



Universidad
Señor de Sipán

**FACULTAD DE INGENIERÍA, ARQUITECTURA Y
URBANISMO**

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS

**Desarrollo de un método de reconocimiento de
placas vehiculares de mototaxi en ambientes no
controlados utilizando procesamiento digital de
imágenes**

**PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO
DE SISTEMAS**

Autor

Bach. Oliva Villalobos Carlos Humberto

ORCID: <https://orcid.org/0000-0001-6355-1537>

Asesor

Mg. Bances Saavedra David Enrique

ORCID: <https://orcid.org/0000-0002-7164-8918>

Línea de Investigación

**Ciencias de la información como herramientas multidisciplinares
y estratégicas en el contexto industrial y de organizaciones**

Sublínea de Investigación

Informática y transformación digital en el contexto industrial y

Organizacional

Pimentel – Perú

2025




DECLARACIÓN JURADA DE ORIGINALIDAD

Quien suscribe la DECLARACIÓN JURADA, soy Oliva Villalobos Carlos Humberto del Programa de Estudios de **Ingeniería de Sistemas** de la Universidad Señor de Sipán, declaro bajo juramento que soy autor del trabajo titulado:

DESARROLLO DE UN MÉTODO DE RECONOCIMIENTO DE PLACAS VEHICULARES DE MOTOTAXI EN AMBIENTES NO CONTROLADOS UTILIZANDO PROCESAMIENTO DIGITAL DE IMÁGENES

El texto de mi trabajo de investigación responde y respeta lo indicado en el Código de Ética de la Universidad Señor de Sipán, conforme a los principios y lineamientos detallados en dicho documento, en relación con las citas y referencias bibliográficas, respetando el derecho de propiedad intelectual, por lo cual informo que la investigación cumple con ser inédito, original y autentico.

En virtud de lo antes mencionado, firman:

Oliva Villalobos Carlos Humberto	DNI: 40522099	
----------------------------------	---------------	---

Pimentel, 20 de enero del 2025

REPORTE DE SIMILITUD TURNITIN

Oliva Villalobos Carlos Humberto

Desarrollo de un método de reconocimiento de placas vehiculares de mototaxi en ambientes no controla

Universidad Señor de Sipan

Detalles del documento

Identificador de la entrega
trn:oid::26396:424749790

Fecha de entrega
29 ene 2025, 4:42 p.m. GMT-5

Fecha de descarga
29 ene 2025, 4:47 p.m. GMT-5

Nombre de archivo
TURNITIN OLIVA VILLALOBOS-INFORME FINAL DE INVESTIGACIÓN-TITULO.docx

Tamaño de archivo
7.0 MB

54 Páginas

10,593 Palabras

57,257 Caracteres



Página 2 of 61 - Descripción general de integridad

Identificador de la entrega trn:oid::26396:424749790

11% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Filtrado desde el informe

- Bibliografía
- Texto mencionado
- Coincidencias menores (menos de 8 palabras)

Fuentes principales

- 10% Fuentes de Internet
- 2% Publicaciones
- 7% Trabajos entregados (trabajos del estudiante)

Marcas de integridad

N.º de alerta de integridad para revisión

- Texto oculto**
713 caracteres sospechosos en N.º de páginas
El texto es alterado para mezclarse con el fondo blanco del documento.

Los algoritmos de nuestro sistema analizan un documento en profundidad para buscar inconsistencias que permitirían distinguirlo de una entrega normal. Si advertimos algo extraño, lo marcamos como una alerta para que pueda revisarlo.

Una marca de alerta no es necesariamente un indicador de problemas. Sin embargo, recomendamos que preste atención y la revise.

**DESARROLLO DE UN MÉTODO DE RECONOCIMIENTO DE PLACAS
VEHICULARES DE MOTOTAXI EN AMBIENTES NO CONTROLADOS
UTILIZANDO PROCESAMIENTO DIGITAL DE IMÁGENES**

Aprobación del jurado

DR. ATALAYA URRUTIA CARLOS WILLIAM
Presidente del Jurado de Tesis

MG. BANCES SAAVEDRA DAVID ENRIQUE
Secretario del Jurado de Tesis

MG. ASENJO CARRANZA ENRIQUE DAVID
Vocal del Jurado de Tesis

Dedicatorias

Dedico con todo mi corazón a mis padres: Rafael Oliva y Domitila Villalobos, ya que sin ellos no habría logrado este propósito. Sus bendiciones a lo largo de mi vida, me protegen y me llevan por el camino correcto, pues mucho de mis logros se los debo a ellos, por eso les doy mi trabajo en ofrenda por su paciencia y amor.

Carlos Humberto Oliva Villalobos.

Agradecimientos

A todos y cada uno de los docentes quienes a lo largo de mi formación académica supieron transmitir sus conocimientos y valores.

Al Mg. Heber Iván Mejía Cabrera, quien contribuyo con su conocimiento, experiencia y motivación, a que culmine con éxito la presente Tesis. Mi más sinceros y eterno agradecimiento a cada uno de ellos.

Carlos Humberto Oliva Villalobos.

ÍNDICE

Resumen.....	viii
Abstract.....	ix
I. INTRODUCCIÓN.....	10
II. MATERIALES Y MÉTODO.....	24
III. RESULTADOS Y DISCUSIÓN.....	52
3.1 Resultados.....	52
3.2 Discusión.....	60
IV. CONCLUSIONES Y RECOMENDACIONES.....	61
4.1 Conclusiones.....	61
4.2 Recomendaciones.....	62
REFERENCIAS.....	64
ANEXOS.....	69

DESARROLLO DE UN MÉTODO DE RECONOCIMIENTO DE PLACAS VEHICULARES DE MOTOTAXI EN AMBIENTES NO CONTROLADOS UTILIZANDO PROCESAMIENTO DIGITAL DE IMÁGENES

Resumen

Los vehículos menores, como las mototaxis, son un medio de transporte público utilizado como herramienta de trabajo por muchos peruanos. Sin embargo, debido a la informalidad y la imprudencia de los conductores, aumentaron las infracciones de tránsito y los robos de estas unidades, lo que generó un problema ligado al monitoreo vial y la seguridad ciudadana. Existen diversas soluciones de ingeniería, como sistemas electrónicos y software comerciales, que procesan información para el reconocimiento de placas, pero solo de automóviles. En el Perú, aunque hay placas de mototaxi, no se desarrolló un método estandarizado para el reconocimiento de sus dígitos en ambientes no controlados. Para abordar esta problemática, se propuso un método de reconocimiento de placas de mototaxi. Se capturaron 150 imágenes con un dispositivo móvil montado en un trípode. Durante el preprocesamiento, las imágenes fueron convertidas de RGB a escala de grises y HSV, extrayéndose el canal de intensidad. Se aplicaron operaciones morfológicas como TOPHAT y BLACKHAT para resaltar detalles, seguidas de un filtro gaussiano para mitigar variaciones de ángulos y ruidos, además de una umbralización adaptativa. En la fase de segmentación, se utilizaron las funciones `findContours` y `boundingRect` de OpenCV para detectar contornos, y el método OTSU para convertir las placas recortadas en imágenes binarias. Finalmente, se empleó el algoritmo K-Vecinos más cercanos (KNN) para clasificar los caracteres. El método alcanzó una precisión del 99.30% y una exactitud del 95.13%, demostrando su viabilidad en entornos no controlados.

Palabras clave: Procesamiento de imágenes, filtro gaussiano, operaciones morfológicas, segmentación de caracteres, algoritmo de clasificación, KNN, placas de rodaje de mototaxi.

Abstract

Smaller vehicles, such as mototaxis, are a means of public transportation used as a work tool by many Peruvians. However, due to the informality and recklessness of drivers, traffic violations and thefts of these units have increased, creating an issue related to road monitoring and public safety. Various engineering solutions, such as electronic systems and commercial software, process information for license plate recognition, but only for automobiles. In Peru, although mototaxis have license plates, no standardized method has been developed for recognizing their digits in uncontrolled environments. To address this issue, a mototaxi license plate recognition method was proposed. A total of 150 images were captured using a mobile device mounted on a tripod. During preprocessing, the images were converted from RGB to grayscale and HSV, extracting the intensity channel. Morphological operations such as TOPHAT and BLACKHAT were applied to enhance details, followed by a Gaussian filter to mitigate angle variations and noise, along with adaptive thresholding. In the segmentation phase, OpenCV's findContours and boundingRect functions were used to detect contours, and the OTSU method was applied to convert the cropped plates into binary images. Finally, the K-Nearest Neighbors (KNN) algorithm was used to classify the characters.

The method achieved an accuracy of 99.30% and a precision of 95.13%, demonstrating its feasibility in uncontrolled environments.

Keywords: Image processing, Gaussian filter, morphological operations, character segmentation, classification algorithm, KNN, mototaxi license plates

I. INTRODUCCIÓN

Existen diversos medios de transporte público, en las ciudades de todo el mundo; entre ellas se tiene el metro, el tren, los buses, los sistemas integrados de transporte como el metropolitano y en las ciudades de países como Filipinas, Indonesia, Malasia, China y la India existen un medio de transporte menor, como la mototaxi.

Las primeras unidades de mototaxis se fabricaron en Tailandia en los años cincuenta, asimismo en la década de los ochenta el Perú fue el primer país de latinoamérica en utilizar este vehículo para el transporte público, el uso se popularizó rápidamente y se le llamó mototaxi [1].

En el año 2019, se vendieron un total de 118, 818 de trimotos a nivel nacional, las marcas más populares fueron Wanxin, Honda, Yansumi, Ssenda, Ronco [2].

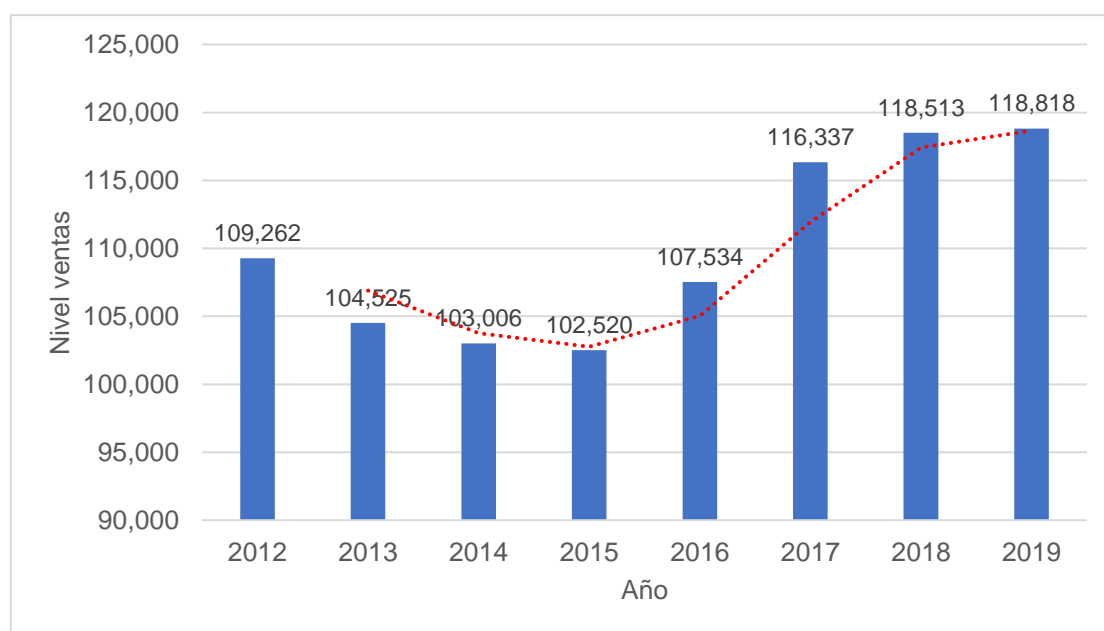


Figura 1. Venta de vehículos trimotos. Fuente: Elaboración propia.

A nivel nacional circulan alrededor de 900 mil mototaxis, de las cuales la mitad de estas unidades son localizadas en la ciudad de Lima y el resto en otras regiones del país [3]. Los ingresos diarios de un mototaxista propietario de su vehículo, bordea entre 80 y 120 soles, por lo tanto, al mes la ganancia sería entre 2,400 y 3,600 soles [4]. Siendo esto

un medio de trabajo para personas que buscan un mercado laboral formal y una solución a las necesidades de transporte público de muchos peruanos [3].

En la Región Lambayeque existen más de 80 mil mototaxis, algunas de estas unidades circulan con imprudencia y otros de manera informal, por esta razón el incremento de infracciones de tránsito, como el desacato a las luces del semáforo, no tener placa, exceso de velocidad y los robos de estas unidades también se ha incrementado, son problemas ligados al monitoreo vial y seguridad ciudadana, que hace complicado reconocer al infractor por el embotellamiento que genera la cantidad de vehículos que transitan en las calles [3].

En consecuencia en el año 2022 fueron robados un total de 21, 240 vehículos a nivel nacional, cifra superior al año anterior en 3 %, asimismo 5,505 fueron mototaxis [5].

En lo que data a los accidentes de tránsito la Policía Nacional del Perú (PNP), reportó un total de 83, 897 casos, incrementando así la tasa en un 2.14% con respecto al año anterior, siendo el departamento de Lambayeque uno de los afectados con 4, 608 accidentes de tránsito, de la cual 632 fueron a causa de las mototaxis [6].

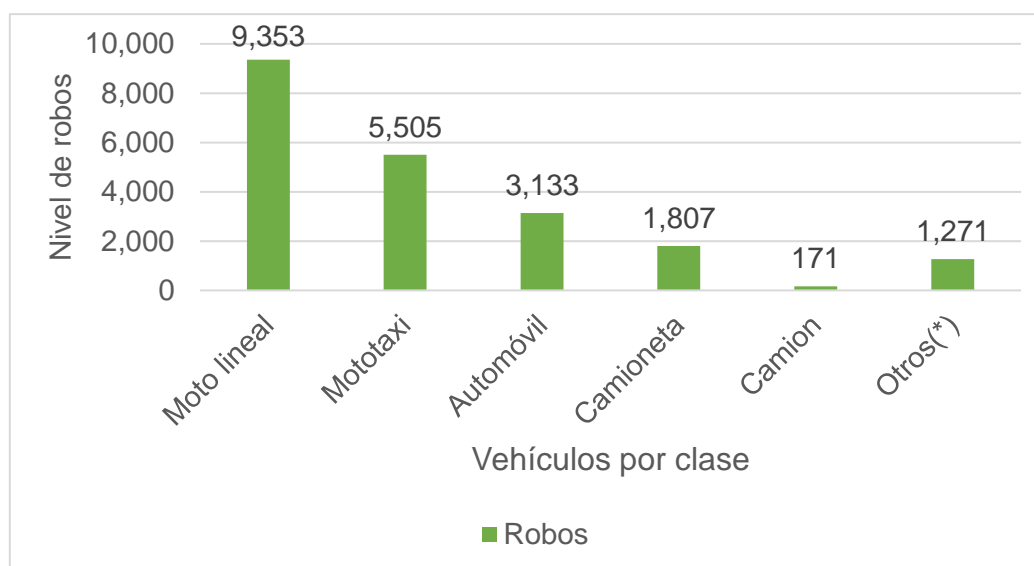


Figura 2. Vehículos robados por clase. Fuente: Elaboración propia.

Existen diversas soluciones de ingeniería, para solucionar las dificultades mencionadas anteriormente, como las fotopapeletas, que se les denominan a las papeletas generadas por medios electrónicos calibrados, que capturan la imagen del

vehículo al momento de cometer la infracción, otra solución es el cinemómetro, este dispositivo detecta vehículos que sobrepasan el límite de velocidad mediante un radar o láser, que al identificar una infracción el cinemómetro toma una foto de la placa de rodaje del vehículo, generalmente en combinación con una cámara, estos equipos son colocados en lugares fijos, en el sentido de la marcha del vehículo, con el fin que el láser capte la velocidad del objetivo, emitiendo una ráfaga de luz infrarroja a la unidad, estos pulsos son emitidos a intervalos calibrados de tiempo, de esta forma se detecta la velocidad del infractor [7]. Del mismo modo también existen sistemas de reconocimiento automático de placas (ANPR), estos sistemas utilizan algoritmos de visión por computadora y técnicas de machine learning que permiten la extracción automática de caracteres de las placas a partir de imágenes o videos para detectar infractores, en la carretera, así como también los sistemas comerciales de control de tránsito y vigilancia, el cual las empresas ofrecen soluciones de vigilancia integradas que incluyen reconocimiento de placas, estos sistemas comerciales combinan hardware y software de procesamiento avanzado para ofrecer una solución integral. Sin embargo, en el Perú existen placas de mototaxi, pero no existe un método estándar de ingeniería para identificar y reconocer caracteres en placas de rodaje de mototaxi en ambientes no controlados, como el ángulo, la forma y la textura, etc. Por esta razón, [8], según los autores lograron una precisión impresionante de 99,8% para la detección de la región de la placa del automovil, sin embargo, para diseñar un sistema de reconocimiento de placa robusto en futuras investigaciones solicitan tener en cuenta el reconocimiento de placas nocturnas con menor luminosidad, menor resolución e identificación de matrículas de diferentes formas y estilos, la detección de vehículos a excesiva velocidad, la identificación de caracteres alfanuméricos ambiguos, el reconocimiento de placas sesgadas y borrosas, la oclusión del vehículo y la identificación de placas en diversas situaciones peligrosas, es decir, lluvia o nieve. Según [9], existen problemas y desafíos frecuentes en el reconocimiento de placas vehiculares como la calidad de la imagen, algunas de baja calidad debido a las técnicas de compresión o la interferencia del entorno, que generan dificultades en el reconocimiento

de caracteres de la placa, también la nitidez para un reconocimiento preciso, y los sistemas con cámaras de baja resolución, ya que incrementan el margen de error, otra dificultad es por la velocidad del vehículo, es decir las captura de imágenes pueden ser borrosas o incompletas, especialmente si las cámaras no están equipadas con obturadores rápidos o con tecnología de alta velocidad, también los obstáculos visuales, como la suciedad en las placas, nieve, o incluso cosas colgantes en los vehículos que interfieren parte de la placa y hacer que el software no logre identificar el número completo, otro desafío el reconocimiento en tiempo real, ya que al procesar la data en tiempo real es una dificultad para el hardware, en ambientes de tráfico con múltiples carriles, se requiere un procesamiento más rápido y eficiente, un problema no técnico la privacidad y protección de la data, ya que genera inquietudes sobre el uso ilegal de información íntima, lo que ha llevado a restricciones estrictas en algunos países .

