiblemente yace o no un borde, destacando que una magnitud alta señala que los colores cambian rápidamente, lo cual implica que hay un borde, y de manera inversa cuando la magnitud es pequeña.

La dirección del gradiente muestra hacia dónde está orientado el borde. Para calcularlos se utiliza el **detector de bordes sobel**, el cual trabaja en primeras derivadas de

manera separada para los ejes X y Y.

La magnitud del gradiente está dada por $m = \sqrt{G_x^2 + G_y^2}$, y la dirección por $\theta = \arctan \frac{G_y}{G_x}$, donde G_x y G_y son las derivadas X y Y en el punto considerado. Una vez ya calculadas se inicia con la detección de bordes.

3 Supresión de no máximos: En esta etapa, si un pixel no es un máximo, este es suprimido. Para hacer esto, se hace una iteración sobre todos los píxeles de la imagen. Asimismo, la orientación de cada pixel se coloca dentro de cuatro contenedores, los cuales rodean al pixel en cuestión, ya que hay cuatro posibles bordes; ir de norte a sur, de este a oeste, o en diagonal de arriba hacia abajo o de abajo hacia arriba.

Asimismo, las cuatro posibilidades necesitan ser consideradas por separado para revisar la supresión de no máximo.

Despues de realizar la supresión de no maximos, se obtienen los llamados “bordes delgados”. Este podría ser roto en varios puntos, lo cual se arreglarán rellenando huecos con otro umbral (con histéresis) en la siguiente etapa.

4 Umbral con histéresis: En la etapa anterior, se marcaron los píxeles que tenían una magnitud de gradiente mayor que el umbral superior. Ahora, utilizando la información de dirección y el umbral inferior, “se amplifican” estos bordes.

Esto es, los pasos son:

- Si el píxel actual no es un borde, se revisa el siguiente
- Si es un borde, se revisa a los dos píxeles en la dirección del borde. Si alguno de ellos (o ambos):
 - Tienen la dirección en el mismo contenedor que el píxel central. La magnitud del gradiente es mayor que el umbral inferior
 - Son el máximo en comparación con sus vecinos (supresión no máxima para estos píxeles), entonces se pueden marcar estos píxeles como un píxel de borde
 - Se realiza el ciclo hasta que no haya cambios en la imagen, así, una vez terminados los cambios se obtienen los bordes de Canny.

II-B. Umbral Adaptativo Básico (Basic Adaptive Thresholding)

Los factores de una imagen tales como las iluminaciones desiguales pueden transformar un histograma perfectamente segmentable en un histograma que no pueda ser particionado efectivamente por un umbral global singular. Una

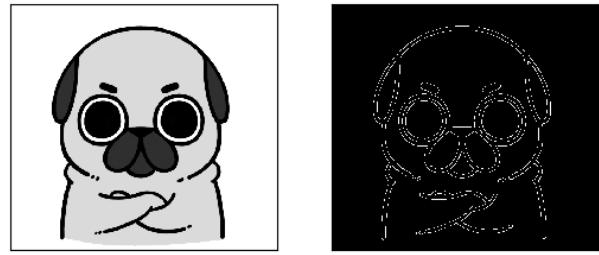


Figura 1: Imagen original e imagen aplicando el filtro de Canny

aproximación para manipular tales situaciones es dividir la imagen en subimagenes y entonces utilizar un umbral diferente para segmentar cada subimagen. Las cuestiones clave en esta aproximación yacen en cómo subdividir la imagen y cómo hacer la estimación del umbral para cada subimagen obtenida. Debido a que el umbral utilizado para cada píxel depende de donde esté localizado dentro de la subimagen, por lo que este tipo de umbral es adaptativo.

II-C. Cascada de Haar

La cascada de Haar es un método de detección de objetos, la aportación principal de este método es su eficiencia en tiempo computacional ya que inclusive permite ser utilizado en tiempo real. Además de que se logra una muy buena tasa de detección con un nivel de falsos positivos bajo.

El aporte completo de dicho método se basa en tres ejes:

- Características de tipo Haar
- Imagen integral
- Cascada de clasificadores

II-C1. Características de tipo Haar: Es un método de detección de objetos que se planteo para la detección de rostros pero puede ser usado para la detección de cualquier otro objeto. El primer aporte significativo de este método es el uso de características de tipo Haar. Se conocen con este nombre debido a que son inspiradas en las funciones de base del método de wavelet que planteó Haar y que es considerado uno de los primeros métodos de reconocimiento de rostros.

Hay muchas motivos para hacer uso de características en lugar de trabajar directamente con los píxeles de la imagen. Una de las razones más importantes es que dichas características se pueden codificar de manera que se ajusten a algún conocimiento específico del objeto y que pueden ser fáciles de aprender con una cantidad finita de datos. Una segunda razón que es crítica para su adopción es que este sistema funciona mucho más rápido que un sistema basado en pixeles.

Las características de tipo Haar son planteadas por Viola como rectángulos que se parten en 2, 3 o 4 subrectángulos, y que se colorean de gris y blanco, y la característica es extraída al hacer sumas y restas de píxeles entre los subrectángulos. En la figura 2 se presentan las características definidas por Viola. Las características son usadas de la siguiente manera la suma de los rectángulos blancos se resta de la suma de los píxeles de los rectángulos grises. De igual manera en la figura se muestra 2 características que son muy significativas la primera es un cambio de tonalidad entre los ojos y las mejillas, y la segunda resalta el hecho de que la zona entre los ojos es más clara que la que rodea los ojos.

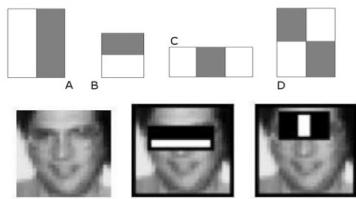


Figura 2: Características de tipo Haar

II-C2. Imagen integral: El segundo aporte del método es el de imagen integral. Debido a que se tienen que hacer la suma de los rectángulos, esta representación permite que el cálculo sea mucho más rápido. En la figura 3 se observa la idea detrás de la representación de la imagen integral.

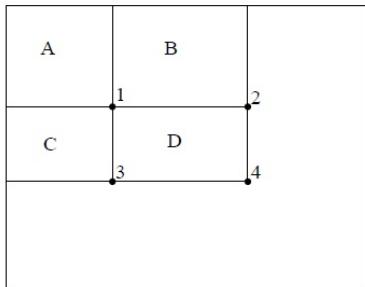


Figura 3: Imagen integral

Esta representación es muy sencilla dado un punto como dentro de la ventana, dicho punto en la representación va a ser igual a la suma de los valores en los píxeles que forman el rectángulo de los píxeles que están a su izquierda y hacia arriba. Por ejemplo en el caso de la figura 3 el punto 3 se calcula sumando A y C, y la suma en el rectángulo D se puede obtener sumando los puntos 4 y 1 y restandole a esto los puntos 2 y 3.

Ya con estos elementos es posible construir un clasificador-detector de rostros suficientemente bueno pero computacionalmente muy costoso y lento, esto último

considerando que el problema que estaban atacando al proponer el método era el de hacer reconocimiento en tiempo real.

II-C3. Cascada de Haar: Para generar el clasificador se hace uso de un algoritmo AdaBoost que a su vez usa como clasificador débil un perceptrón. Uno de los problemas a los que se enfrenta al construir el clasificador descrito anteriormente es que debe añadir una nueva característica es computacionalmente caro. Sin embargo esto involucraba aceptar una tasa de falsos positivos alta.

Recordando que AdaBoost tiene un umbral de 1/2 que implica una alta tasa de detección con un mínimo error entonces los autores del método decidieron hacer pruebas variando dicho umbral y descubrieron que al disminuirlo se podía obtener una tasa de detección suficientemente alta y a su vez disminuyendo el nivel de falsos positivos.

Siguiendo esta idea adoptaron un sistema que denominaron de cascada de clasificadores. La idea básica de dicho sistema es usar Adaboost inicialmente para generar un clasificador con pocas características permitiendo eliminar muchos subventanas con lo que el siguiente clasificador no tendría que revisarlas. El siguiente clasificador consideraría mas características para reclasificar los falsos positivos disminuyendo el umbral de AdaBoost a su vez.

Este esquema implica que se debe considerar en cada escenario las tasas de detección y de falsos positivos aceptables para cumplir con cierto objetivo indicado por el usuario. Este enfoque se eligió porque el problema de optimización que implica elegir de manera óptima el número de escenarios, el número de características para cada escenario y el umbral que se debe elegir puede tornarse muy complicado.