Trabajos previos en el ámbito del reconocimiento automático de matrículas (ALPR) han desarrollado métodos efectivos que pueden ser aplicados para abordar esta problemática. Por ello [10], realizó la investigación, Car License Plate Segmentation And Recognition System Based On Deep Learning, en IT & communication Center, University of Basrah, Basrah, Iraq. El problema del reconocimiento automático de matrículas (ALPR). Por esta razón, presentó el método de reconocimiento de matrículas de automóviles iraquíes, basado en técnicas de aprendizaje profundo de redes neuronales convolucionales (CNN). El método propuesto presentó 4 etapas, en la segmentación de matrículas se usaron técnicas de procesamiento de imágenes para cortar la matrícula vertical y horizontalmente, detección de bordes y búsqueda de contornos para dividir la matrícula en cuatro partes: izquierda, número inferior y gobernación de la placa iraquí. En la etapa de separación de caracteres primero se cambió de tamaño, se modificó a escala de grises, luego se aplicó un filtro bilateral y también detección de bordes con el algoritmo Canny, asimismo, se aplicó técnicas morfológicas para encontrar contornos y eliminar líneas no deseadas, luego se aplicó rectángulos alrededor de los objetos detectados para segmentar los caracteres. En la fase de reconocimiento de caracteres se usó redes

neuronales convolucionales (CNN) para el reconocimiento de números y letras, conjunto de datos que incluyó 2, 000 imágenes divididas en 22 clases (números del 0 al 9 y letras árabes), arquitectura CNN compuesta por capas convolucionales, capas de agrupamiento máximo y capas densas, utilizando la función de activación ReLU. Resultados, el método propuesto alcanzó una tasa de éxito en el reconocimiento de números, letras y palabras de 98 %, la eficiencia del método se probó con 500 imágenes de placas, incluyendo algunas en ángulos inclinados, y se obtuvieron buenos resultados en términos de precisión y eficiencia, a pesar de algunos casos difíciles, como imágenes distorsionadas o tomadas desde ángulos complicados, el sistema demostró robustez y precisión. [11], realizó la investigación, Automatic Number Plate Recognition Of Saudi License Car Plates, en Department of Computer Science Umm Al-Qura University Makkah, Saudi Arabia. El reconocimiento automático de placas de vehículos en Arabia Saudita, utilizando técnicas de procesamiento de imágenes. Por esta razón, presentó como método de solución en la etapa del preprocesamiento, las imágenes se convirtió a escala de grises y se aplicaron técnicas de interpolación para mejorar la calidad, en la detección de bordes se utiliza el algoritmo Canny para identificar los bordes en la imagen. En la etapa de segmentación, se aplicó un perfil de proyección horizontal para segmentar la matrícula. En la etapa de reconocimiento de la placa, reconocimiento óptico de caracteres (OCR), para extraer letras y números de las imágenes procesadas, luego se reformateo para las letras árabes, se aplicó un re-shaper para asegurar que se reconozcan correctamente, los resultados del reconocimiento se superponen en las imágenes originales para visualizar las placas detectadas. Resultados, el método propuesto logró una precisión del 92,4 % para caracteres árabes y del 96 % para caracteres en inglés. Esto demuestra la efectividad del enfoque utilizado en el reconocimiento automático de placas en condiciones variadas. Este enfoque integral combina técnicas avanzadas de procesamiento de imágenes con algoritmos robustos para lograr un sistema eficiente y preciso en el reconocimiento de placas vehiculares. [12], realizó la investigación, Barrier Access Control Using Sensors Platform and Vehicle License Plate Characters Recognition, en la University Islamabad

de Pakistan. Las variaciones en el fondo de la placa, las fuentes y las deformaciones que causan problemas de reconocimiento. Por esta razón, presentó software de control de acceso preciso, que usa una plataforma de sensor y reconocimiento de matrícula de vehículo. En la fase de adquisición: cuando los sensores ultrasónicos detectan el vehículo en un rango específico de distancia, la cámara se activa a distancia de 1 a 3 metros, además en la fase de preprocesamiento: se convierte la imagen capturada a escala de grises para reducir la complejidad y el proceso, luego se empleó un detector de bordes el cual se utilizó el operador Canny para identificar los contornos en la imagen, para suavizar filtro gaussiano que a la vez reduce el ruido, en la imagen de borde generada, en la etapa de segmentación, se realizó varias operaciones morfológicas como dilatación, erosión horizontal, erosión vertical y relleno de agujeros, se binariza usando el algoritmo de umbral, el cual primero se calcula el histograma de intensidad de la imagen de la región de la placa (LP), y luego se localiza los dos picos más altos en este histograma, además se convierte la imagen en escala de grises de la LP a binario. En la fase de extracción de placa (LP) a través del recuento de componentes conectados, se encuentra los componentes del cuadro de delimitación rectangular, basados en componentes de conectividad en la imagen erosionada. En la fase de clasificación: se emplea el algoritmo método k vecinos más cercanos (KNN). En la fase de reconocimiento: el reconocimiento óptico de caracteres (OCR), se usó para reconocer las letras y los dígitos de la LP extraída, las características del personaje son zonificación, perímetro, grado, número de Euler, resumen de píxeles de filas y columnas particulares. Resultados, se asigna etiquetas de clase 0 - 9; 10 - 35 para los dígitos y para los alfabetos A - Z, para reconocer los caracteres LP, se extrajeron un total de 3,643 caracteres de imágenes de LPs computarizados y escritos a mano. Los resultados obtenidos del sistema propuesto, muestra una precisión en la extracción del 98%, para la segmentación 96% y para el reconocimiento 93%. Conclusión, el sistema propuesto detecta automáticamente un vehículo en una entrada a través de sensores ultrasónicos y luego lo reconoce mediante el sistema basado en imágenes de su placa, que puede tener varios fondos, fuentes y

estilos, además se llegó a determinar cuál tenía la mejor tasa de reconocimiento. [13], realizó la investigación, Research on Recognition Technology of License Plate Image, en la Xi'an Technological University de China. El problema principal radica en la precisión y eficiencia del reconocimiento de caracteres en condiciones diversas, como variaciones de iluminación y ángulos de inclinación. Por esta razón, se presentó un clasificador de caracteres mediante un método de uno a muchos. En la etapa de preprocesamiento: el espacio de color RGB se convierte en el espacio de color HSI, para obtener la región candidata. En la fase de procesamiento: se trabaja en la escala de grises, la binarización, el suavizado de la imagen y corrección de inclinación de la matrícula, se utiliza el operador canny para la detección de bordes. En la etapa de segmentación: cuando el umbral es mayor que el valor de proyección, se ajusta a cero, la matriz se escanea de izquierda a derecha, para almacenar el valor de la proyección vertical. En la etapa de reconocimiento: utiliza análisis de topología y geometría, los caracteres se refinan para obtener el esqueleto, como horizontal, vertical, puntos y otras características de la dirección del trazo, el número de Euler es un análisis topológico que representa la conectividad de los caracteres. En la etapa de extracción de caracteres: el propósito de la normalización es extraer las características de tamaño de la licencia, ajustar uniformemente el tamaño de la imagen de muestra a 30 x 30, se establece el fondo en negro y el valor de píxel es 0, asimismo el primer plano en blanco, el valor de píxel es uno, se escanea el número de píxeles de cada fila o columna y se divide entre treinta para obtener la característica de 306 caracteres de cada fila o columna. En la etapa de clasificación: se utilizó, el clasificador de la máquina de vectores de soporte (SVM). Resultados, la tasa de reconocimiento propuesto es de 92,6 % comparado con el método de coincidencia de plantillas de flab 89 % y el método de coincidencia de redes neuronales 91,8 %. El método de uno a muchos se utiliza para identificar los personajes en varias clases. Conclusión, el método propuesto tiene mayor precisión. [14], realizó la investigación, Design and Implementation of Vehicles Identification and Tracking System, en la University of Baghdad de Irak. Soluciones de reconocimiento de matrículas, son ineficientes porque se

ejecutan manualmente y carecen de automatización. Por esta razón, el método propuesto consiste en una serie de subsistemas de reconocimiento de placas distribuidas que son compatibles con las cámaras. En la fase de adquisición: por medio de un sensor es detectado el vehículo y se activa la cámara para hacer la toma respectiva. Para la detección de matrículas los autores propusieron 2 pasos, el paso de ubicación aproximada: la cual se basa en una técnica moderna de detección de objetos y el paso de ubicación precisa: la cual se basa en técnicas de procesamiento. En la etapa de procesamiento: la imagen es convertida a escala de grises, además para detectar bordes se usó el algoritmo Canny, se aplicó el algoritmo de contorno para detectar objetos cerrados, para eliminar la distorsión se usó la homografía planar. En la fase de segmentación de caracteres: se aplica un filtro bilateral a la imagen resultante para mejorarla y eliminar ruidos, además se utiliza un umbral adaptativo para binarizar la imagen, se realiza la operación de contorno y se calcula un cuadro delimitador, encontramos la región de interés que contiene solo caracteres usando coordenadas de recuadros delimitadores. Para la clasificación de caracteres: se empleó vecinos k-más cercanos (KNN); para el entrenamiento se usaron un total de 25 muestras de entrenamiento para cada categoría, las muestras de entrenamiento son imágenes de personajes que se redimensionan a 20 x 30 píxeles. Resultados, el método tiene una precisión de detección 95%, precisión de segmentación 91%, exactitud de reconocimiento 89%, tiempo total de procesamiento 2 - 4 segundos. Conclusiones, para diseñar e implementar un sistema automático de identificación y seguimiento de vehículos, se empleó aprendizaje automático con técnicas de procesamiento de imágenes. [15], realizó la investigación, RESEARCH OF INDONESIAN LICENSE PLATES RECOGNITION ON MOVING VEHICLES, en Department of Computer Engineering Maranatha Christian University 65 Jl. Suria Sumantri, Bandung, Indonesia, 40164. el problema del reconocimiento de matrículas de vehículos en movimiento, lo cual es un desafío debido a la dificultad de capturar imágenes nítidas y claras de las placas en condiciones cambiantes, como baja iluminación y condiciones climáticas adversas como la lluvia. Por

esta razón, se planteó un método de solución de detección de vehículos en movimiento: en la etapa del procesamiento, se selecciona una imagen estática de un vehículo en movimiento utilizando técnicas que analizan la diferencia entre imágenes con y sin vehículos, se aplica un método para mejorar la claridad de las imágenes desenfocadas, lo cual es esencial para el reconocimiento preciso de las matrículas, En la fase de segmentación, se utilizan algoritmos de detección de bordes, como el filtro Sobel, para extraer los caracteres en las matrículas, también se aplican métodos como el análisis de componentes conectados (CCA) y características de textura para mejorar la precisión. Para la etapa del reconocimiento de caracteres se usó redes neuronales para el reconocimiento de caracteres, lo que permite una identificación más precisa incluso cuando hay defectos en los caracteres. Conclusiones, la metodología propuesta logró un éxito del 98.1% en el reconocimiento de matrículas bajo condiciones óptimas. Sin embargo, se controla que la tasa de éxito disminuya en condiciones desfavorables, como mala iluminación o lluvia intensa, la implementación del método de desenfoco mejoró la tasa de reconocimiento en un 1.7% en comparación con métodos que no aplicaron esta técnica, a pesar del alto rendimiento general, todavía existen desafíos significativos relacionados con el ruido ambiental que pueden afectar la precisión del reconocimiento.

[16], realizó la investigación, Parking entrance control using license plate detection and recognition, en la Al-Azhar University de Egipto. Mejorar las técnicas actuales de detección y reconocimiento de placas de automóviles. Por esta razón, se implementó un marco híbrido para un sistema robusto de detección y reconocimiento de matrículas. En la fase de adquisición: Se instala una cámara digital en la entrada del estacionamiento, para recoger la imagen del automóvil. En la etapa de procesamiento: el método de detección depende de dos colores de licencia, fondos blanco y amarillo, la imagen RGB dada se convierte en imagen de índice, luego usa un filtro de dos colores, el rango blanco y el rango amarillo; se elimina los objetos de ruido, usando la función `bwareaopen()` para eliminar objetos menores de 200, usando la operación morfológica tanto para la erosión como para la dilatación. En la fase de segmentación: la imagen de la placa dada se

convierte de escala de grises a blanco y negro (BW), en función de un umbral correspondiente a lo siguiente $Y_{ij}=\{1 \text{ if } x_{ij}>\theta, 0 \text{ otherwise}$, usando dos pasos de filtros, primero eliminar los objetos más pequeños de la imagen binaria y dejar los objetos más grandes, asimismo eliminar los objetos que no están en el mismo rango vertical. En la etapa de reconocimiento: se utilizó para calcular la tasa de correlación entre dos imágenes A y B, $r=\frac{\sum_i \sum_j (A_{ij}-\bar{A})(B_{ij}-\bar{B})}{\sqrt{(\sum_i \sum_j (A_{ij}-\bar{A})^2 * \sum_i \sum_j (B_{ij}-\bar{B})^2)}}$. En la fase de clasificación: se usa el algoritmo máquina de vectores de soporte (SVM), este algoritmo busca el hiperplano de separación máximo, que se define con la distancia máxima entre las tuplas de entrenamiento. Resultados, Se logró un promedio en detección de 97.8%, en segmentación del 98% y en reconocimiento 97%. Conclusión, se propuso un método híbrido para mejorar las técnicas actuales de detección y reconocimiento de placas de automóviles. En caso la imagen sea de gran escala 2000 x 1950 píxeles, los autores proponen la transformación discreta de wavelets (DWT) antes de la etapa de detección, la cual genera 4 coeficientes aproximación, horizontal, vertical, diagonal, además el uso DWT disminuye el tiempo de la etapa de detección de 0.4892 segundos a 0.167, aproximadamente 67% de tasa de tiempo de disminución, aplicando en la segmentación de placas detectadas, el resultado experimental dio una tasa de segmentación correcta del 98%; para el reconocimiento, el patrón con la correlación máxima se usa como el patrón reconocido, en caso de similitud, la función MultiSvmTrain () se utilizó para el entrenamiento dado con un vector de grupo correspondiente y clasifica un conjunto de prueba dado usando la función MultiSvmClassify () de acuerdo con una relación de uno contra todos. [17], en su investigación, Automated System for Defect Identification and Character Recognition using IR Images of SS-Plates, en la International Journal of Recent Technology and Engineering (IJRTE) de la India. La falta de precisión y el alto consumo de tiempo, donde la detección temprana y precisa de defectos es una fase significativa del control vehicular. Por esta razón, presentó un marco de procesamiento de imágenes para la identificación automatizada de defectos superficiales en la placa de acero inoxidable (SS). En la fase de procesamiento: se convierte la imagen a gama de grises,

esta se procesa usando un filtro gaussiano para tratar los problemas tales como conversión, eliminación de ruido, cambio de tamaño de imagen, mejora de calidad y la técnica de ecualización de histograma adaptativo. En la etapa de extracción: el método de extracción candidato consiste en la ponderación de operador laplaciano de Gauss (LoG) multiescala, la mejora de la densidad del borde, la extracción y eliminación de caracteres mediante un filtro morfológico, el operador Sobel en dirección vertical y horizontal para el reconocimiento de borde. En la fase de extracción: a partir de la imagen binaria, se introduce análisis de componentes conectados (CCA), para crear bloques de caracteres, también realiza un carácter simple y robusto de la rotación de la placa SS. En la etapa de segmentación: se usa descriptores de características, importante para la etapa de clasificación final, la cual consta de forma como perímetro, circularidad, elipticidad, compacidad, varianza de distancia desde los puntos del borde hasta el centroide del defecto, solidez, excentricidad y convexidad son utilizados para expresar la irregularidad del borde; característica de textura, muchos descriptores de características se toman para observar la cuantificación de la textura de la región candidata, con la técnica del detector de bordes Canny se consume para tomar el mapa de bordes de la región candidata, para resaltar la característica de densidad de borde, se utiliza el número resultante de píxeles de borde contados. En la fase de clasificación: los objetos defectuosos se clasifican utilizando el modelo SVM-RFE. Resultados, demostraron que el sistema propuesto puede mejorar significativamente la precisión en 94.88% en comparación con otro clasificador, Random Forest (RF) con 92.72%, red neuronal artificial (ANN) con 90.25% y Adaboost con 89.73%. Conclusión, los problemas se producen debido a la inspección manual de defectos, se reducen considerablemente, con la detección temprana y precisa con el nuevo sistema mejorando la precisión en 94.88% en comparación con otro clasificador.

[18], realizó la investigación, An efficient method for extraction and recognition of bangla characters from vehicle license plates, en el país de Bangladesh. La ineficiencia de algoritmos de detección y extracción de la región de interés (ROI) de las imágenes de placa. Por esta razón, presenta un sistema automático para detectar el área de la placa y

reconocer los caracteres. En la fase de preprocesamiento: convertir la imagen RGB a gris, utilizando el método estándar del Comité Nacional de Estándares de Televisión Nacional (NTSC), después esta se mejora utilizando la técnica de ecualización de histograma. En la etapa de procesamiento: se realiza mediante la operación morfológica y el análisis de histograma de perfiles de proyección vertical y horizontal, también se utiliza un umbral dinámico para filtrar los valores del histograma horizontal, vertical; el ROI se extrae de la imagen de la matrícula utilizando diferentes propiedades geométricas, como el área, el cuadro delimitador y la relación de aspecto. En la etapa de segmentación: se utiliza el análisis de componentes conectados (CCA) y tecnología de cuadro delimitador. En la etapa de clasificación: el clasificador de la máquina de vectores de soporte (SVM), donde las características extraídas del histograma de gradiente orientado (HOG) se utilizan como entrada. Resultado, el algoritmo propuesto se aplicó a 630 imágenes de licencia de diferentes categorías de vehículos y logró una precisión del 91% en la extracción de ROI y 94,6% de precisión en la extracción de caracteres. Conclusión, se ha presentado un enfoque para la extracción de caracteres bengalíes, letras y dígitos de las placas de los vehículos de Bangladesh y el reconocimiento de estos caracteres. El método ha sido probado en dígitos y letras de Bangladesh extraídos de varias placas de vehículos. [19], realizó la investigación, Toward an Optimized Neutrosophic k-Means With Genetic Algorithm for Automatic Vehicle License Plate Recognition (ONKM-AVLPR), en la Kafrelsheikh University de Egypt. Diversas deficiencias en la matrícula con respecto a la forma, tamaño, signos y colores de matrícula de un país, perjudicando en la precisión e identificación de placas ya que son ineficientes. Por esta razón, propusieron una nueva metodología para el reconocimiento de placas (LP) y un conjunto neutrosófico (NS) optimizado basado en el algoritmo genético (GA). En la fase de adquisición: se captura la imagen con una cámara marca Nikon de alta resolución las tomas se hacen a una distancia de 2 a 3 metros en la parte posterior y delantera del vehículo con las condiciones de iluminación nublado, soleado y lluvioso. En la fase de pre procesamiento: la imagen rojo, verde, azul (RGB) del automóvil, se reduce a 50% de su escala original para reducir

el tiempo de cálculo, además mientras que la imagen en la gama de grises tiene solo un canal, por lo que convertimos la imagen RGB al formato de escala de grises, se aumenta el contraste de las imágenes para facilitar el proceso de detección de LP. En la etapa de procesamiento: para detectar los bordes se utiliza el operador sobel y operaciones morfológicas que consiste en la erosión y dilatación de la imagen para aislar el fondo de la placa. En la etapa de extracción: se utiliza GA para optimizar las operaciones NS, el uso de NS disminuye la indeterminación en las imágenes LP. En la etapa de segmentación: se usa el algoritmo de agrupamiento k-means que suma los objetos en grupos. En la fase de clasificación: utilizo análisis de etiquetado de componentes conectados (CCLA), para identificar las regiones de píxeles conectadas y agrupar los píxeles apropiados en componentes para extraer cada carácter de manera efectiva. Resultados el método propuesto tiene como resultado 96,67% de precisión de reconocimiento, para una LP inglesa corrupta de baja resolución, asimismo el sistema propuesto alcanza aproximadamente el 94,27% de precisión en la perturbación de la imagen, es decir destello en la imagen, ruido externo y variación de iluminación, además alcanza aproximadamente el 92.5% de precisión de identificación correcta, sin embargo, los métodos tradicionales logran aproximadamente un 79% de precisión de identificación correcta en presencia de tales degradaciones de imagen. Conclusión, la metodología tiene una alta tasa de precisión de reconocimiento en degradaciones de imagen populares. [20], realizó la investigación, License Plate Extraction for Moving Vehicles, en la Yonsei University de Korea. La extracción de matrículas de vehículos en movimiento puede variar en gran medida el tamaño por estar inclinadas. Por esta razón, propuso un algoritmo de extracción de matrículas. En la fase de procesamiento: para eliminar el ruido se utiliza el filtro Wiener; para la detección de bordes alrededor del carácter de la placa se utilizó el filtro diferencia de gauss (DoG), además para reducir los bloques de borde no alineados horizontalmente se utilizaron propiedades geométricas, quedando regiones candidatas de matrículas, convirtiendo las imágenes de entrada al espacio de color (HSV) y se produjo histogramas (HS), además se hizo histogramas de referencia de las muestras

de entrenamiento y se compararon las regiones candidatas de la placa con los histogramas de referencia, utilizando la correlación y la distancia de Bhattacharyya, asimismo si el histograma de una región candidata y el histograma de referencia es mayor que un valor umbral, la región candidata se descarta; si la correlación entre el histograma de una región candidata y el histograma de referencia es menor que un valor umbral, la región candidata también se descarta. En la etapa de clasificación: se utiliza el clasificador en cascada Haar basado en las características de Haar para encontrar regiones candidatas. Resultados, para el conjunto de datos propuesto, el recuerdo fue de 0,72, la precisión fue de 0,88 y el puntaje fue de 0,79, además para el conjunto de datos de Caltech, el retiro fue de 0.86, la precisión fue de 0.96 y el puntaje fue 0,91. Conclusiones, los resultados experimentales con datos reales mostraron un rendimiento prometedor.