El algoritmo para calcular la cascada de Haar se muestra a continuación

```

El usuario define
f: nivel maximo aceptable de
falsos positivos por escenario;
d: minima tasa de deteccion por escenario;
F_target: tasa de falsos positivos final
F0=1.0; D0=1.0; i=0;
While F_i>F_target
    i=i+1;
    n_i=0; F_i=F_{i-1};
    While F_i>f x F_{i-1}
        n_i=n_i+1;
        Entrenamos AdaBoost con n_i
        caracteristicas
        Usamos un conjunto de
        validacion para calcular los nuevos F_i
    
```

```

y D_i con el clasifi-
cador cascada
Disminuimos el umbral
para el i-esimo clasifi-
cador hasta que nuestro
clasificador de cascada tenga
una tasa de detección de al
menos dxD_{i-1};
N<-NULL; #vaciamos el
conjunto de los negativos
if F_i>F_target
    Los falsos
    positivos
    se colocan en el
    conjunto de los
    negativos ;

```

Este algoritmo es muy rápido y en general da muy buenos resultados al detectar objetos. En la figura 4 se presenta una idea básica de como funciona el algoritmo sobre un rostro.

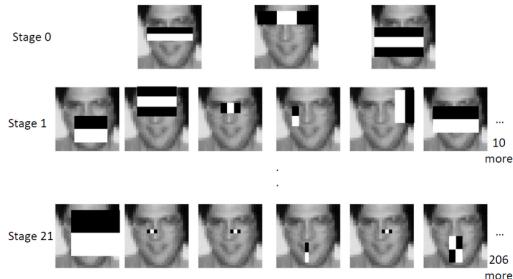


Figura 4: Aplicando una cascada de Haar sobre un rostro

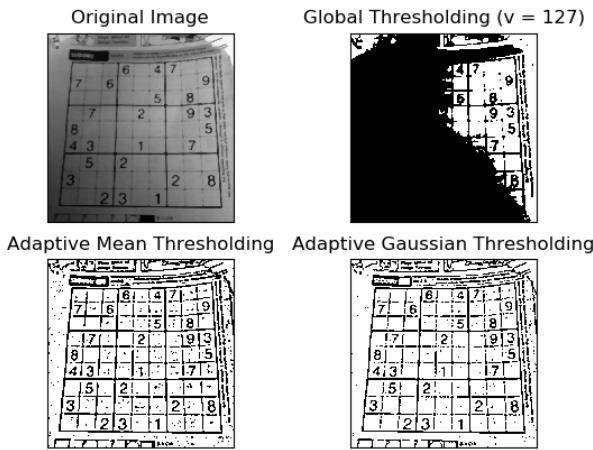


Figura 5: Imagen original con cambios de iluminación, imagen con umbral global, imagen con umbral adaptativo y kernel de media, imagen con umbral adaptativo y kernel gaussiano

III. DETECCIÓN Y LECTURA DE CARÁCTERES

Una vez encontradas las coordenadas de las placas vehiculares se debe proceder a encontrar los caracteres dentro de la placa e identificarlos para conseguir la lectura de la placa. Para esto se utilizaron dos métodos distintos para ver cual mostraba los mejores resultados, usando los métodos de detección de placas, el primero de dicho métodos es uno basado en K-Nearest Neighbors y el segundo es un método específicamente diseñado para la lectura de caracteres en imágenes conocido como Tesseract que es open source y se sigue mejorando su desarrollo hasta la fecha.

III-A. K-Nearest Neighbors (KNN)

Uno de los algoritmos mas simples de clasificación usados en machine learning es el de k-Nearest Neighbors, en el cual se tienen etiquetados casos en un conjunto de entrenamiento. Por ejemplo un conjunto de datos de entrenamiento puede consistir de n atributos o características que pueden ser representados por un vector de longitud n

$$x = (x_1, x_2, \dots, x_n)$$

donde x_1, x_2, \dots, x_n representan cada uno, cada una de las características de los objetos en el conjunto de entrenamiento. Cada uno de estos vectores x debe estar etiquetados por una variable categórica y . Así se tendrá un espacio n-dimensional en donde cada uno de los vectores x esté representados. En una situación ideal todos los vectores x etiquetados bajo una variable categórica y estarán cerca en dicho espacio n-dimensional, mientras que otros vectores etiquetados se encuentren cerca de si mismos.