La revisión de trabajos previos evidencia avances significativos en el ámbito del reconocimiento automático de matrículas (ALPR), pero en el Perú no existen soluciones de ingeniería en reconocimiento de placas de rodaje de mototaxi, asimismo se puede encontrar investigaciones en repositorios sobre placas vehiculares de automóviles, por la cual algunas soluciones presentan deficiencias en sus métodos, por esta razón, esta investigación podrá contribuir en el campo de la ingeniería y en aplicaciones de imágenes en el reconocimiento de placas de mototaxi en ambientes no controlados. Se implementó un método de reconocimiento de placas de rodaje de mototaxi, empleando algoritmos en cada etapa del procesamiento de imágenes. La investigación en este campo resulta importante, porque a partir de este método se puede generar opciones de solución para que más adelante pueda ser utilizada por quien lo requiera.

En este contexto, el objetivo principal fue desarrollar un método para el reconocimiento de placas vehiculares de mototaxis en ambientes no controlados utilizando procesamiento digital de imágenes. A partir de esta premisa, se planteó la siguiente pregunta de investigación: ¿De qué manera se pueden reconocer las placas vehiculares de mototaxis en ambientes no controlados? Para responder a esta interrogante, se formuló la siguiente hipótesis: si se desarrolla un método de procesamiento digital de imágenes entonces se

podrá reconocer placas vehiculares de mototaxi en ambientes no controlados.

II. MATERIALES Y MÉTODO

Materiales

Tabla 1. Materiales de investigación

N°	Nombre	Descripción
1	Dataset de Placas de mototaxi	Consta de 150 imágenes de placas de rodaje en ambientes no controlados.
2	Software	Anaconda
3	Lenguaje Python	Cuenta con librerías especializadas en aprendizaje automático, además, se destaca por la legibilidad de su código
4	Laptop	Procesador: Intel(R) Core(TM)i5-8250U CPU @ 1.60GHz (8 CPUs) Intel(R) UHD Graphics 620 RAM: 12.00 GB Sistema operativo Windows 10 PRO

Nota: Listado de materiales utilizados para la investigación. Elaboración propia

La investigación realizada fue de tipo cuantitativa, aplicada y tecnológica, ya que se analizaron variables independientes y dependientes mediante indicadores estadísticos para realizar comparaciones de resultados según las pruebas ejecutadas, con el objetivo de dar solución a la problemática planteada [21]. El diseño del estudio se caracterizó por ser cuasi-experimental, debido a que la ejecución de la muestra dependió directamente de la finalidad de la investigación. En cuanto a la población del estudio, esta estuvo conformada por diversos algoritmos empleados en el procesamiento de imágenes digitales. Entre ellos se incluyeron Gaussian Blur, Convolve, Mediam, Mean, Minimum, Maximum, ACO, Canny, Sobel, Roberts Edge Detection, OTSU, ISODATA, histogramas de gradiente orientado (HOG), análisis de componentes conectados (CCA), Técnica

Hough Transform, KNN, SVM, Redes Neuronales Artificiales (ANN), MLP, Random Forest (RF), Redes Bayes, Adaptive Boosting (AdaBoost), reconocimiento óptico de características (OCR), K-means, Perceptrón multicapa (MLP) y el algoritmo Viola-Jones. Estos algoritmos fueron distribuidos en cada etapa del procesamiento de imágenes digitales y referenciados en las **tablas 6 – 7** y el **Anexo 08**, además la muestra seleccionada se determinó a partir de la evaluación de los algoritmos en cada una de las etapas del procesamiento de imágenes para la creación del método de reconocimiento de placas de mototaxis. Los algoritmos con mejor desempeño incluyeron cv2.cvtColor, operaciones morfológicas como TopHat y BlackHat, Gaussian Blur, técnica de umbralización adaptativa, método de Otsu, detección de bordes con Canny, K-Nearest Neighbors (KNN), histogramas de gradientes orientados (HOG) y reconocimiento óptico de caracteres (OCR), como se documenta en las **tablas 6 -7** y el **Anexo 08**.

En términos de recolección de datos, se registraron electrónicamente mediante un script que generaba automáticamente el método propuesto.

Método de investigación

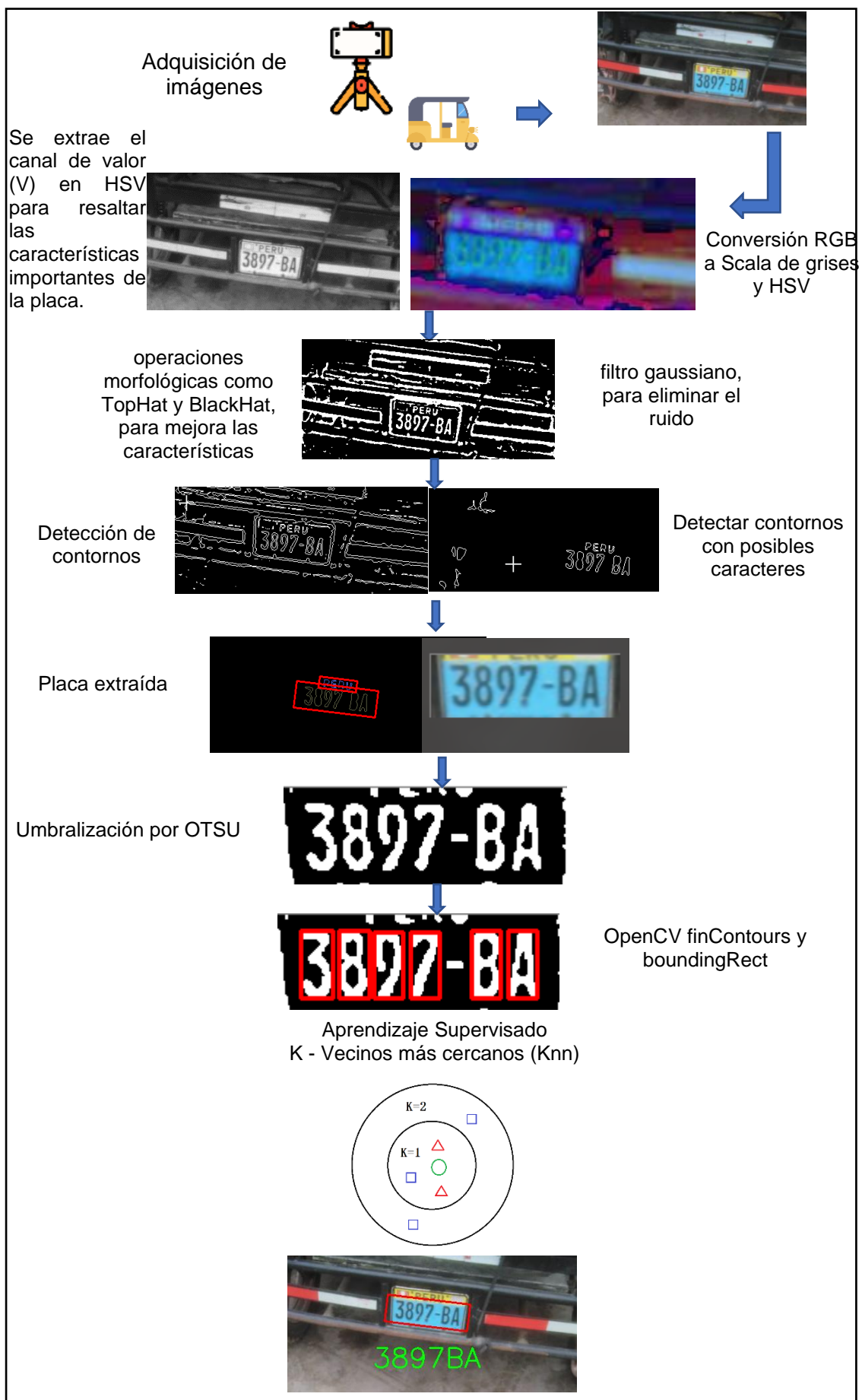


Figura 3. Método propuesto. Fuente: Elaboración propia.

Aporte práctico.

En esta investigación se realizó un esquema general, que consta de 8 tareas, que tuvo como finalidad el desarrollo del tema del método de reconocimiento de placas vehiculares de mototaxi en ambientes no controlados y se sitúa de la siguiente forma.



Figura 4. Esquema general del tema del método de reconocimiento de placas de rodaje de mototaxi. Fuente: Elaboración propia.

A) Adquisición de imágenes digitales

Para esta etapa se realizó un proceso de 3 pasos para la adquisición de imágenes de las placas de rodaje de mototaxi y se representa de la siguiente manera.

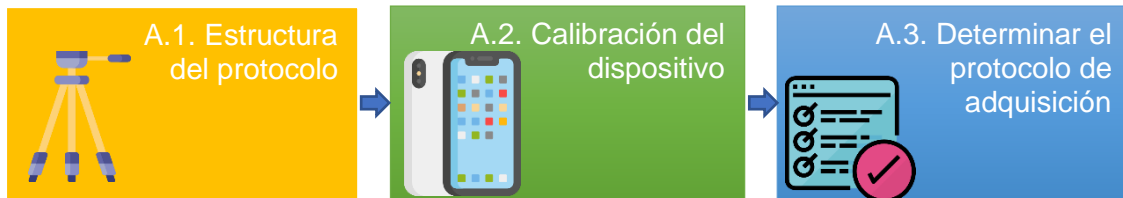


Figura 5. Etapas a seguir para la obtención de imágenes. Fuente: Elaboración propia.

A.1. Estructura del protocolo de adquisición.

Para realizar la adquisición de las imágenes se estableció un protocolo específico, para ello se utilizó un trípode previamente fabricado de tubo de metal de la marca JETION, que es una herramienta regulable y resistente, al respecto fue configurado a 1,20 metros de altura donde en la parte superior fue sujetado el dispositivo celular, como se observa en la figura.

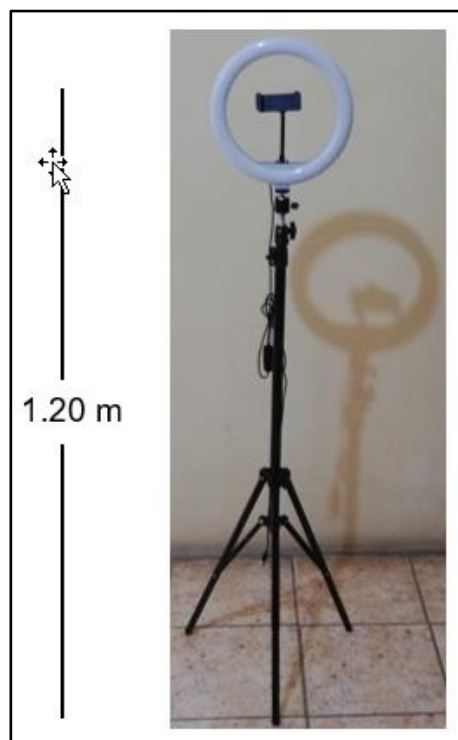


Figura 6. Estructura real del protocolo de adquisición. Fuente: Elaboración propia.

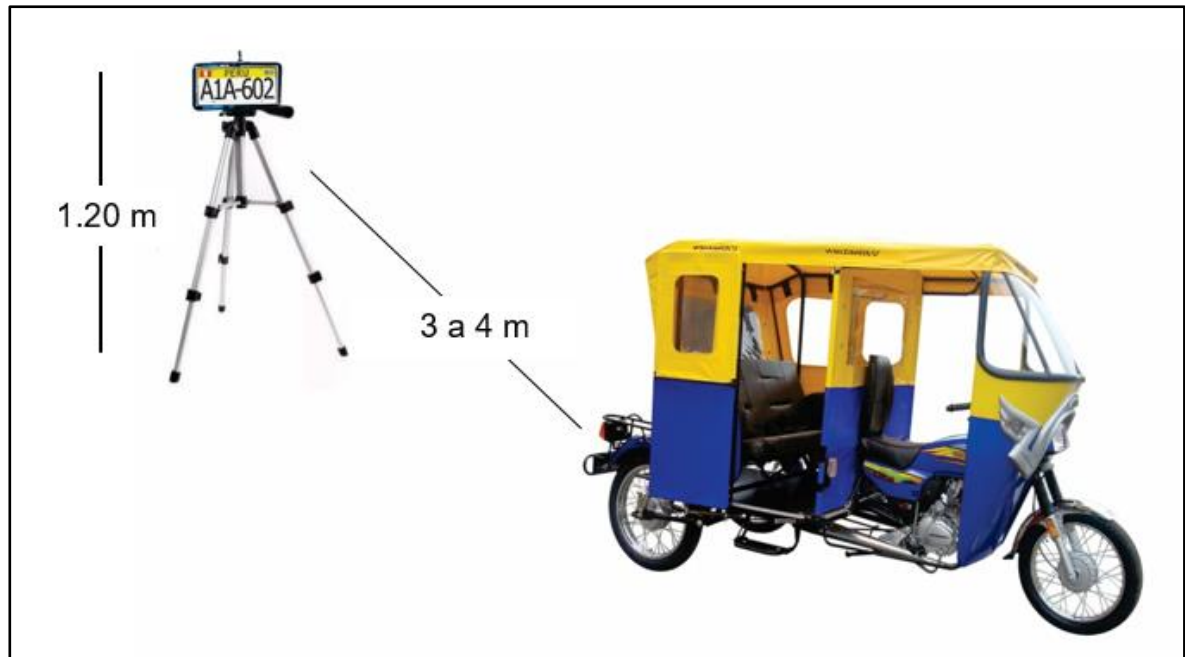


Figura 7. Prototipo específico para el proyecto. Fuente: Elaboración propia.

A.2. Calibración del dispositivo

En la siguiente tabla se puntualizó las especificaciones de pantalla, que se utilizó para la adquisición de imágenes.

Tabla 2. Especificaciones técnicas de pantalla.

Características	
<p>Dimensión diagonal: 5.9".</p> <p>Tipo: IPS.</p> <p>Resolución: 1080 x 2160 px.</p> <p>Espesor de píxeles: 409 ppi (píxeles por pulgada).</p> <p>Profundidad de color: 24 bit.</p> <p>Espacio de la pantalla: Área frontal total del dispositivo ocupado por la pantalla. 76.72 %.</p> <p>Proporción: Relación entre el alto y el ancho del dispositivo. 2:1.</p> <p>Ancho: 2.64".</p> <p>Alto: 5.28".</p> <p>Otras características:</p> <p>Capacitiva.</p> <p>Multitáctil.</p> <p>Pantalla con vidrio curvo (2.5D).</p> <p>GFF full lamination</p>	

Nota: Tomado de Hawei.

También se detalló las especificaciones de la cámara, como se muestra a continuación.

Tabla 3. Especificaciones técnicas de la cámara.

Características	
Sensor: CMOS BSI (backside illumination).	
Apertura de diafragma: f/2.2.	
Tipo de flash: LED.	
Resolución de imagen: 4608 x 3456 px.	
Megapíxeles de imagen: 16 MP.	
Resolución de video: 1920 x 1080 px.	
Megapíxeles de video: 2.07 MP.	
Cuadros por segundo: 30.	
Otras características.	
Foco automático.	
Disparador continuo.	
Zoom digital.	
Geo-etiquetado.	
Panorama.	
HDR.	
Foto táctil.	
Detector de rostros.	
White balance settings.	
ISO settings.	
Exposure compensation.	
Temporizador.	
Scene mode.	
Phase detection.	
Secondary rear camera - 2 MP.	

Nota: Tomado de Huawei.

Luego, para tener imágenes digitales con buenas características, se estableció parámetros de configuración del dispositivo de adquisición.

Tabla 4. Configuración del dispositivo de adquisición de imágenes.

Característica	Detalle	Descripción
Apertura focal	f/2.2	El sensor con menor apertura, tiene más nitidez, debido a que captura escenarios más profundos.
Velocidad de obturación	1/12s	Se utilizó normalmente para congelar el movimiento y reducir el tiempo de exposición. Para capturar objetos en movimiento, la velocidad de disparo recomendada es entre 1/80 y 1/125.
Sensibilidad ISO	100 - 200	Se utilizó para las imágenes diurnas. Por otra parte para las imágenes con poca luz se utilizó un valor por encima de 400.
AF	AF-C	Se recomienda para enfoques con escenas en movimiento.

Nota: Elaboración propia.

Determinar el protocolo de adquisición

La adquisición de las imágenes se obtuvo en un ambiente no controlado, por consiguiente, el dispositivo de adquisición se configuró a una distancia de 2 - 4 metros y con un ángulo descendente no menor de 45°, hacia la placa de rodaje, por otra parte, el tripode fue

ubicado en una intersección de calles de alto flujo vehicular y sobre una veredera, con el fin de adquirir la mayor cantidad de imágenes posibles, donde se obtuvo un dataset de 150 imágenes de placas de rodaje de mototaxi.



Figura 8. Imagen real del protocolo de adquisición de imágenes de placas de mototaxi.
Fuente: Elaboración propia.

B) Construcción del dataset.

Se creó un dataset de las imágenes adquiridas para realizar las pruebas de reconocimiento con el método propuesto, de modo que, fueron 150 fotografías en formato JPEG, de mototaxis estacionadas y tomas de en movimiento, al mismo tiempo, se tuvo en cuenta el horario, así como también de las placas de vista angular, para la obtención de imágenes para este tipo de investigación.

La organización de las carpetas fue creada de la siguiente manera: la carpeta principal denominada



También, dentro de ella se creó la subcarpeta como se observa en la imagen.



En ese mismo contexto, al abrir la subcarpeta mencionada nos mostrará la siguiente interfaz.



Finalmente, en la carpeta PLACAS fueron almacenadas las imágenes adquiridas por el dispositivo de adquisición, para luego ser procesadas por el método propuesto, además estas imágenes están compartidas en el siguiente enlace: https://drive.google.com/drive/folders/18a_LbZnPh9aOwAJQceCNZcZ35SHcyZUA?usp=sharing



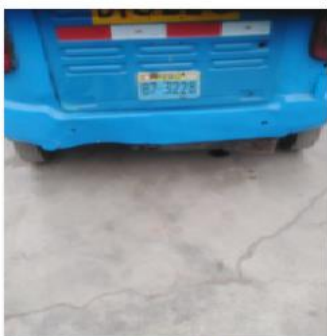
mototaxi_H_DIA (1)



mototaxi_H_DIA (2)



mototaxi_H_DIA (3)



mototaxi_H_DIA (5)



mototaxi_H_DIA (6)



mototaxi_H_DIA (7)

Figura 9. Imágenes adquiridas. Fuente: Elaboración propia.

C) Seleccionar el método a implementar.

En esta investigación se realizó una Revisión Sistemática de la Literatura (SRL) sobre algoritmos, técnicas y métodos para el reconocimiento de placas vehiculares, utilizando las reglas definido por [22] de modo que se realizó 5 interrogantes de investigación que guardan concordancia con los elementos del PICOC, así como también se definieron los términos de búsqueda para la realización de la investigación de artículos en repositorios reconocidos, logrando los siguientes resultados: Scopus con 2,827, IEEEXplore con 821, ScienceDirect con 539 y ACM con 84 resultados, asimismo se definieron criterios de inclusión e exclusión para evaluar los artículos seleccionados, además se hizo el seguimiento de la calidad del contenido, finalmente se extrajeron los datos, para su análisis y evaluación.

Tabla 5. Cadena de búsqueda

Nro.	Cadena de Búsqueda	Base de Datos
1	("vehicle license" or "plate recognition" or " methods recognition ")	SCOPUS
2	("vehicle license" or "plate recognition")	IEEE
3	("algorithms" OR "technique") AND ("method recognition" OR "Plaque vehicle" OR "Recognition plate vehicle")	SCIENCEDIREC T
4	("vehicle enrollment" or "plate recognition")	IOP
5	("vehicle enrollment" or "plate recognition")	ACM

Nota: Palabras claves. **Fuente:** Elaboración propia.

En los últimos 5 años se ha encontrado diversos algoritmos, técnicas y métodos de reconocimiento de placas de rodaje solo de automóviles, sin embargo, se ha tomado en cuenta para el desarrollo del método de reconocimiento de placas de rodaje de mototaxi,

los algoritmos de mejores resultados evaluados por cada investigador en sus artículos ver **Tabla 6-8**, así como también mediante una selección de técnicas, algoritmos y métodos de mejor desempeño ver **Anexo 08**; en cada etapa del procesamiento de imágenes, finalmente la información obtenida se presenta en las tablas siguientes.

En la etapa del preprocesamiento, se optó por la técnica de transformaciones morfológicas como TOPHAT, BLACKHAT, también filtrado gaussiano. Por ser técnicas referenciadas por varios investigadores con mejor desempeño en cuanto a la erosión, dilatación, apertura y cierre en las imágenes.

Tabla 6. Cuadro de de investigaciones sobre preprocesamiento de imágenes.

Investigadores	Algoritmos, técnicas y métodos						
	Filtro gaussiano	Sobel	Transformaciones morfológicas: TOPHAT,	ACO	Canny	Se aplica un filtro	filtro Wiener y filtros DoG
[23]	X		X				
[17]							
[24]				X			
[25]		X					
[12]	X	X					
[14]	X		X				
[26]	X		X				
[27]		X					
[20]			X				X
[23]	X						
[28]			X				

Nota: Algoritmos selectos de artículos científicos. Fuente: Elaboración propia.

Simultáneamente en la etapa de la segmentación, se optó por el método OTSU, por tener ser una técnica referenciada por varios investigadores.

Tabla 7. Cuadro de resultados de investigaciones sobre segmentación de imágenes.

Investigadores	Algoritmos, técnicas y métodos			
	(ACC)	OTSU	histograma de gradiente orientado (HOG)	k- el algoritmo de agrupamiento means
[25]	X			
[27]	X	X		
[29]		X		
[19]		X		
[30]	X			
[31]		X		

Nota: Algoritmos selectos de artículos científicos. **Fuente:** Elaboración propia.

Finalmente, en la etapa de la clasificación existen una diversidad de técnicas, la misma que se optó por KNN, por ser utilizadas por varios autores, como se evidencia en la tabla siguiente.

Tabla 8. Cuadro de resultados de investigaciones sobre clasificación de imágenes.

Investigadores	Algoritmos, técnicas y métodos							
	máquina de vectores de soporte (SVM)	Red neuronal artificial (ANN)	RF	Adaptive Boosting (AdaBoost)	OCR	MPL	KNN	CNN
[18]	X			X				
[17]	X							
[24]	X							
[25]								
[12]			X			X	X	
[14]							X	
[32]								
[27]	X							
[29]							X	
[31]		X					X	
[33]							X	

Nota: Algoritmos selectos de artículos científicos. **Fuente:** Elaboración propia.

Asumiendo lo analizado en las tablas anteriores, se realizó la selección de los algoritmos para la creación del método para el reconocimiento de placas de rodaje de mototaxi, que empieza en la etapa de adquisición de imágenes, el cual se empleó los protocolos específicos mencionados anteriormente, para realizar las tomas fotográfica obteniendo 150 imágenes, para muestra el siguiente código:

```
imgEscenaOriginal = cv2.imread("D:\\software iso\\METODO DE RECONOCIMIENTO DE
PLACAS DE MOTOTAXI\\DPLACAS\\DIA\\mototaxi_h_dia (11).jpeg")
Esta función carga una imagen desde un archivo, se representa como un arreglo (matriz), en
donde cada elemento representa un píxel.
```

Asimismo en esta siguiente etapa se preprocesó las imágenes, para ello se convirtió de RGB a Scala de grises para simplificar el procesamiento y HSV donde se extrajo el canal de intensidad (value), puesto que los colores primarios son afectados por la iluminación, cuando las condiciones de la imagen cambian por ser adquiridas en ambientes no controlados, para esto se dividió los tres canales de la imagen, tono (H), saturación (S), mostrando solo el canal de valor (V), de la imagen, también para la mejora de contraste se empleó la función maximizarContraste(imgEscalaDeGrises) esto es para resaltar los detalles de la imagen. Esta realiza varios pasos claves necesarios para mejorar la calidad de la imagen y facilitar la detección posterior de elementos, como números y letras.

Se utilizó esta lista de librerías en el método de investigación para el reconocimiento de placas de mototaxis. OpenCV (cv2): que proporciona herramientas para el preprocesamiento, procesamiento, detección de bordes, segmentación, y detección de contornos. NumPy: manipulación de matrices y cálculos numéricos, esenciales para manipular datos de imágenes como arrays. scikit-learn: para implementar algoritmos de aprendizaje supervisado, como el K-Nearest Neighbors (KNN) para la clasificar caracteres, basados en el conjunto de datos de entrenamiento. Matplotlib: se usa para la visualización de imágenes y mostrar resultados. time: para medir el tiempo de ejecución de diferentes etapas del proceso, calculando métricas de rendimiento.

Iniciamos detallando el código fuente más importante del método

```
def preprocesamiento(imgOriginal):
    imgEscalaDeGrises = extraerValor(imgOriginal)
    imgMaxContrastGrayscale = maximizarContraste(imgEscalaDeGrises)
    height, width = imgEscalaDeGrises.shape
    imgBlurred = np.zeros((height, width, 1), np.uint8)
    imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale,
    GAUSSIAN_SMOOTH_FILTER_SIZE, 0)
    imgUmbral = cv2.adaptiveThreshold(imgBlurred, 255.0,
    cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,
    ADAPTIVE_THRESH_BLOCK_SIZE, ADAPTIVE_THRESH_WEIGHT)
    return imgEscalaDeGrises, imgUmbral
```

De acuerdo a la función, estos son los parámetros principales: conversión a escala de grises (extraerValor): esto optimiza la imagen suprimiendo información de color y conservando la intensidad luminosa, el aumento del contraste (maximizarContraste): para mejorar las propiedades de la imagen para que sean más fáciles de detectar, el desenfoque Gaussiano (GaussianBlur): para suavizar la imagen para reducir el ruido y detalles innecesarios, para facilitar la segmentación, el Umbral adaptativo (adaptiveThreshold): que convierte la imagen en blanco y negro (binaria) con un método adaptativo, para facilitar la segmentación de las regiones de interés (como la placa). Otra función importante: def extraerValor(imgOriginal), esta función convierte la imagen de su espacio de color RGB a escala de grises usando el canal de valor de la imagen en el espacio de color HSV.

```
def extraerValor(imgOriginal):
    height, width, numChannels = imgOriginal.shape
    imgHSV = cv2.cvtColor(imgOriginal, cv2.COLOR_BGR2HSV)
    imgHue, imgSaturation, imgValue = cv2.split(imgHSV)
    return imgValue
```

Proceso:

La imagen se convierte de RGB a HSV.

Se extrae el canal "Value" del modelo de color HSV, que representa la intensidad de la luz (brillo) en cada píxel, fue necesario convertir una imagen de un espacio de color a otro, puesto que los colores de las placas de mototaxi, se ven afectados por la iluminación ya que fueron tomadas en ambientes no controlados. La fórmula de conversión es la siguiente.

$$H = \left[\begin{array}{l} \frac{G - B}{V - \min(R, G, B)} * 60^\circ, \text{ if } V = R \\ \left(2 + \frac{B - R}{\max - \min} \right) * 60^\circ, \text{ if } V = G \\ \left(4 + \frac{R - G}{\max - \min} \right) * 60^\circ, \text{ if } V = B \\ \left(6 + \frac{R - G}{\max - \min} \right) * 60^\circ, \text{ if } V = B \text{ and } G < B \end{array} \right]$$

El componente de saturación S es:

$$S = \begin{cases} \frac{V - \min(R, G, B)}{\max}, & \text{if } V \neq 0 \\ 0 & \end{cases}$$

El modelo HSV es representado por H, S, V, donde H es el tono, S es la saturación y V es el brillo, Además: H y S: contienen la información de color de la imagen, V: representa la información de brillo. Se dividió los tres canales de la imagen, tono (H), saturación (S), luego se mostró solo el canal de valor (V), de la imagen, dando como resultado una imagen a escala de grises [34].

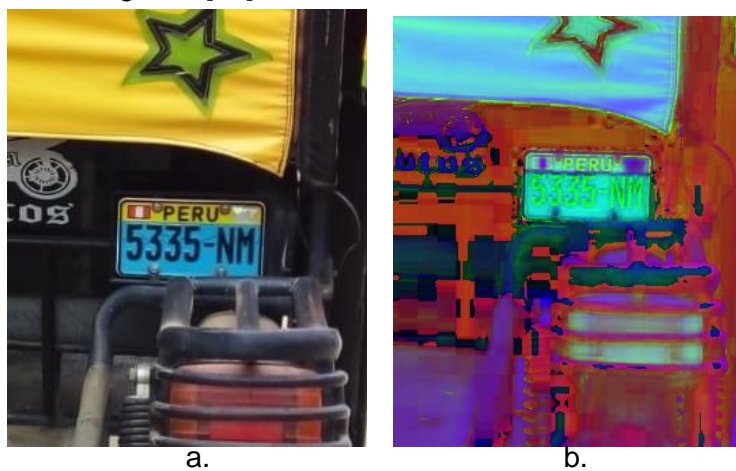


Figura 10. Etapa preprocesamiento. a) Imagen original. b) Imagen HSV.
Fuente: Elaboración propia.

Este paso se usa para enfatizar las propiedades importantes de la imagen usando operaciones morfológicas que mejoran el contraste entre las áreas de interés (como los caracteres de la placa).

```
def maximizarContraste(imgEscalaDeGrises):
    height, width = imgEscalaDeGrises.shape
    imgTopHat = np.zeros((height, width, 1), np.uint8)
    imgBlackHat = np.zeros((height, width, 1), np.uint8)
    structuringElement = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    imgTopHat = cv2.morphologyEx(imgEscalaDeGrises, cv2.MORPH_TOPHAT,
    structuringElement)
    imgBlackHat = cv2.morphologyEx(imgEscalaDeGrises, cv2.MORPH_BLACKHAT,
    structuringElement)
    imgEscalaDeGrisesPlusTopHat = cv2.add(imgEscalaDeGrises, imgTopHat)
    imgEscalaDeGrisesPlusTopHatMinusBlackHat =
    cv2.subtract(imgEscalaDeGrisesPlusTopHat, imgBlackHat)
    return imgEscalaDeGrisesPlusTopHatMinusBlackHat
```

por otra parte se aplicó operaciones morfológicas como TopHat y BlackHat para mejorar las características selectas, se aplicó filtro gaussiano para eliminar el ruido, con un tamaño de filtro definido como (5, 5), este se ejecuta con cv2.GaussianBlur que consiste en ajustar sutilmente los niveles de gris entre píxeles adyacentes, obteniendo una imagen con los bordes más suaves en la que se pierden minúsculos detalles, de manera similar a lo que ocurre en las fotografías desenfocadas, en ese mismo contexto se realizó la umbralización adaptativa, esta técnica se usa para convertir la imagen en escala de grises en una imagen binaria, enfatizando las áreas de interés y los caracteres en la placa, la función cv2.adaptiveThreshold usa un tamaño de bloque y un peso ajustables. La operación TopHat destacó las áreas más brillantes de las imágenes, particularmente aquellas que son más pequeñas que los elementos estructurales, como los caracteres de las placas sobre un fondo oscuro. Por otro lado, la operación BlackHat realzó las áreas más oscuras que son más pequeñas que los elementos de textura, ayudando a corregir variaciones de sombras o irregularidades en el fondo.

Finalmente, la combinación de los resultados de TopHat y BlackHat permitió cancelar las interferencias entre ambas operaciones. Esta integración generó imágenes con un contraste mejorado, donde las características importantes de los caracteres y símbolos de las placas quedaron claramente definidas y listas para etapas posteriores del procesamiento.

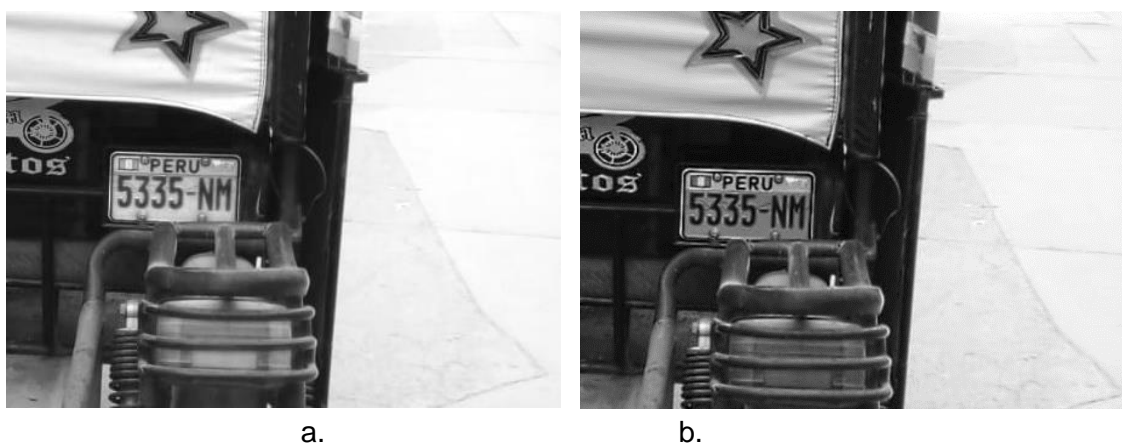


Figura 11. Etapa preprocesamiento. a) Se muestra el valor(V), escala de grises. b) Transformación morfológica TopHack, BlackHat. Fuente: Elaboración propia.

En base a lo mencionado anteriormente, se observó que en algunas imágenes presentaron ruido, por lo tanto, se utilizó la técnica de suavizado gaussiano, para eliminar detalles que puedan interferir en la detección de contornos. La fórmula de la función gaussiana es.

$$f(x) = a \cdot \exp\left(-\frac{(x-b)^2}{2c^2}\right)$$

Donde:

a: es la altura del pico de la curva.

b: es la posición del centro del pico.

c: la desviación estándar, controla el ancho de la campana.

```
#22 imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale,  
    GAUSSIAN_SMOOTH_FILTER_SIZE, 0)
```

En la línea 22, esta función aplica un filtro de suavizado Gaussiano con un tamaño de kernel fijo de (5x5 en el caso del método), suaviza la imagen, lo que ayuda a reducir el ruido y a prevalecer las áreas más importantes, suprimiendo detalles que podrían confundir al método.



a.

b.

Figura 12. Etapa procesamiento. a) Imagen escala de grises. b) Imagen con filtro gaussiano. Fuente: Elaboración propia.