Dado lo anterior, se espera que los datos pertenecientes a una categoría estén bastante cerca mientras que se encuentran alejados aquellos que no pertenecen a su categoría. Es decir, se espera que exista una relación funcional intrínseca en los datos que establezca una separación a manera de distancia. Por lo que el algoritmo KNN hace uso de esto y al entrar una nueva observación se buscan sus k-vecinos más cercanos con base a alguna función de distancia predefinida anteriormente y se elige a que categoría pertenece generalmente en base a un criterio de mayoría.

En nuestro caso se deben tener fuentes muy parecidas a aquellas con las que se encuentran escritas los caracteres en las placas. Se genera una imagen de entrenamiento que contiene todos los dígitos y caracteres usando dichas fuentes, una imagen de entrenamiento puede observarse en la figura 6. A partir de aquí se segmentan las imágenes usando un umbral global, y se localizan los contornos y de manera interactiva se establece a que dígito o carácter pertenece al ser seleccionado cada carácter durante el entrenamiento.

Haciendo lo anterior se obtiene un vector por cada uno de los caracteres y clasificados con respecto al carácter en específico que es, siendo éste etiquetado por el usuario.

```

0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

```

Figura 6: Imagen de entrenamiento para KNN

III-B. Tesseract OCR

Tesseract fue desarrollado por Hewlett Packard Labs, pero en 2005 se libero en colaboración con la Universidad de las Vegas, Nevada. A partir de 2006 debido a que se libero varios contribuidores han aparecido para mejorar sus resultados.

Tesseract 4.0 es la versión mas nueva de este producto, y la novedad es que aparte del proyecto Legacy que se tenia en Tesseract 3.0 ahora se incorpooro una red neuronal de tipo LSTM que es la usada por defecto para la lectura de caracteres en imágenes.

III-B1. LSTM: Las redes LSTM son un tipo especial de RNN, capaces de aprender dependencias de plazo largo. Fueron introducidas por Hochreiter & Schmidhuber en 1997, y fueron refinadas y popularizadas por mucha gente en trabajos subsecuentes. Estas redes trabajan tremadamente bien en una gran variedad de problemas, y son ampliamente usadas.

LSTM's estan explicitamente diseñadas para evitar el problema de dependencia de largo plazo. Esto lo logra, recordando información por largo periodos de tiempo siendo este su comportamiento basico.

Todas las redes neuronales recurrentes tienen la forma de una cadena de modulos repetidos de redes neuronales. En las RNNs estandar, este modulo repetido tiene una estructura muy simple, que puede ser una simple capa de activación con la función tangente hiperbólico como puede observarse en la figura 7. Por su parte las redes LSTM tambien tienen esta estructura de cadena, pero el modulo de repetición tiene una estructura diferente. En lugar de tener una sola capa de una sola red neuronal, tiene cuatro, interactuando en una forma muy especial como puede verse en la figura 8.

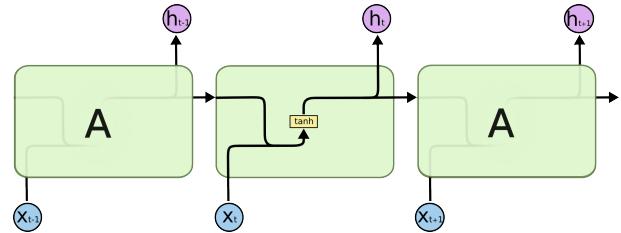


Figura 7: El modulo repetido en una RNN estándar contiene una sola capa.

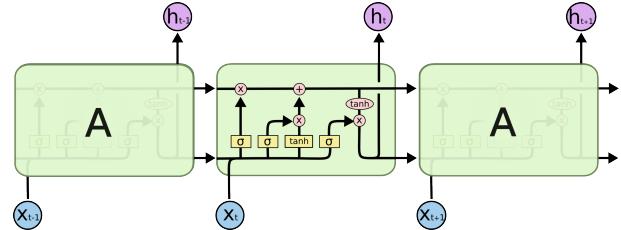


Figura 8: El modulo repetido en una red LSTM tiene cuatro capas interactuando.