Posteriormente se aplicó el umbral adaptativo, esta técnica modifica la imagen en escala de grises a una imagen binaria, en lugar de utilizar un umbral global en toda la imagen, se utilizó un umbral local adaptativo en diversas áreas.

```
imgUmbral = cv2.adaptiveThreshold(imgBlurred, 255.0, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, ADAPTIVE_THRESH_BLOCK_SIZE, ADAPTIVE_THRESH_WEIGHT)
```

Donde al umbralización adaptativa, divide la imagen en pequeños bloques y calcula el umbral individualmente para cada bloque. La fórmula general es:

$$T(x,y) = \frac{\sum(x',y') \in \text{vecindario}(x',y')}{N} - C$$

$I(x',y')$ Son valores de intensidad en una población alrededor de (x',y')

N es el número de píxeles en la población (bloque)

C es una constante de configuración que se suprime para controlar el umbral.

```
#24 imgUmbral = cv2.adaptiveThreshold(imgBlurred, 255.0, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, ADAPTIVE_THRESH_BLOCK_SIZE, ADAPTIVE_THRESH_WEIGHT)
```

En la línea veinticuatro, el umbral adaptivo convierte la imagen suavizada en una imagen binaria, donde las placas y caracteres se acentúan claramente, la binarización adaptativa ajusta el umbral para cada área de la imagen, cuando hay variaciones de iluminación.



Figura 13. Imagen umbralizada. Fuente: Elaboración propia.

asimismo en la fase de segmentación, se realizó la detección de contornos, el cual se realizó mediante las funciones de OpenCV `findContours` y `boundingRect`, donde la primera función toma como entrada la imagen binaria del proceso anterior y genera una serie de vectores de puntos por cada contorno detectado, de modo que la segunda función sirvió para dibujar un recuadro que rodea el contorno, agregando a lo anterior se aplicó el método OTSU donde se obtiene el umbral utilizando técnicas estadísticas, para posteriormente en la fase de clasificación que se centra en reconocer los caracteres dentro de la placa de rodaje de la mototaxi, se realiza el entrenamiento el algoritmo K-Nearest Neighbors (KNN), utilizando conjuntos de datos de caracteres previamente etiquetados, para ello el entrenamiento se realiza con imágenes aplanadas y las clasificaciones convenientes, posteriormente para la etapa del reconocimiento de caracteres, los caracteres detectados se redimensionan y luego se clasifican mediante el modelo KNN entrenado, este proceso es realizado mediante la función `cv2.ml.KNearest_findNearest`, que compara cada carácter detectado con los datos entrenados con la finalidad de identificar.

En cuanto la función: `cv2.findContours`, detecta contornos en la imagen después de ser binarizada, estos contornos son curvas que conectan puntos seguidos en la imagen que tienen el mismo color o intensidad, se usan para detectar las posibles áreas que cuente con los caracteres de la placa.

Función:

```
_, contours, npaHierarchy = cv2.findContours(imgUmbralCopy, cv2.RETR_LIST,  
cv2.CHAIN_APPROX_SIMPLE)
```

`cv2.findContours()`: Detecta bordes de objetos blancos sobre un fondo negro (imagen binarizada). El algoritmo utiliza el método de aproximación de Douglas-Peucker para reducir los contornos detectados, la función no devuelve una fórmula clara, pero utiliza internamente algoritmos geométricos para acercar los contornos como secuencias de puntos, después de que la imagen ha sido binarizada, `findContours()` detecta las áreas cerradas, que figuran las posibles formas de los caracteres de las placas. Las áreas se

evaluarán próximamente para revisar si contienen caracteres admitidos.

```
#234 imgUmbralCopy = imgUmbral.copy()
#237 _, contours, npaHierarchy = cv2.findContours(imgUmbralCopy, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)
```

En la línea doscientos treinta y cuatro, la función `imgUmbral.copy()` sobrescribe todos los píxeles en el destino, asimismo en la línea doscientos treinta y siete la función `cv2.findContours` devuelve tres valores, según la versión de `opencv` para encontrar todos los contornos para ubicar la placa.

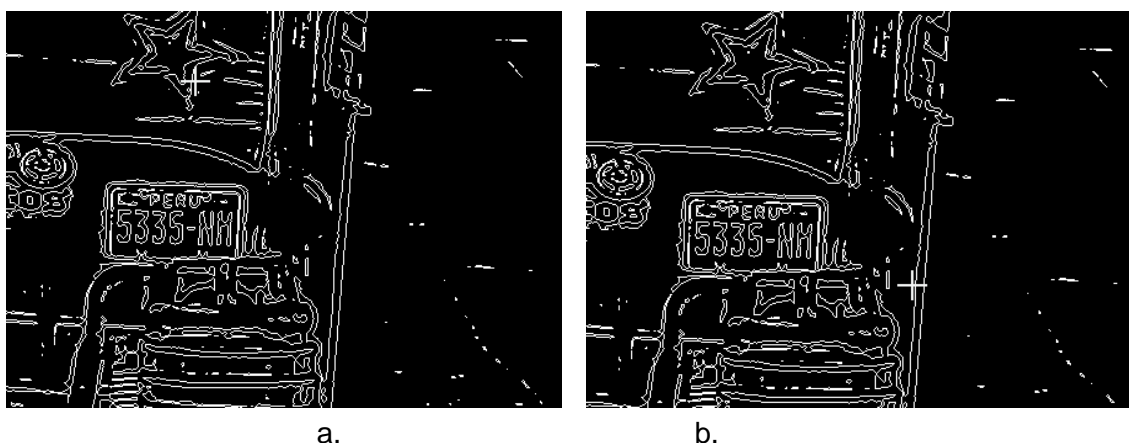


Figura 14. Etapa procesamiento. a) Imagen binarizada. b) Resultado de la función `findContours`. Fuente: Elaboración propia.

Después del proceso anterior, se procedió a eliminar contornos innecesarios para quedarse con contornos similares a los caracteres.

```
#38 imgEscalaDeGrisScene, imgUmbralScene =
Preprocesamiento.preprocesamiento(imgEscenaOriginal)
#47 listaDePosiblesCaracteresEnEscena =
encontrarPosiblePersonajesEnLaEscena(imgUmbralScene)
#53 imgContornos = np.zeros((height, width, 3), np.uint8)
#55 contours = []
#57 for posibleCaracter in listaDePosiblesCaracteresEnEscena:
contours.append(posibleCaracter.contour)
```

En la línea treinta y ocho a las variables `imgEscalaDeGrisScene`, `imgUmbralScene` se le asigna la función del preprocesamiento, para obtener imágenes en escala de grises y umbral, además en la línea cuarenta y siete se buscó todos los caracteres posibles en la imagen, asimismo en la línea cincuenta y tres a `imgContornos` se le asigna a blanco y negro, además, en la línea cincuenta y cinco primero encuentra todos los contornos con el vector `contours= []`, finalmente incluye los contornos que podrían ser caracteres.



Figura 15. Imagen procesada. Fuente: Elaboración propia.

Luego se utilizó el método OTSU para la segmentación de caracteres, que realiza la separación de la imagen original en regiones de interés (ROI), de igual manera como la placa presenta áreas grises, así como también ruido se empleó esta técnica de binarización para calcular automáticamente un valor de umbral a partir del histograma de imagen para una imagen bimodal. Su fórmula matemática es:

$$P_r(r_q) = \frac{n_q}{N}$$

$$q = 0, 1, 2 \dots L - 1$$

Donde:

Pr: normaliza dicho histograma como una función de probabilidad discreta.

nq: número de pixeles que tienen un grado de intensidad.

N: número general de pixeles que tiene una imagen.

rq y L: número general de posibles de niveles de intensidad en una imagen.

Después de normalizar el histograma, se eligió un umbral k tal que C_0 incorpore con niveles $\{0, 1, 2, \dots, k-1\}$ a un conjunto de pixeles y C_{-1} con niveles $\{k, k+1, k+2, \dots, L-1\}$ otro grupo de pixeles. Del mismo modo, la técnica de Otsu selecciona el valor del umbral maximizando la varianza σ_w^2 , el cual está definida por:

$$\sigma_b^2 = w_0 * (u_0 - u_t)^2 + w_1 * (u_1 - u_T)^2$$

$$w_0 = \sum_{q=0}^{k-1} P_q(r_q)$$

$$w_1 = \sum_{q=k}^{L-1} P_q(r_q)$$

Las medidas para cada una de las clases son definidas como:

$$u_0 = \sum_{q=0}^{k-1} q * \frac{P_q(r_q)}{w_0}$$
$$u_1 = \sum_{q=k}^{L-1} q * \frac{P_q(r_q)}{w_1}$$

Seguido de hallar el umbral t , que maximice la varianza, de tal manera Otsu definió que este es el umbral óptimo:

$$t^* = \text{Max}\{\sigma_b^2(t)\}$$

Donde: t

$$1 \leq t \leq L$$

Luego de expresar la fórmula matemática del método Otsu, se procedió a iniciar con la codificación de esta etapa en el lenguaje python.

```
#100 thresholdValue, posiblePlaca.imgUmbral = cv2.threshold(posiblePlaca.imgUmbral, 0.0, 255.0, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
```

En la línea cien, a las variables `thresholdValue`, `posiblePlaca.imgUmbral` se le asigna el primer argumento, `posiblePlaca.imgUmbral` que fue la imagen en escala de grises, seguido del valor del umbral que se utiliza para clasificar los valores de los píxeles, finalmente en el tercer parámetro el máximo valor que representa el total que se dará, si el valor del píxel es mayor, a veces menor que el valor de umbral, del mismo modo se realizó con los tipos de umbral `THRESH_BINARY | cv2.THRESH_OTSU`, el cual permitió para encontrar el centro de masa de la imagen y dar con el umbral para eliminar las áreas grises del fondo de la placa.



Figura 16. Imagen umbralizada. Fuente: Elaboración propia.

Además, la delimitación con `BoundingRect`, traza un rectángulo delimitador alrededor de cada contorno, este rectángulo consigue las coordenadas (x, y) de la esquina superior

izquierda del carácter, asimismo de su ancho y altura.

En la visualización de la segmentación de caracteres, se dibuja un rectángulo alrededor de cada carácter detectado, permitiendo resaltar cada carácter y comprobar la exactitud del reconocimiento.

```
#411 cv2.rectangle(imgUmbralColor, pt1, pt2, Main.SCALAR_RED, 2)
#414 imgROI = imgUmbral[caracterActual.intBoundingRectY :
caracterActual.intBoundingRectY + caracterActual.intBoundingRectHeight,
caracterActual.intBoundingRectX : caracterActual.intBoundingRectX +
caracterActual.intBoundingRectWidth]
#417 imgROIResized = cv2.resize(imgROI, (RESIZED_CHAR_IMAGE_WIDTH,
RESIZED_CHAR_IMAGE_HEIGHT))
```

En la línea cuatrocientos once se dibujó un cuadro rojo alrededor de cada carácter de la imagen de la placa umbralizada para resaltar la región de interés (ROI), del mismo modo en la línea de código cuatrocientos catorce con la función Boundingrect se realizó el recorte de cada carácter en cuanto a sus valores de coordenada x, y, ancho y altura, además en la línea cuatrocientos diecisiete se escala la imagen en cuanto a los términos de ancho y alto.



Figura 17. Segmentación de caracteres de la placa. Fuente: Elaboración propia.

Para el cálculo de distancia entre caracteres, en el proceso de clasificación de caracteres, se mide la distancia entre los centros de los caracteres de la placa con la finalidad de agrupar aquellos caracteres que pertenecen a la misma placa.

Función:

```
def distanciaEntreCaracteres(firstChar, secondChar):
    intX = abs(firstChar.intCenterX - secondChar.intCenterX)
    intY = abs(firstChar.intCenterY - secondChar.intCenterY)
    return math.sqrt((intX ** 2) + (intY ** 2))
```

Distancia Euclidiana: Se usa para calcular la distancia entre dos puntos en el plano. La fórmula es:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Donde $(x_1 - y_1)$ y $(x_2 - y_2)$,son las coordenadas de los centros de dos caracteres.

La fórmula ayuda a agrupar los caracteres de la placa que están contiguo unos de otros, lo que es básico para establecer cuáles caracteres están en la misma placa.

Finalmente en el proceso de la clasificación, en esta investigación, cada posible carácter de la placa se redimensiona a un tamaño estándar, generalmente de 20x30 píxeles, y se aplanan a un vector de una sola dimensión. Esto permite al método comparar caracteres de la placa de diversos tamaños y ubicaciones en la imagen.

```
imgROIResized = cv2.resize(imgROI, (RESIZED_CHAR_IMAGE_WIDTH, RESIZED_CHAR_IMAGE_HEIGHT)) # N° d
npaROIResized = imgROIResized.reshape((1, RESIZED_CHAR_IMAGE_WIDTH * RESIZED_CHAR_IMAGE_HEIGHT))
npaROIResized = np.float32(npaROIResized)# convertir de 1d numpy array of ints a 1d numpy array
```

Figura 18. Parámetros para redimensionar y aplanar las imágenes. Fuente: Elaboración propia.

Posteriormente para el reconocimiento de caracteres se utilizó el algoritmo K-Nearest Neighbors (KNN), que clasifica los caracteres en función de los vecinos más cercanos en el área de características [35].

Función:

```
retval, npaResults, neigh_resp, dists = kNearest.findNearest(npaROIResized, k=1)
```

El algoritmo de clasificación KNN, asigna una clase a un objeto basándose en la mayoría de los vecinos más cercanos. Para $k=1$, el algoritmo busca el vecino más cercano y asigna su clase al nuevo punto. Su fórmula es:

$$\hat{y} = \text{Clase}(x \text{ vecino_mas_cercano})$$

Donde \hat{y} es la clase predicha y x es el conjunto de características de la imagen de entrada.

El funcionamiento dentro del método, el sistema redimensiona los caracteres detectados de la placa, asimismo los aplanan en un vector para luego comparar el vector con los datos de entrenamiento y así determinar la clase del carácter letras o números, asimismo para el muestreo de la placa detectada y los caracteres clasificados, en esta parte ya clasificados los caracteres, el método dibuja un recuadro rojo alrededor de la placa del

mototaxi y digita los caracteres detectados en verde.

De igual forma para dibujar el recuadro rojo alrededor de la placa, se utilizó el método `cv2.boxPoints()`, con la finalidad de definir los ejes de los vértices del recuadro que rodea la placa. Luego, se usa `cv2.line()` para conectar estos puntos y formar el contorno de la placa en color rojo.

```
138 # Función para dibujar el rectángulo rojo en la placa
139 def dibujarRectanguloRojoEnPlaca(imgEscenaOriginal, licPlaca):
140     p2fRectPoints = cv2.boxPoints(licPlaca.ubicacionDeLaPlacaEnLaEscena)
141
142     cv2.line(imgEscenaOriginal, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]), SCALAR_RED, 2)
143     cv2.line(imgEscenaOriginal, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]), SCALAR_RED, 2)
144     cv2.line(imgEscenaOriginal, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]), SCALAR_RED, 2)
145     cv2.line(imgEscenaOriginal, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]), SCALAR_RED, 2)
```

Figura 19. Función para dibujar el recuadro rojo en la placa.

#139 En la línea ciento treinta y nueve se estableció la función `dibujarRectanguloRojoEnPlaca()`, del mismo modo la función `cv2.boxPoints` devuelve un valor, de la placa de una imagen, que luego dibujó una línea roja en el recuadro de la placa



Figura 20. Final del proceso, reconocimiento de caracteres en la placa de mototaxi.

Fuente: Elaboración propia.

III. RESULTADOS Y DISCUSIÓN

3.1 Resultados

En esta Investigación para conocer los resultados del método propuesto se contó con un dataset con 150 imágenes de mototaxis en formato JPEG, con resolución de 300ppp, que fueron tomadas en horario diurno de 8 am a 2pm, donde la placa del vehículo menor cuenta con 6 caracteres, Luego se realizó las pruebas, entre las que consistió en ejecutar el método a imagen por imagen, además su resultado se registró en el instrumento de recolección de datos como se muestra en la **tabla 9 y 10**.

Donde genero la matriz de confusión de 711 caracteres como verdaderos positivos (VP), 188 caracteres como verdaderos negativos (VN), 5 caracteres como falsos positivos (FP) y finalmente 41 caracteres como falsos negativos (FN), a partir de estas variables se calcularon las métricas de operacionalización.

Tabla 9. Matriz de confusión.

		Resultados Clasificación	
		Positivos	Negativos
Resultados	Positivos	VP	FP
Reales	Negativos	FN	VN

Nota: Elaboración propia.

Con base a la tabla anterior, después de la ejecución del método con una imagen se ingresa los datos a la matriz de confusión.

Tabla 10. Cuadro resumen general de las variables de la matriz, en cuanto al reconocimiento de caracteres del dataset de imágenes.

Ítem	Valor
Verdadero positivo (VP)	711
Falso Positivo (FP)	5
Verdadero Negativo (VN)	188
Falso Negativo (FN)	41

Nota: Elaboración propia.

Posteriormente se presentan los resultados del método de reconocimiento de placas de rodaje de mototaxi, para ello se usó la herramienta administrador de tareas de Windows, asimismo se implementó un script para realizar este tipo de mediciones ver Figura 23 y 24, el cual en las pruebas permitió obtener el porcentaje de consumo de CPU, el porcentaje de consumo de memoria RAM y el promedio de tiempo de respuesta. Por otra parte, también se realizó la medición de los siguientes indicadores Exactitud, Precisión, Recall, F1 Score. Estos indicadores se tomaron en cuenta por ser los más usuales en las mediciones de la dimensión de las variables por los autores en sus artículos científicos sobre reconocimiento de placas vehiculares.

En la evaluación del método el indicador Grado de consumo de CPU, en este punto es importante, ya que se formula un método a una posible solución para el reconocimiento de caracteres en la placa de rodaje de mototaxi, así como también será un aporte a la disciplina de la carrera; en la cual es necesario un buen rendimiento del sistema, para ello se realizó una medición del grado de consumo de CPU, que consistió en ejecutar cada imagen tomada como experimento con el método propuesto, además se configuró un script para realizar esta medición ver Figura 23 y 24, el total de las imágenes fueron 150, del mismo modo con base a los resultados alcanzados de cada imagen, se obtuvo la media o promedio de consumo de CPU, donde el porcentaje promedio fue de 10%.

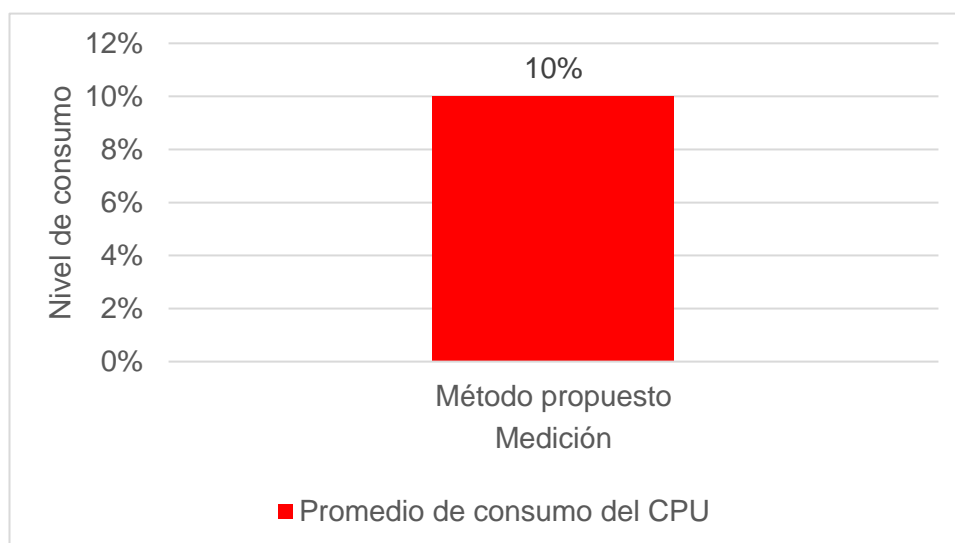


Figura 21. Promedio de consumo del CPU del método propuesto. Fuente: Elaboración propia.

El segundo indicador fue el grado de consumo de la memoria RAM, es importante ya que al ejecutar el método de reconocimiento de placas vehiculares de mototaxi, la computadora hace uso de la RAM, es decir almacena la información en forma temporal de las ejecuciones realizadas, asimismo este tipo de proyectos con Machine Learning consumen mucha memoria y se requiere el óptimo rendimiento por parte del sistema, en tal sentido en esta investigación es importante saber el grado de consumo de este indicador, para ello se realizó una medición, que consistió en la ejecución de cada imagen tomada como experimento con el método propuesto, además se configuró un script para realizar esta medición ver Figura 23 y 24, con base a los resultados de las 150 imágenes se obtuvo la media o promedio de consumo de memoria RAM, como se evidencio en la tabla anterior, en la siguiente figura se muestra el resultado obtenido del grado de consumo de la memoria RAM, donde el porcentaje promedio fue de 61 %.

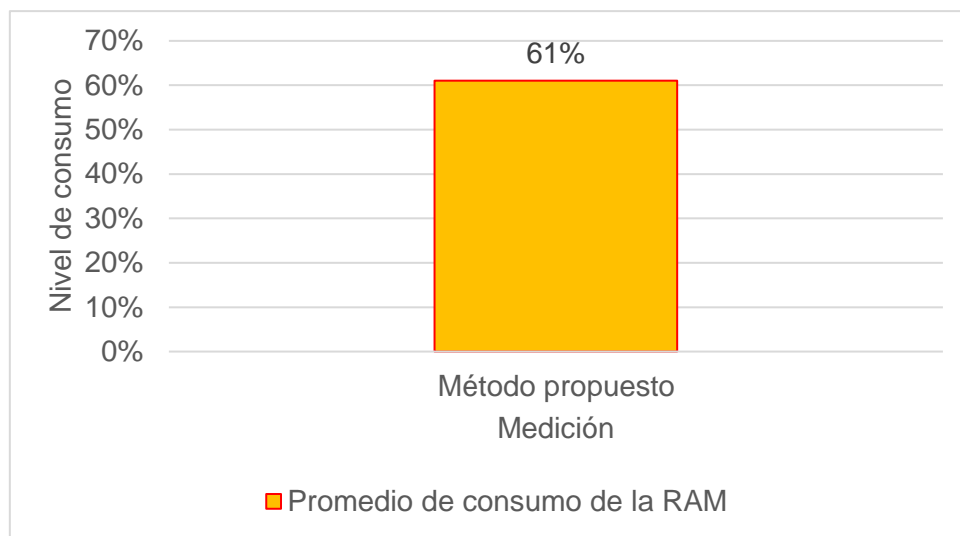


Figura 22. Promedio de consumo de la memoria RAM, del método propuesto.

Fuente: Elaboración propia.

El tercer indicador que se tomó en cuenta es el tiempo promedio de respuesta, la cual se programó un código para obtener este indicador, como se evidencia en la figura siguiente.

```

15 def main():
16     # Medir el consumo inicial de recursos
17     cpu_inicial = psutil.cpu_percent(interval=1)
18     memoria_inicial = psutil.virtual_memory().percent
19     tiempo_inicial = time.time()
20
# Medir consumo final de recursos
cpu_final = psutil.cpu_percent(interval=1)
memoria_final = psutil.virtual_memory().percent
tiempo_final = time.time()

consumo_cpu = round(cpu_final)
consumo_memoria = round(memoria_final)
tiempo_respuesta = round(tiempo_final - tiempo_inicial, 2)

# Mostrar métricas de rendimiento
print(f"Consumo de CPU: {consumo_cpu}%")
print(f"Consumo de Memoria: {consumo_memoria}%")
print(f"Tiempo promedio de respuesta: {tiempo_respuesta} segundos")

```

Figura 23. Medición al inicio y al final del consumo de recursos.

El cuarto indicador es la Precisión, donde los verdaderos positivos (VP), son los caracteres de la placa detectados correctamente, y los falsos positivos (FP), son las regiones de las imágenes de la placa que no contienen ningún carácter, pero que han sido detectadas como regiones con carácter por el método, los cuales se encuentran en la matriz de confusión, su fórmula se representa de la siguiente manera.

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

Para obtener las métricas, se utilizó la librería sklearn.metrics, que es una librería utilizada en Python en problemas de clasificación y aprendizaje automático.

```

77 # ---- Cálculo de métricas basado en comparación carácter por carácter ----
78 caracteres_detectados = licPlaca.strChars.strip() # Eliminar espacios en blanco
79 longitud_placa_real = len(placa_real)
80 longitud_placa_detectada = len(caracteres_detectados)
81
82 # Ajustar comparación para placas detectadas más largas
83 if longitud_placa_detectada > longitud_placa_real:
84     caracteres_detectados = caracteres_detectados[:longitud_placa_real] # Limitar al tamaño de la placa real
85
86 # Inicializar variables para el cálculo de las métricas
87 VP = 0 # Verdaderos Positivos (caracteres correctos detectados)
88 FP = 0 # Falsos Positivos (caracteres incorrectos detectados)
89 FN = 0 # Falsos Negativos (caracteres no detectados)
90
91 # Comparar carácter por carácter entre la placa detectada y la real
92 for i in range(min(longitud_placa_real, longitud_placa_detectada)):
93     if caracteres_detectados[i] == placa_real[i]:
94         VP += 1
95     else:
96         FP += 1
97
98 # Si la placa detectada tiene más caracteres que la real, esos son falsos positivos
99 if longitud_placa_detectada > longitud_placa_real:
100     FP += (longitud_placa_detectada - longitud_placa_real)
101
102 # Si la placa real tiene más caracteres que los detectados, esos son falsos negativos
103 if longitud_placa_real > longitud_placa_detectada:
104     FN += (longitud_placa_real - longitud_placa_detectada)
105
106 # Calcular precisión, recall y exactitud
107 precision = VP / (VP + FP) if (VP + FP) > 0 else 0
108 recall = VP / (VP + FN) if (VP + FN) > 0 else 0
109 exactitud = VP / longitud_placa_real if longitud_placa_real > 0 else 0
110
111 # Mostrar métricas
112 print(f"Precisión: {round(precision * 100, 2)}%")
113 print(f"Recall: {round(recall * 100, 2)}%")
114 print(f"Exactitud: {round(exactitud * 100, 2)}%")

```

Figura 24. Cálculo de las métricas basados en la comparación de carácter por carácter de la placa. Fuente Elaboración propia.

Asimismo, después de realizar las pruebas con el dataset de 150 imágenes de mototaxi con un promedio de 6 caracteres por placa de rodaje de mototaxi, el método obtuvo una precisión de 99.30% de reconocimiento de caracteres en la placa de rodaje, como se muestra en la figura.

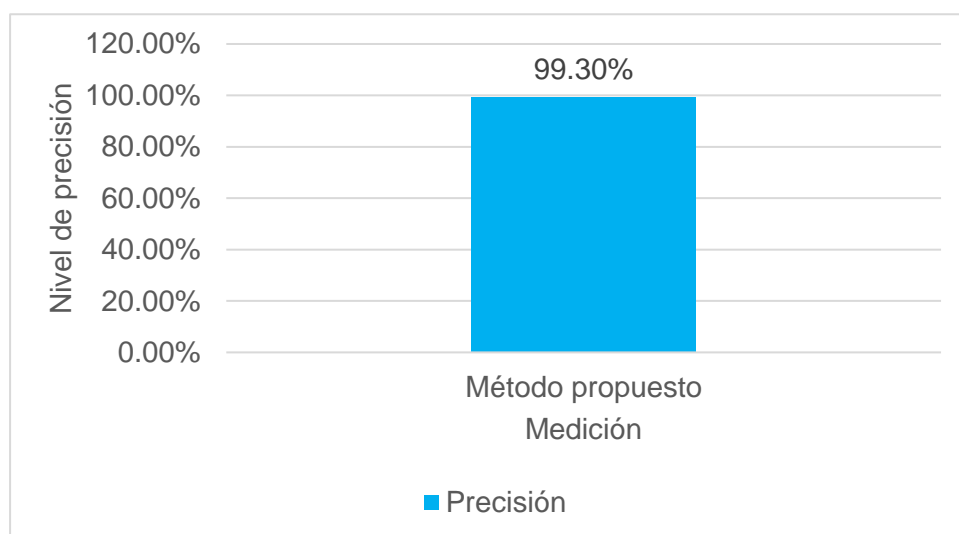


Figura 25. Resultado de la precisión en porcentaje del método propuesto. Fuente: Elaboración propia.

El quinto indicador que se tomó en cuenta es la exactitud, donde los verdaderos positivos (VP), son los caracteres de la placa detectados correctamente, los verdaderos negativos (VN), son los caracteres de la placa detectados incorrectamente, los falsos positivos (FP), son las regiones de las imágenes que no contienen ningún carácter, pero que han sido detectadas como regiones con carácter por el método, falsos negativos (FN), son las regiones en las imágenes que contienen caracteres reales, pero el método no las ha detectado, los cuales se encuentran en la matriz de confusión, su fórmula se representa de la siguiente manera.

$$E = \frac{VP+VN}{VP+VN+FP+FN}$$

Asimismo, después de ejecutar las pruebas con el dataset de 150 imágenes de mototaxi con un promedio de 6 caracteres por placa de rodaje de mototaxi, el método obtuvo una exactitud de 95.13% de reconocimiento de caracteres en la placa, como se muestra en la figura.

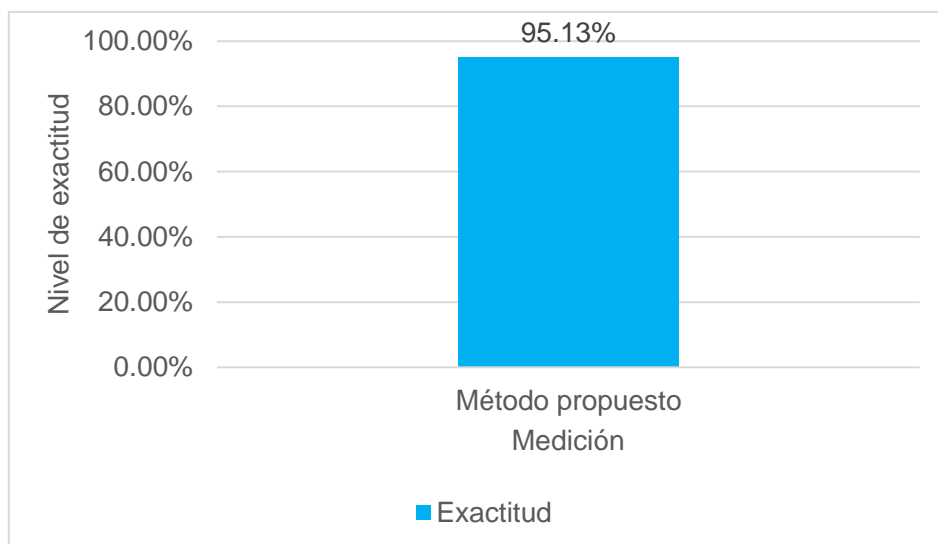


Figura 26. Resultado de la exactitud, obtenido con el método propuesto, utilizando la matriz de confusión. Fuente: Elaboración propia.

El sexto indicador que se tomó en cuenta es el Recall, donde los verdaderos positivos (VP), son los caracteres de la placa detectados correctamente y los falsos negativos (FN), son las regiones en las imágenes que contienen caracteres reales, pero el método no las

ha detectado, los cuales se encuentran en la matriz de confusión, su fórmula se representa de la siguiente manera.

$$R = \frac{VP}{VP+FN}$$

Asimismo, después de ejecutar las pruebas con el dataset de 150 imágenes de mototaxi con un promedio de 6 caracteres por placa de rodaje de mototaxi, el método obtuvo una sensibilidad de 94.55% de reconocimiento de caracteres en la placa, como se muestra en la figura.

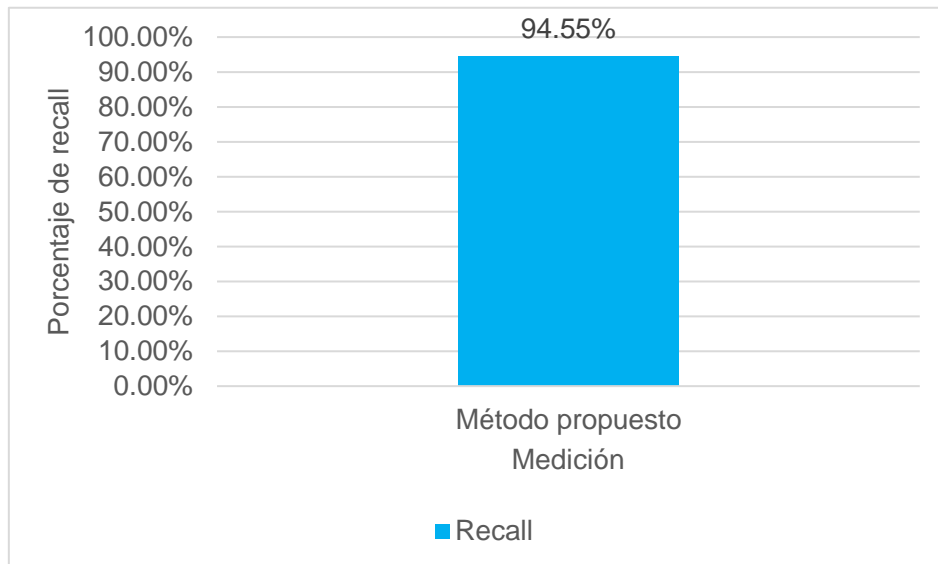


Figura 27. Resultado de la métrica recall, obtenido con el método propuesto, utilizando la matriz de confusión. Fuente: Elaboración propia.

También en este trabajo se consideró como indicador, F Score o F₁, el cual nos da como resultado la medida de precisión que tiene nuestro método de reconocimiento de caracteres en la placa de rodaje de mototaxi, asimismo, organiza los valores de precisión y de recall, de tal forma se promedió las 150 imágenes de placas dando como resultado la precisión del método con 96.87%, como se detalla en la siguiente figura.

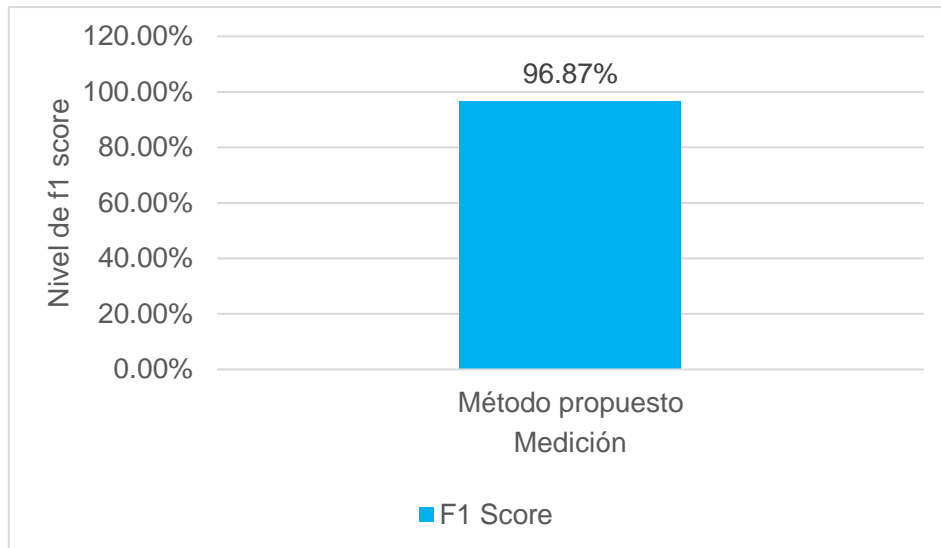


Figura 28. Resultado de F1 Score, en porcentaje del método propuesto.
Fuente: Elaboración propia.

Finalmente se muestra dos gráficos resumen, de las métricas explicadas anteriormente, tanto en la variable independiente como el de la variable dependiente.

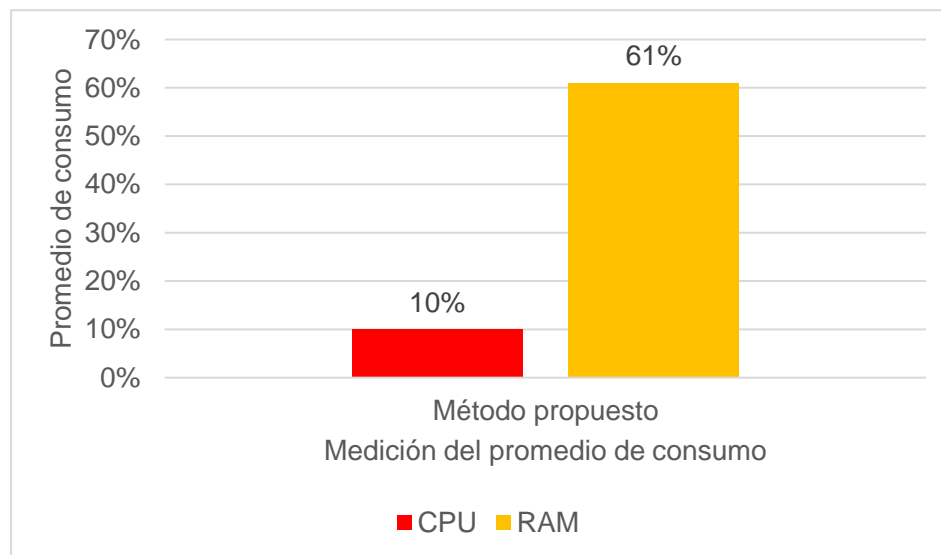


Figura 29. Resultados de la variable independiente. Promedio de consumo del CPU y de la memoria RAM en porcentaje. Fuente: Elaboración propia.