La idea clave en las redes LSTM esta en el estado celda que es la linea que corre de manera horizontal a través de la parte superior de la figura 8 y que puede observarse mejor en la figura 9 donde puede verse que funciona como una cinta de transporte que avanza a través de toda la cadena con solo algunas interacciones lineales menores, por lo que la información fluye a lo largo del estado sin cambiar demasiado. Mientras que las otras flechas casi imperceptibles en esta última figura funcionan como puerta que interactuan con el estado celda observando que información es realmente relevante para permitir su paso.

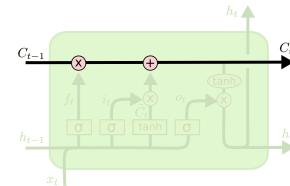


Figura 9: Estado celda de una red LSTM.

El primer paso es nuestra red LSTM es decidir que información vamos a conservar para que continue su camino a lo largo del estado celda. Esta decisión se logra usando una capa sigmoide llamada la capa de la puerta del olvido. Esta capa puede verse en la figura 10 donde la información es discriminada al hacerse uso de una función sigmoide. Después de esto, la información que sobrevivio a este proceso entra a una nueva capa donde nuevamente es procesada por una función sigmoide y reconfigurada después en otra capa con una función tangente hiperbólica con el fin de crear un vector de nuevos valores candidatos como se observa en la

figura 11.

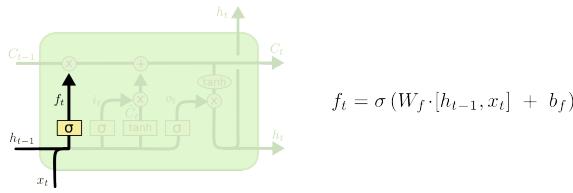


Figura 10: Capa sigmoide

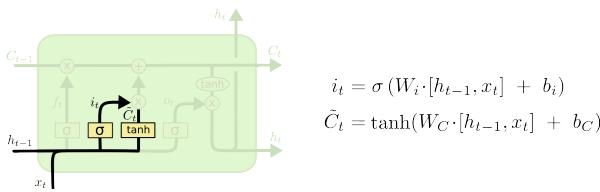


Figura 11: Capas sigmoide y Tanh

Por último este nuevo vector se filtra de nuevo usando una combinación de función de activación sigmoide y tangente hiperbólico como puede observarse en la figura 12.

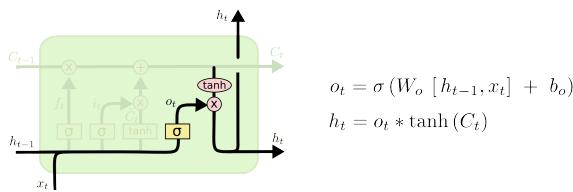


Figura 12: Capas finales sigmoide y Tanh

Figura 13: En la esquina superior izquierda se tiene una placa con desgaste, en la esquina superior derecha una imagen con oclusión (tornillo cubriendo los caracteres de la placa). En la imagen de la esquina inferior izquierda se tiene una imagen desenfocada y en la imagen de la esquina inferior derecha una imagen con cambios de iluminación sobre la placa.

En las figuras 14 y 15 se muestran algunos de los procesos y resultados de los métodos usados para la detección de placas.



Figura 14: Imagen original de uno de los automóviles e imagen al aplicarsele el filtro de Canny.



Figura 15: Contornos encontrados usando el método de Casada de Haar.

¹http://www.medialab.ntua.gr/research/LPRdatabase/Still_images/day/day_very_close_view.zip

En la realización del proyecto se obtuvieron resultados tanto de la detección de la placa, como resultados del OCR realizado sobre el texto de la placa. Sin embargo, debido a la variabilidad en las imágenes, las placas obtenidas presentaban tambien dicha variabilidad por lo que se optó por combinar los resultados obtenidos por ambos métodos de lectura OCR para establecer las metricas de desempeño.

Cabe aclarar que aun cuando todos los métodos se usaran al mismo tiempo para el reconocimiento de una placa el proceso es muy rapido y puede inclusive ser usado para reconocimiento en tiempo real.