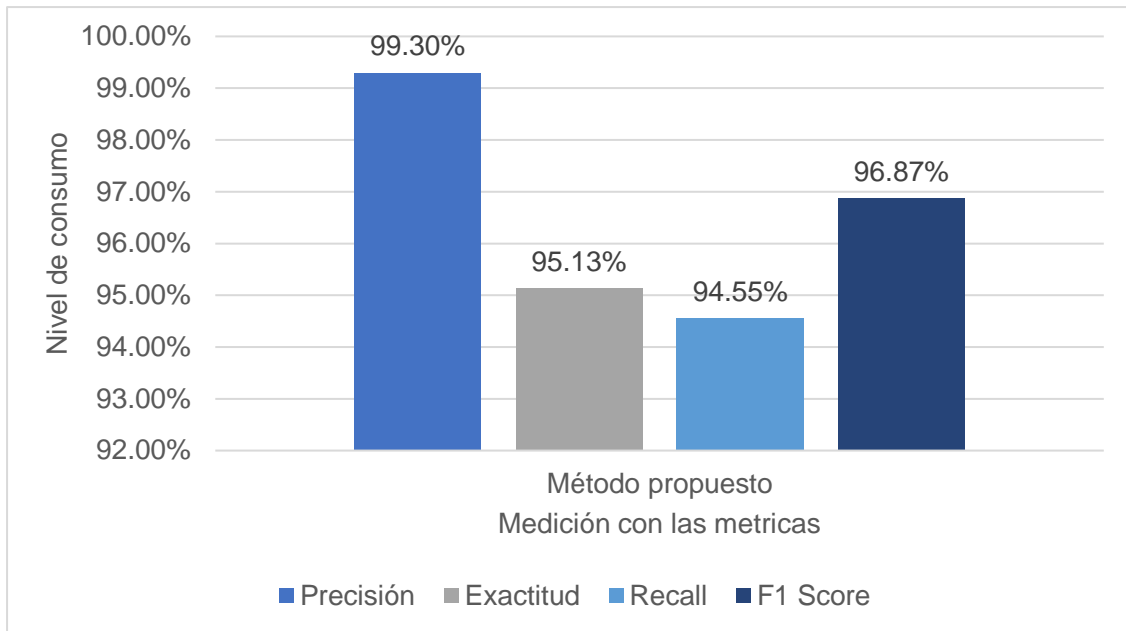


Figura 30. Resultados de las métricas de la variable dependiente, en porcentaje Fuente: Elaboración propia.

3.2 Discusión

En esta parte del documento nos permitió realizar las comparaciones con investigaciones que usaron algoritmos de procesamiento de imágenes, en efecto en esta investigación se obtuvo el 99.30%, de precisión del método de reconocimiento de placas de rodaje de mototaxi, por el contrario [26] en su investigación obtuvo el 95% de reconocimiento, por tal razón el método propuesto trabajó con 150 imágenes y para la segmentación de caracteres se usó algoritmo OTSU, a diferencia de los autores mencionados que usaron un dataset de 50 imágenes y en la segmentación usaron algoritmo SURF, el cual indicaron que tuvieron problemas para tratar la imagen con ruido y en los contornos.

En la investigación [36] se obtuvo una precisión de 85%, sin embargo, en la investigación propuesta la precisión fue 99.30%, ya que cuenta con un dataset de imágenes de dimensiones 886 x 1073 y como obturador electrónico configurado a 1/12 a diferencia de la otra investigación que tuvo como tamaño de imagen de 640 x 480 y como obturador electrónico configurado a 1/100, por lo que hace que su muestra de entrada sea de baja calidad, lo que demuestra que la investigación propuesta cuenta con mejores

muestras.

[16] en su investigación “Parking entrance control using license plate detection and recognition”, obtuvieron resultados similares a la investigación propuesta en cuanto a sus métricas, puesto que emplearon casi las mismas técnicas de procesamiento de imágenes, por el contrario en la etapa de procesamiento, ellos solo trabajaron dos colores de placa, que son para fondos blanco y fondo amarillo, es por ello que en esta investigación no se consideró dichos procesos en la fase, por lo que demuestra que con este trabajo no habría problemas para detectar la placa de mototaxi y de otros tipos vehículos.

[37], en su investigación presentaron un algoritmo de reconocimiento automático de matrículas, en el que destacaron el análisis discriminante lineal (LDA) para la extracción de características, y sin segmentación de la imagen, alcanzando una precisión similar al del método propuesto, sin embargo, en este trabajo, si se consideró la segmentación de caracteres, por lo tanto, se obtuvo un mejor reconocimiento de placas en ambientes no controlados, ya que ellos recomendaron mejorar este aspecto.

IV. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

En esta investigación con base a los resultados se llegó a las siguientes conclusiones. La configuración del dispositivo celular en los parámetros de, apertura local, la velocidad de obturación y la sensibilidad, obtienen favorables resultados de imágenes, es decir la nitidez, fotografías en movimiento, tomas con poca luz. En efecto otras configuraciones no generan estos resultados.

Aunque los escenarios de iluminación en algunas ocasiones no fueron las más favorables, durante la construcción del dataset de las imágenes de mototaxi, se logró un porcentaje de sensibilidad de 94.55%, la cual es tolerable, considerando el estado del dataset.

Se hizo una la revisión de la literatura científica de los últimos 5 años, en que se

encontró una diversidad de algoritmos en cada etapa del procesamiento de imágenes, de modo que para el desarrollo del nuevo método se trabajó con el top de algoritmos con la información recopilada.

Para el desarrollo del método se tomó en cuenta una selección por medio de un ranking de mejores resultados de algoritmos, técnicas, métodos de procesamiento de imágenes eligiendo los filtros con mejores resultados para esta tesis.

El lenguaje de programación Python 3.6 resulta ventajoso para la identificación de placas de rodaje de mototaxi, por la diversidad de bibliotecas disponibles que nos facilita la codificación.

Al ejecutar el método de reconocimiento de placas vehiculares de mototaxi, con las 150 imágenes de prueba, se concluyó que el método propuesto es de alta precisión, que fue capaz de reconocer con éxito 711 caracteres de 899 caracteres posibles. Lo cual es una mejora significativa en el caso de las placas de varios estilos

Los resultados demostraron que el método propuesto con 150 imágenes de prueba, alcanzó un rendimiento en cuanto a su precisión de 99.30%, ya que se trabajó con menos data de imágenes.

El método propuesto tiene mejores resultados en cuanto a sus indicadores, en condiciones como desenfoque, baja resolución, ruido de caracteres, sin embargo, después de analizar los resultados, se concluyó que algunos errores cometidos por el método en algunos casos se deben a un único carácter, como confundir la letra D con O, con 0; la letra B con el número 8 y el número 2 con z, pero estos errores se subsanan configurando el método con algoritmos, filtros o también operadores morfológicos.

4.2 Recomendaciones

Se recomienda configurar el dispositivo celular, con los ajustes de, apertura local con f/2.2, velocidad de obturación con 1/12s y la sensibilidad ISO con 400, esto con la finalidad de facilitar el procesamiento de las imágenes.

Si se requiere obtener resultados similares a esta investigación, se recomienda el uso

de versiones exactas como las librerías NumPy 1.19.2, OpenCV 3.3.1, ya que facilitan considerablemente el trabajo a la hora de codificar el método en cada una de sus etapas del procesamiento de imágenes, puesto que hay operaciones matemáticas difíciles para la programación, como las que tienen algunas técnicas de tratamiento de imágenes, estas deficiencias se reducen a simples funciones con poco código gracias a las librerías mencionadas anteriormente, permitiendo fijarse en el diseño y optimizar el rendimiento del sistema.

REFERENCIAS

- [1] M. Guevara, “El país donde nació la ‘mototaxi’ y su llegada a Sudamérica: es usado como transporte público en más de una nación,” 23-09-2024. [Online]. Available: <https://larepublica.pe/mundo/2024/09/23/el-pais-donde-nacio-la-mototaxi-y-su-llegada-a-sudamerica-es-usado-como-transporte-publico-en-mas-de-una-nacion-240442>
- [2] ASOCIACIÓN AUTOMOTRIZ DEL PERÚ, “ASOCIACIÓN AUTOMOTRIZ DEL PERÚ.” Accessed: Jun. 18, 2025. [Online]. Available: <https://aap.org.pe/informes-estadisticos/diciembre-2019/>
- [3] V. X. N, “Volumen xlvii n.º 2 volumen xxix n.º,” 2022.
- [4] T. S. Rocio, “PROYECTO DE LEY QUE PROPONE EL ACCESO AL SISTEMA DE SEGURIDAD SOCIAL EN SALUD Y PENSIONES A LOS PRESTADORES DE SERVICIO DE TRANSPORTE DE VEHÍCULOS MENORES - MOTOTAXIS.,” vol. 3, pp. 1–10, 2022.
- [5] INEI, “Estadísticas de la Criminalidad, Seguridad Ciudadana y Violencia: Una vision desde los registros administrativos Enero - Noviembre 2022,” *Reportes Mens. INEI*, pp. 1–96, 2022, [Online]. Available: <http://proyecto.inei.gob.pe/enapres/wp-content/uploads/2023/06/Estadísticas-de-Criminalidad-Seguridad-Ciudadana-y-Violencia.-Enero-Noviembre-2022.pdf>
- [6] Policia Nacional del Perú, “1 | P á g i na,” pp. 1–56, 2023.
- [7] Ministerio de Transportes y Comunicaciones, “Decreto Supremo N° 016-2009-Mtc Texto Unico Ordenado Del Reglamento Nacional De Transito,” pp. 1–118, 2020.
- [8] M. Y. Arafat, A. S. M. Khairuddin, U. Khairuddin, and R. Paramesran, “Systematic review on vehicular licence plate recognition framework in intelligent transport systems,” *IET Intell. Transp. Syst.*, vol. 13, no. 5, pp. 745–755, 2019, doi: 10.1049/iet-its.2018.5151.
- [9] W. Weihong and T. Jiaoyang, “Research on License Plate Recognition Algorithms Based on Deep Learning in Complex Environment,” *IEEE Access*, vol. 8, pp. 91661–

- 91675, 2020, doi: 10.1109/ACCESS.2020.2994287.
- [10] G. Y. Abbass and A. F. Marhoon, "Car license plate segmentation and recognition system based on deep learning," *Bull. Electr. Eng. Informatics*, vol. 11, no. 4, pp. 1983–1989, 2022, doi: 10.11591/eei.v11i4.3434.
- [11] R. Antar, J. Alotaibi, M. Alghamdi, and S. Alghamdi, "Automatic Number Plate Recognition of Saudi License Car Plates," *Eng. Technol. Appl. Sci. Res.*, vol. 12, no. 2, pp. 8266–8272, 2022, doi: 10.48084/etasr.4727.
- [12] D. Ullah, Farman, Anwar, Hafeez, Shahzadi, IramUr Rehman, Ata Mehmood, Shizra Niaz, Sania Awan, Khalid Mahmood Khan, Ajmal Kwak, "Barrier access control using sensors platform and vehicle license plate characters recognition," *Sensors (Switzerland)*, vol. 19, no. 13, pp. 1–20, 2019, doi: 10.3390/s19133015.
- [13] L. Pingping and X. Wenjing, "Research on Recognition Technology of License Plate Image," *Proc. - 2nd Int. Conf. Comput. Network, Electron. Autom. ICCNEA 2019*, pp. 52–56, 2019, doi: 10.1109/ICCNEA.2019.00020.
- [14] A. M. Mostfa and O. A. L. A. Ridha, "Design and implementation of vehicles identification and tracking system," *ACM Int. Conf. Proceeding Ser.*, pp. 222–227, 2019, doi: 10.1145/3321289.3321311.
- [15] M. C. Wijaya, "Research of Indonesian License Plates Recognition on Moving Vehicles," *EUREKA, Phys. Eng.*, vol. 2022, no. 6, pp. 185–198, 2022, doi: 10.21303/2461-4262.2022.002424.
- [16] E. S. H. A. Farag, Mohamed S., Mohie El Din M. M., "Parking entrance control using license plate detection and recognition," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 15, no. 1, pp. 476–483, 2019, doi: 10.11591/ijeecs.v15.i1.pp476-483.
- [17] B. V. K. V. Elanangai, "Automated system for defect identification and character recognition using IR images of SS-plates," *Int. J. Recent Technol. Eng.*, vol. 8, no. 3, pp. 6958–6964, 2019, doi: 10.35940/ijrte.C6009.098319.
- [18] R. Islam, M. R. Islam, and K. H. Talukder, "An efficient method for extraction and recognition of bangla characters from vehicle license plates," *Multimed. Tools Appl.*,

- 2020, doi: 10.1007/s11042-020-08629-8.
- [19] O. M. Yousif Bedir Bedir, Ata Mohamed Maher, Fawzy Nehal, "Toward an Optimized Neutrosophic k-Means with Genetic Algorithm for Automatic Vehicle License Plate Recognition (ONKM-AVLPR)," *IEEE Access*, vol. 8, pp. 49285–49312, 2020, doi: 10.1109/ACCESS.2020.2979185.
- [20] Y. Cheon and C. Lee, "License Plate Extraction for Moving Vehicles," *10th Int. Conf. Information, Intell. Syst. Appl. IISA 2019*, pp. 1–6, 2019, doi: 10.1109/IISA.2019.8900778.
- [21] A. P. Fernández and P. Díaz, "La investigación cualitativa y la investigación cuantitativa," *Investig. Educ.*, vol. 7, no. 11, pp. 72–91, 2003.
- [22] Barbara Kitchenham and S. Charters, "Методи за автоматично управление на подедни устройства при Jack-up системите," 2007, doi: 10.1145/1134285.1134500.
- [23] P. Agarwal, K. Chopra, M. Kashif, and V. Kumari, "Implementing ALPR for detection of traffic violations: A step towards sustainability," *Procedia Comput. Sci.*, vol. 132, pp. 738–743, 2018, doi: 10.1016/j.procs.2018.05.085.
- [24] D. Sandha, Somalin, Rana, "Automatic Car Number Plate Recognition System for Authorization," *Circ. Comput. Sci.*, vol. MCSP2017, no. 01, pp. 30–34, 2017, doi: 10.22632/ccs-2017-mcsp036.
- [25] N. Saleem, H. Muazzam, H. M. Tahir, and U. Farooq, "Automatic license plate recognition using extracted features," *2016 4th Int. Symp. Comput. Bus. Intell. ISCBI 2016*, pp. 221–225, 2016, doi: 10.1109/ISCBI.2016.7743288.
- [26] A. Nosseir and R. Roshdy, "Extraction of Egyptian license plate numbers and characters using surf and cross correlation," *ACM Int. Conf. Proceeding Ser.*, pp. 48–55, 2018, doi: 10.1145/3220267.3220276.
- [27] H. S. Kosala Gamma, Harjoko Agus, "License plate detection based on convolutional neural network - Support vector machine (CNN-SVM)," *ACM Int. Conf. Proceeding Ser.*, pp. 1–5, 2017, doi: 10.1145/3177404.3177436.
- [28] S. M. Silva and C. R. Jung, "Real-time license plate detection and recognition using

- deep convolutional neural networks,” *J. Vis. Commun. Image Represent.*, vol. 71, p. 102773, 2020, doi: <https://doi.org/10.1016/j.jvcir.2020.102773>.
- [29] S. Azam and M. Gavrilova, “License plate image patch filtering using HOG descriptor and bio-inspired optimization,” *ACM Int. Conf. Proceeding Ser.*, vol. Part F1286, 2017, doi: 10.1145/3095140.3095141.
- [30] J. C. De Goma, L. A. B. Ammuyutan, H. L. S. Capulong, K. P. Naranjo, and M. Devaraj, “Vehicular Obstruction Detection in the Zebra Lane Using Computer Vision,” *2019 IEEE 6th Int. Conf. Ind. Eng. Appl. ICIEA 2019*, pp. 362–366, 2019, doi: 10.1109/IEA.2019.8715022.
- [31] M. Shehata, M. T. Abou-Kreisha, and H. Elnashar, “Deep machine learning based egyptian vehicle license plate recognition systems,” *arXiv*, pp. 69–76, 2021.
- [32] A. T. Musaddid, A. Bejo, and R. Hidayat, “Improvement of Character Segmentation for Indonesian License Plate Recognition Algorithm using CNN,” *2019 2nd Int. Semin. Res. Inf. Technol. Intell. Syst. ISRITI 2019*, pp. 279–283, 2019, doi: 10.1109/ISRITI48646.2019.9034614.
- [33] A. Kumar Sahoo, “Automatic recognition of Indian vehicles license plates using machine learning approaches,” *Mater. Today Proc.*, vol. 49, pp. 2982–2988, 2022, doi: <https://doi.org/10.1016/j.matpr.2020.09.046>.
- [34] J. F. V. Serrano, A. B. M. Díaz, and Á. S. Calle, “Visión por computador.,” p. 240, 2000.
- [35] J. Gironés-Roig, J. Casas-Roma, J. Minguillón-Alfonso, and R. Caihuelas-Quiles, *Minería de datos: Modelos y Algoritmos* Gironés-Roig, J., Casas-Roma, J., Minguillón-Alfonso, J., & Caihuelas-Quiles, R. (2017). *Minería de datos: Modelos y Algoritmos.*, no. 1. 2017. [Online]. Available: https://www.cambridge.org/core/product/identifier/CBO9781139058452A007/type/book_part
- [36] B. Ta, Tung Duy Le, Duc Anh Le, Mai Thi Tran, Toan Van Do, Tuan Trong Nguyen, Van Duc Trinh, Chien Van Jeon, “Automatic number plate recognition on electronic

- toll collection systems for vietnamese conditions,” *ACM IMCOM 2015 - Proc.*, pp. 1–5, 2015, doi: 10.1145/2701126.2701187.
- [37] G. Z. Khan, Arsalan Mehmood; Awan, Sheryar Mehmood; Arif, Muhammad; Mahmood, Zahid; Khan, “A Robust Segmentation Free License Plate Recognition Method,” *1st Int. Conf. Electr. Commun. Comput. Eng. ICECCE 2019*, no. July, pp. 1–6, 2019, doi: 10.1109/ICECCE47252.2019.8940769.
- [38] M. A. Javeed *et al.*, “Lane Line Detection and Object Scene Segmentation Using Otsu Thresholding and the Fast Hough Transform for Intelligent Vehicles in Complex Road Conditions,” *Electronics*, vol. 12, no. 5, 2023, doi: 10.3390/electronics12051079.
- [39] Lubna, N. Mufti, and S. A. A. Shah, “Automatic Number Plate Recognition : A Detailed Survey of,” *Sensors*, vol. 21, no. 9, p. 3028, 2021, [Online]. Available: <https://www.mdpi.com/1424-8220/21/9/3028/pdf>
- [40] C. C. Peng, C. J. Tsai, T. Y. Chang, J. Y. Yeh, H. Dai, and M. H. Tsai, “A fast and noise tolerable binarization method for automatic license plate recognition in the open environment in Taiwan,” *Symmetry (Basel)*, vol. 12, no. 8, 2020, doi: 10.3390/SYM12081374.
- [41] M. D. C. Velarde and G. Velarde, “Benchmarking Algorithms for Automatic License Plate Recognition,” pp. 1–6, 2022, [Online]. Available: <http://arxiv.org/abs/2203.14298>

ANEXOS

Anexo 1. Acta de revisión de similitud de la investigación.