IV-A. Resultados Detección placa

Para esta etapa se realizo una inspección manual sobre las subregiones de imagen que se obtienen al aplicar cada uno de los metodos de detección de placas (Canny,Threshold opcion1, Threshold opcion2, Cascadas de Haar) y se categorizo como 1 si en alguno de los contornos obtenidos se obtenian todos los caracteres de la placa, como 0.5 en caso de obtener al menos la mitad de los caracteres de manera legible y como 0 en cualquier otro caso.

Realizando este proceso de inspección manual se obtuvieron los siguientes resultados:

Metodo	Canny	Thresholding 1	Thresholding 2	Haar
Precisión	0.893	0.409	0.647	0.610

Tabla I: Resumen de resultados sobre la detección de placas.

Al combinarse todos los metodos previos se puede obtener una precisión de detección de la placa del **0.991**, lo cual equivale a solo no obtener una placa completa en las 122 imágenes.

IV-B. Resultados OCR texto placa

Para obtener la precisión en lo que respecta al reconocimiento de los caracteres de la placa se optó por usar una métrica de cadenas de texto (Distancia Levenshtein) para estimar la similaridad entre los caracteres en la placa de la imagen original y los obtenidos con los métodos de detección y lectura de caracteres.

La distancia Levenshtein o distancia de medición se define como el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra, se usa ampliamente en teoría de la información y ciencias de la computación. En este sentido se debe entender por operación, a bien una inserción, la eliminación o la sustitución de un carácter.

Dado que queremos medir que tan cerca se encuentra nuestra predicción del texto de la placa se calcula una metrica de similaridad, haciendo uso de la distancia levenshtein (para normalizar la distancia entre 0 y 1)

$$\text{Similaridad} = (\text{largo_placa} - \text{dist_lev})/\text{largo_placa}$$

Se usaron los resultados combinando de los metodos de detección de placas previos para poder tener todas las placas posibles. Como resultado de la variabilidad presente en la detección de las placas y la variabilidad en las imágenes mencionada anteriormente se opto por combinar los resultados obtenidos por knn y Tesseract para realizar el calculo de las dos métricas descritas anteriormente.

Haciendo lo anterior se obtuvo que el numero de placas para las cuales se detecto una cadena que tiene a lo mas (distancia levenshtein 3) es **66** de las 121 placas detectadas por los tres métodos, lo cual representa poco mas de la mitad de las placas del conjunto probado.

Para la segunda métrica se considero tomar la media de los valores obtenidos de ella para cada lectura de la placa, con esto se obtuvo un promedio de **0.409**. En las siguientes tablas se muestran algunos resultados de la lectura de las placas con respecto a la segunda métrica de similaridad.

Archivo	Correcto	Predicho	Similaridad
HPIM1118.JPG	YHT5335	YHT5335	1.000000
HPIM1249.JPG	YHB3424	YHB3424	1.000000
HPIM0645.JPG	MIE8817	MIE8817	1.000000
HPIM0676.JPG	YBX4482	YBX4482	1.000000
HPIM0707.JPG	YXZ5674	YXZ5674	1.000000
HPIM0677.JPG	MIA7795	MIA7795	1.000000
HPIM1242.JPG	YEP8932	YEP8932	1.000000
HPIM1244.JPG	YEAE8054	YEAE8054	1.000000
66.JPG	YMNT7927	YMNT7927	1.000000
65.JPG	YMNT7927	YMNT7927	1.000000
PHTO0051.JPG	YKK2252	YKK2252	1.000000
HPIM0719.JPG	MIZ3540	MIZ3540	1.000000

Tabla II: Primeras 10 imágenes para las que se encontro la mayor similaridad en la lectura de las placas.

Archivo	Correcto	Similaridad
PHTO0072.JPG	MIA2957	0.0
HPIM0784.JPG	MIB6822	0.0
HPIM0785.JPG	YYT9659	0.0
HPIM1170.JPG	IBZ6378	0.0
HPIM1211.JPG	MIM9621	0.0
HPIM0925.JPG	YBX4482	0.0
HPIM0933.JPG	TK1688	0.0
HPIM0965.JPG	MIZ7793	0.0
HPIM1004.JPG	ZMA2474	0.0
HPIM1039.JPG	IZT2313	0.0

Tabla III: Primeras 10 imágenes para las que se encontro la peor similaridad en la lectura de las placas. En este casos se muestra que ninguno de los algoritmos de OCR fue capaz de extraer el texto ni siquiera de forma parcialmente correcta.

V. CONCLUSIONES

Existen varias formas de resolver parcialmente el problema del reconocimiento debido a que se pueden presentar muchas variaciones en las imágenes aun cuando se tenga una cámara fija para solo obtener proyecciones lineales y no tener tanta variabilidad. Variaciones como cambios de intensidad de luz en las placas así como la presencia de occlusiones que pueden llevar a los métodos tradicionales a fallar.

En este trabajo se presentaron tres formas distintas para solucionar inicialmente el problema de la detección de placas en el cual el enfoque usando Adaptive Thresholding-Canny mostró los mejores resultados. Sin embargo, aun pueden alcanzarse a lograr mejoras con el método de Cascada de Haar pero debido a que toma un tiempo considerable para su entrenamiento no fue posible considerar mas combinaciones de parámetros por lo que se considero solo el conjunto de parámetros que mejores resultados mostró.

Aunque el Global Thresholding es un método muy básico de procesamiento de imágenes digitales mostró resultados considerablemente buenos. Por parte de los métodos de OCR se encontró que las variaciones por rotaciones, occlusiones (ej. tornillos), desvanecimientos e incluso variaciones en el tamaño de la placa pueden afectar demasiado su desempeño.

Aun cuando se encontraron resultados no tan alentadores incluso combinando ambos métodos de OCR se debe considerar que debido a la variación en las imágenes de las placas obtenidas, el resultado es considerablemente bueno, y cuando se encuentran las placas de frente ambos métodos logran excelentes resultados.

REFERENCIAS

- [1] Louka Dlagnekov. License plate detection using adaboost. *Computer Science and Engineering Department, San Diego*, 2004.
- [2] Rafael C Gonzalez, Richard E Woods, et al. Digital image processing, 2002.
- [3] Wing Teng Ho, Wooi Hen Yap, and Yong Haur Tay. Learning-based license plate detection on edge features. In *Proceedings of the 10th MMU International Symposium on Information and Communications Technologies*, 2007.
- [4] Ana Riza F Quiros, Rhen Anjerome Bedruz, Aaron Christian Uy, Alexander Abad, Argel Bandala, Elmer P Dadios, Arvin Fernando, and De La Salle. A knn-based approach for the machine vision of character recognition of license plate numbers. In *Region 10 Conference, TENCON 2017-2017 IEEE*, pages 1081–1086. IEEE, 2017.
- [5] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

VI. ANEXO

VI-A. Tecnologías utilizadas

- Sistema Operativo Linux, Fedora 27, Ubuntu 16.04.5 LTS, Ubuntu 18.04 LTS
- Python 3.5, Python 3.6.5, Python 3.6.6
- opencv-3.2.0-15, opencv-3.2.0-15

VI-B. Métodos de la clase principal (*ReconocePlaca*)

```
def encuentra_placas(self, tipo_prep = "canny"):  
    # Argumentos:  
    # tipo_prep: Canny, Thresh tipol ,  
    # Thresh tipo2, Haar  
  
    def redim_placa(self, x, y, w, h):  
        return imgCropped  
  
    def elige_texto(self, text):  
        return placa_text  
  
    def prep_haar(self):  
  
    def prep_knn(self, x, y, w, h):  
        return plate_chars  
  
    def placa_ocr(self, tipo_ocr = "tesseract"):  
        self.possible_textos = lista_texto
```

VI-C. Código de ejemplo

```
import platerecog as pr  
  
wdir = "data/"  
filename = wdir + "33.JPG"  
  
# Leemos imagen  
recog = pr.ReconocePlaca(filename)  
  
# Buscamos posibles placas  
recog.encontrar_placas(tipo_prep = "canny")  
  
# Buscamos posibles textos  
  
print("Con Tesseract")  
recog.placa_ocr(tipo_ocr = "tesseract")  
  
for texto in recog.possible_textos:  
    print(texto)  
  
print("Con KNN")  
recog.placa_ocr(tipo_ocr = "knn")  
  
for texto in recog.possible_textos:  
    print(texto)
```

VI-D. Códigos empleados

Con la finalidad de ordenar mejor las funciones programadas se decidió hacer uso de POO y tener una clase global:

- platerecog.py
- knn2.py

Códigos utilizados en las pruebas:

- deteccion_placas.ipynb
- Encuentra_chars.ipynb
- Encuentra_placas.ipynb
- deteccion_placas.ipynb
- unittests.py
- test_detect_all_images.py
- recorta_placas.py

Código para obtención de resultados (sección IV):

- resultados_deteccion.ipynb

VI-E. Datos

Las imágenes que se usan de fuente se tienen en formato JPG, los preprocesos necesarios se realizan dentro de los métodos prep de la clase principal, entre los procesamientos que se hacen es redimensionamiento, *deblurring*, recortes, *thresholding* binarios , detección de bordes; las imágenes que se generan en esta etapa se guardan para fines ilustrativos dado que al realizar el procesamiento no es necesario preservarlas.



Figura 16: Imágenes de muestra tomadas de la base de datos original.

VI-F. Diagrama del Sistema de Detección Automática de Placas

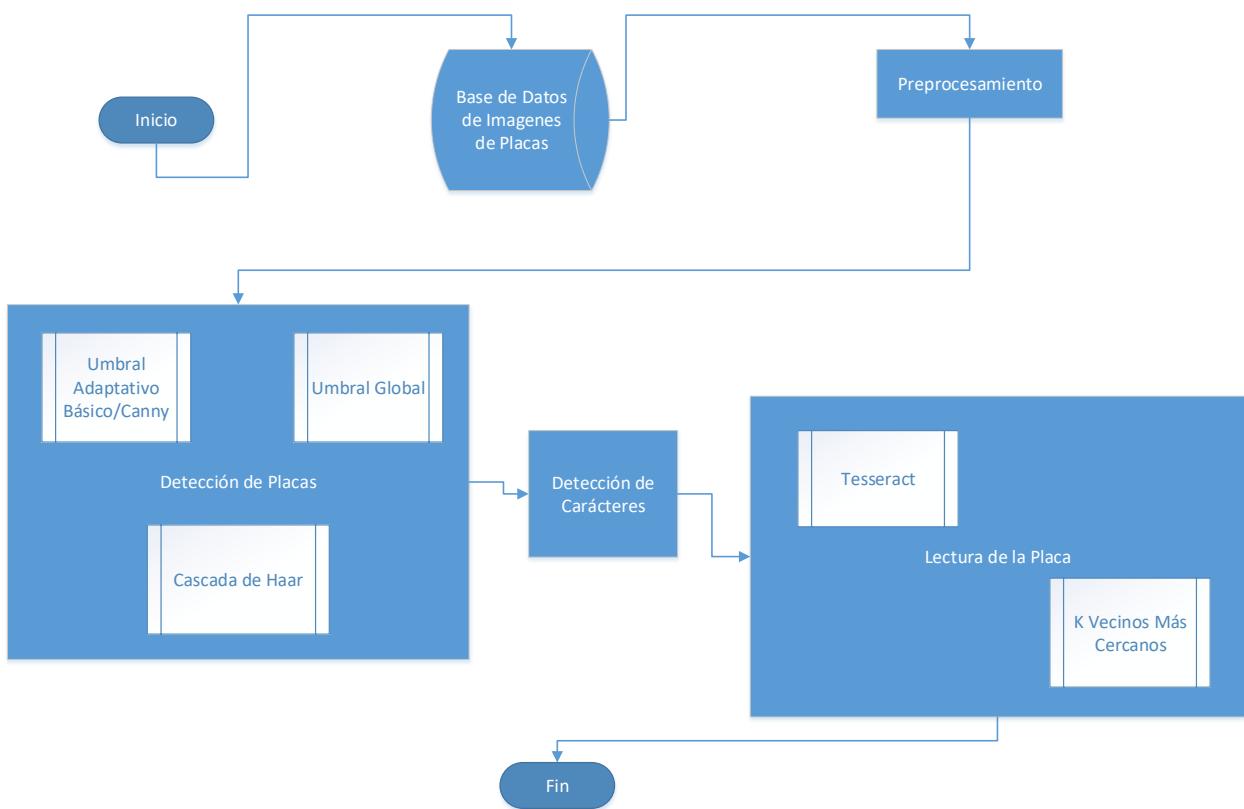


Figura 17: Diagrama general para el sistema de detección automática de placas.