ACTA DE REVISIÓN DE SIMILITUD DE LA INVESTIGACIÓN

Yo Heber Iván Mejía Cabrera del curso de **Investigación II**, del Programa de Estudios de **Ingeniería de Sistemas**, luego de revisar la investigación del estudiante, Calos Humberto Oliva Villalobos, titulada:

DESARROLLO DE UN MÉTODO DE RECONOCIMIENTO DE PLACAS VEHICULARES DE MOTOTAXI EN AMBIENTES NO CONTROLADOS UTILIZANDO PROCESAMIENTO DIGITAL DE IMÁGENES

Dejo constancia que la investigación antes indicada tiene un índice de similitud del porcentaje 16 %, verificable en el reporte de originalidad mediante el software de similitud TURNITIN. Por lo que se concluye que cada una de las coincidencias detectadas no constituyen plagio y cumple con lo establecido en la Directiva sobre índice de similitud de los productos académicos y de investigación en la Universidad Señor de Sipán S.A.C. vigente.

En virtud de lo antes mencionado, firma:

Heber Iván Mejía Cabrera	DNI: 41639565	
--------------------------	---------------	---

Pimentel, 17 de enero del 2025

Anexo 2. Acta de aprobación de asesor.

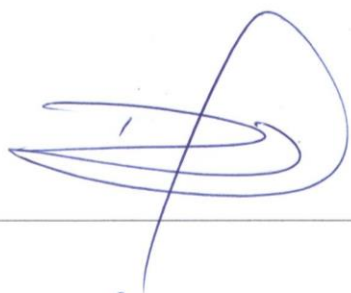

ACTA DE APROBACIÓN DEL ASESOR



ACTA DE APROBACIÓN DEL ASESOR

Yo **Bances Saavedra David Enrique** quien suscribe como asesor designado mediante Resolución de Facultad N° 0426-2023/FIAU-USS, del proyecto de investigación titulado **DESARROLLO DE UN MÉTODO DE RECONOCIMIENTO DE PLACAS VEHICULARES DE MOTOTAXI EN AMBIENTES NO CONTROLADOS UTILIZANDO PROCESAMIENTO DIGITAL DE IMÁGENES**, desarrollado por el estudiante: **OLIVA VILLALOBOS CARLOS HUMBERTO**, del programa de estudios de **Ingeniería de Sistemas**, acredito haber revisado, realizado observaciones y recomendaciones pertinentes, encontrándose expedito para su revisión por parte del docente del curso.

En virtud de lo antes mencionado, firman:

Bances Saavedra David Enrique (Asesor)	40444130	
Oliva Villalobos Carlos Humberto (Autor 1)	40522099	

Pimentel 04 de diciembre del 2024

Anexo 3. Tabla de operacionalización de variables.

Variables	Dimensión	Indicador	Ítem	Técnica e instrumentos de recolección de datos
		Grado de consumo de CPU	$Cc = \sum_j^n \frac{cc_j}{n}$	
Variables independientes: método de procesamiento digital de imágenes	Consumo de recursos	Grado de consumo de memoria	$Cm = \sum_j^n \frac{cm_j}{n}$	Instrumentos mecánicos electrónicos / Registro electrónico.
		Tiempo promedio de respuesta	$TR = TF - TI$	
Variables	Rendimiento	Exactitud		

dependientes:
Reconocimiento
de placas

$$E = \frac{VP + VN}{VP + VN + FP + FN}$$

Precisión

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

Exhaustividad,
Sensibilidad o
Recall

$$R = \frac{VP}{VP + FN}$$

Anexo 4. Matriz de consistencia.

DESARROLLO DE UN METODO DE RECONOCIMIENTO DE PLACAS VEHICULARES DE MOTOTAXI EN AMBIENTES NO CONTROLADOS UTILIZANDO PROCESAMIENTO DIGITAL DE IMÁGENES					
Título					
Problema	Hipótesis	Objetivo General	Objetivos específicos	Tipo de investigación	Diseño de investigación
¿De qué manera se podrá reconocer placas vehiculares de mototaxis en ambientes no controlados?	Si se desarrolla un método de procesamiento digital de imágenes entonces se podrá reconocer placas vehiculares de mototaxi en ambientes no controlados.	Desarrollar un método de reconocimiento de placas vehiculares de mototaxi en ambientes no controlados utilizando procesamiento digital de imágenes.	a) Construir el dataset de imágenes de placas de mototaxi. b) Seleccionar el método a implementar. c) Implementar el método seleccionado en un lenguaje de programación. d) Ejecutar pruebas de reconocimiento de placa de mototaxis. e) Analizar los resultados.	La investigación es cuantitativa, aplicada y tecnológica.	El diseño es cuasi-experimental

Anexo 5. Instrumentos de recolección de datos validados.

Formato de grado de consumo de CPU.

Dimensión	Valor
Uso	
Velocidad	
Procesos	
Subprocesos	
Identificadores	
Tiempo activo	

Formato grado de consumo de memoria RAM.

Dimensión	Valor
En uso comprimido	
Disponible	
Confirmada	
En caché	
Tiempo	

Ítem	Valor
Verdadero positivo (VP)	
Falso Positivo (FP)	
Verdadero Negativo (VN)	
Falso Negativo (FN)	

Ítem	Valor
Precisión	
Exactitud	
Recall	
F1-Score	

Anexo 6. Código fuente o documentación fuente del producto de ingeniería.

Módulo: DetectarCaracter.py Funciones y Técnicas:

```
# DetectChars.py
import os
import cv2
import numpy as np
import math
import random
import Main
import Preprocesamiento
import PosibleCaracter

# variables de nivel de modulo

kNearest = cv2.ml.KNearest_create()

# constantes para checkIfPossibleChar, esto verifica solo un carácter posible (no
se compara con otro carácter)
MIN_PIXEL_WIDTH = 2
MIN_PIXEL_HEIGHT = 8

MIN_ASPECT_RATIO = 0.25
MAX_ASPECT_RATIO = 1.0

MIN_PIXEL_AREA = 80

# constantes para comparar 2 caracteres
MIN_DIAG_SIZE_MULTIPLE_AWAY = 0.3
MAX_DIAG_SIZE_MULTIPLE_AWAY = 5.0

MAX_CHANGE_IN_AREA = 0.5

MAX_CHANGE_IN_WIDTH = 0.8
MAX_CHANGE_IN_HEIGHT = 0.2

MAX_ANGLE_BETWEEN_CHARS = 12.0

# otras constantes
MIN_NUMBER_OF_MATCHING_CHARS = 3

RESIZED_CHAR_IMAGE_WIDTH = 20
RESIZED_CHAR_IMAGE_HEIGHT = 30

MIN_CONTOUR_AREA = 100

#####
def cargarDatosKNNyEntrenamientoKNN():
    todosLosContornosConDatos = [] # declarar listas vacias,
    contornosValidosConDatos = [] # los llenaremos en breve

    try:
        npaclasificacion = np.loadtxt("clasificacion.txt", np.float32) #
        leer en clasificaciones de entrenamiento
    except:
        # si el archivo no se pudo abrir
        print("error, no se puede abrir clasificacion.txt, programa de salida\n") #
        muestra el mensaje de error
        os.system("pausa")
        return False
    # y devuelve falso
    # end try

    try:
        npaImágenesAplanadas = np.loadtxt("imagenes_aplanadas.txt", np.float32)
        # leer en las imágenes de entrenamiento
    except:
        # si el archivo no se pudo abrir
        print("error, no se puede abrir imagenes_aplanadas.txt, saliendo del programa\n")
        # muestra el mensaje de error
        os.system("pause")
        return False
    # y delve falso
```

```

# end try

npaclasificacion = npaclasificacion.reshape((npaclasificacion.size, 1)) #
remodelar la matriz numpy a 1d, necesario pasar para llamar a entrenar
kNearest.setDefaultK(1) #
establece el valor predeterminado de K en 1

kNearest.train(npaImágenesAplanadas, cv2.ml.ROW_SAMPLE, npaclasificacion) #
entrenar objeto KNN

return True # si llegamos aquí, el entrenamiento fue
exitoso, así que devuelva verdadero
# end function

#####
def detectarCaracteresEnPlacas(listaPosiblePlacas):
    intContadorDePlaca = 0
    imgContornos = None
    contours = []

    if len(listaPosiblePlacas) == 0: # si la lista de posibles placas esta vacia
        return listaPosiblePlacas # retornar
    # end if

    # en este punto podemos estar seguros de que la lista de posibles placas tiene
    al menos una placa

    for posiblePlaca in listaPosiblePlacas: # para cada placa posible, este es un
        gran bucle for que ocupa la mayor parte de la función

            posiblePlaca.imgEscalaDeGrises, posiblePlaca.imgUmbral =
            Preprocesamiento.preprocesamiento(posiblePlaca.imgPlaca) # preproceso para obtener
            imágenes en escala de grises y umbral

            if Main.MostrarPasos == True: # mostrar pasos
                cv2.imshow("5a", posiblePlaca.imgPlaca)
                cv2.imshow("5b", posiblePlaca.imgEscalaDeGrises)
                cv2.imshow("5c", posiblePlaca.imgUmbral)
            # end if # show steps
            #####

            # Aumente el tamaño de la imagen de la placa para facilitar la
            visualización y la detección de carbonilla.
            posiblePlaca.imgUmbral = cv2.resize(posiblePlaca.imgUmbral, (0, 0), fx = 1.6, fy =
            1.6)

            # umbral de nuevo para eliminar las áreas grises
            thresholdValue, posiblePlaca.imgUmbral = cv2.threshold(posiblePlaca.imgUmbral, 0.0,
            255.0, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

            if Main.MostrarPasos == True: # mostrar pasos
                #####
                cv2.imshow("5d", posiblePlaca.imgUmbral)
            # end if # show steps
            #####

            # encuentra todos los caracteres posibles en la placa,
            # esta función primero encuentra todos los contornos, luego solo incluye
            los contornos que podrían ser caracteres (sin comparación con otros caracteres todavía)
            listaDePosiblesCaracteresEnLaPlaca =
            encontrarCaracteresPosiblesEnPlaca(posiblePlaca.imgEscalaDeGrises, posiblePlaca.imgUmbral)

            if Main.MostrarPasos == True: # mostrar pasos
                #####
                height, width, numChannels = posiblePlaca.imgPlaca.shape
                imgContornos = np.zeros((height, width, 3), np.uint8)
                del contours[:] # borrar la lista de
            contornos

            for posibleCaracter in listaDePosiblesCaracteresEnLaPlaca:
                contours.append(posibleCaracter.contour)
            # end for

            cv2.drawContours(imgContornos, contours, -1, Main.SCALAR_WHITE)

            cv2.imshow("6", imgContornos)

```

```

# end if # show steps
#####

# dada una lista de todos los caracteres posibles, busque grupos de
caracteres coincidentes dentro de la placa
listaDeListasDeCaracteresCoincidentesEnLaPlaca =
buscarListaDeListasDeCaracteresCoincidentes(listaDePosiblesCaracteresEnLaPlaca)

if Main.MostrarPasos == True: # show steps
#####
imgContornos = np.zeros((height, width, 3), np.uint8)
del contours[:]

for listaDeCaracteresCoincidentes in
listaDeListasDeCaracteresCoincidentesEnLaPlaca:
    intRandomBlue = random.randint(0, 255)
    intRandomGreen = random.randint(0, 255)
    intRandomRed = random.randint(0, 255)

    for matchingChar in listaDeCaracteresCoincidentes:
        contours.append(matchingChar.contour)
    # end for
    cv2.drawContours(imgContornos, contours, -1, (intRandomBlue,
intRandomGreen, intRandomRed))
    # end for
    cv2.imshow("7", imgContornos)
# end if # show steps
#####

if (len(listaDeListasDeCaracteresCoincidentesEnLaPlaca) == 0):
# si no se encontraron grupos de caracteres coincidentes en la placa

if Main.MostrarPasos == True: # show steps
#####
print("caracteres encontrados en el número de placa " + str(
intContadorDePlaca) + " = (ninguno), haga clic en cualquier imagen y
presione una tecla para continuar . . .")
intContadorDePlaca = intContadorDePlaca + 1
cv2.destroyWindow("8")
cv2.destroyWindow("9")
cv2.destroyWindow("10")
cv2.waitKey(0)
# end if # show steps
#####

posiblePlaca.strChars = ""
continue # volver al
principio del bucle for
# end if

for i in range(0, len(listaDeListasDeCaracteresCoincidentesEnLaPlaca)): # dentro de
cada lista de caracteres coincidentes
    listaDeListasDeCaracteresCoincidentesEnLaPlaca[i].sort(key = lambda
matchingChar: matchingChar.intCenterX) # ordenar caracteres de izquierda a derecha
    listaDeListasDeCaracteresCoincidentesEnLaPlaca[i] =
eliminarCaracteresSuperpuestosInternos(listaDeListasDeCaracteresCoincidentesEnLaPlaca[i]) #
y eliminar caracteres internos superpuestos

if Main.MostrarPasos == True: # show steps
#####
imgContornos = np.zeros((height, width, 3), np.uint8)

for listaDeCaracteresCoincidentes in
listaDeListasDeCaracteresCoincidentesEnLaPlaca:
    intRandomBlue = random.randint(0, 255)
    intRandomGreen = random.randint(0, 255)
    intRandomRed = random.randint(0, 255)

    del contours[:]

    for matchingChar in listaDeCaracteresCoincidentes:
        contours.append(matchingChar.contour)
    # end for

    cv2.drawContours(imgContornos, contours, -1, (intRandomBlue,
intRandomGreen, intRandomRed))
    # end for

```

```

        cv2.imshow("8", imgContornos)
        # end if # show steps
#####

        # dentro de cada placa posible, suponga que la lista más larga de
        caracteres coincidentes potenciales es la lista real de caracteres
        intLenOfLongestListOfChars = 0
        intIndexOfLongestListOfChars = 0

        # recorrer todos los vectores de caracteres coincidentes, obtener el índice
        del que tiene más caracteres
        for i in range(0, len(listaDeListasDeCaracteresCoincidentesEnLaPlaca)):
            if len(listaDeListasDeCaracteresCoincidentesEnLaPlaca[i]) >
intLenOfLongestListOfChars:
                intLenOfLongestListOfChars =
len(listaDeListasDeCaracteresCoincidentesEnLaPlaca[i])
                intIndexOfLongestListOfChars = i
            # end if
        # end for

        # suponga que la lista más larga de caracteres coincidentes dentro de la
        placa es la lista real de caracteres
        longestlistaDeCaracteresCoincidentesInPlate =
listaDeListasDeCaracteresCoincidentesEnLaPlaca[intIndexOfLongestListOfChars]

        if Main.MostrarPasos == True: # show steps
#####
            imgContornos = np.zeros((height, width, 3), np.uint8)
            del contours[:]

            for matchingChar in longestlistaDeCaracteresCoincidentesInPlate:
                contours.append(matchingChar.contour)
            # end for

            cv2.drawContours(imgContornos, contours, -1, Main.SCALAR_WHITE)

            cv2.imshow("9", imgContornos)
            # end if # show steps
#####

            posiblePlaca.strChars = reconocerCaracteresEnPlaca(posiblePlaca.imgUmbral,
longestlistaDeCaracteresCoincidentesInPlate)

            if Main.MostrarPasos == True: # show steps
#####
                print("caracteres encontrados en el número de placa " + str(
                    intContadorDePlaca) + " = " + posiblePlaca.strChars + ", haga clic en
cualquier imagen y presione una tecla para continuar . . .")
                intContadorDePlaca = intContadorDePlaca + 1
                cv2.waitKey(0)
            # end if # show steps
#####

            # end of big for loop that takes up most of the function

            if Main.MostrarPasos == True:
                print("\ndetección de caracteres completa, haga clic en cualquier imagen y presione
una tecla para continuar . . .\n")
                cv2.waitKey(0)
            # end if

            return listaPosiblePlacas
        # end function

#####
#####
def encontrarCaracteresPosiblesEnPlaca(imgEscalaDeGrises, imgUmbral):
    listaDePosibleCaracteres = [] # this will be the return value
    contours = []
    imgUmbralCopy = imgUmbral.copy()

    # find all contours in plate
    _, contours, npaHierarchy = cv2.findContours(imgUmbralCopy, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE)

    for contour in contours: # for each contour
        posibleCaracter = PosibleCaracter.PossibleCaracter(contour)

```

```

        if comprobarSiEsPosible(possibleCharacter):                # if contour is a possible
char, note this does not compare to other chars (yet) . . .
            listaDePosibleCaracteres.append(possibleCharacter)    # add to list of
possible chars
        # end if
    # end if

    return listaDePosibleCaracteres
# end function

#####
#####
def comprobarSiEsPosible(possibleCharacter):
    # this function is a 'first pass' that does a rough check on a contour to see
if it could be a char,
    # note that we are not (yet) comparing the char to other chars to look for a
group
    if (possibleCharacter.intBoundingRectArea > MIN_PIXEL_AREA and
        possibleCharacter.intBoundingRectWidth > MIN_PIXEL_WIDTH and
possibleCharacter.intBoundingRectHeight > MIN_PIXEL_HEIGHT and
        MIN_ASPECT_RATIO < possibleCharacter.fltAspectRatio and
possibleCharacter.fltAspectRatio < MAX_ASPECT_RATIO):
        return True
    else:
        return False
    # end if
# end function

#####
#####
def buscarListaDeListasDeCaracteresCoincidentes(listaDePosibleCaracteres):
    # with this function, we start off with all the possible chars in one big list
    # the purpose of this function is to re-arrange the one big list of chars into
a list of lists of matching chars,
    # note that chars that are not found to be in a group of matches do not need to
be considered further
    listOfListsOfMatchingChars = []                # this will be the return value

    for possibleCharacter in listaDePosibleCaracteres:            # for each
possible char in the one big list of chars
        listaDeCaracteresCoincidentes =
buscarListaDeCaracteresCoincidentes(possibleCharacter, listaDePosibleCaracteres)    #
find all chars in the big list that match the current char

        listaDeCaracteresCoincidentes.append(possibleCharacter)    # also add the
current char to current possible list of matching chars

        if len(listaDeCaracteresCoincidentes) < MIN_NUMBER_OF_MATCHING_CHARS:    # if
current possible list of matching chars is not long enough to constitute a possible plate
            continue                # jump back to the top of the for loop and
try again with next char, note that it's not necessary
            # to save the list in any way since it did
not have enough chars to be a possible plate
        # end if

        # if we get here, the current list passed
test as a "group" or "cluster" of matching chars
        listOfListsOfMatchingChars.append(listaDeCaracteresCoincidentes)    # so add to
our list of lists of matching chars

        listaDePosibleCaracteresWithCurrentMatchesRemoved = []

        # remove the current list of matching chars
from the big list so we don't use those same chars twice,
        # make sure to make a new big list for this
since we don't want to change the original big list
        listaDePosibleCaracteresWithCurrentMatchesRemoved =
list(set(listaDePosibleCaracteres) - set(listaDeCaracteresCoincidentes))

        recursiveListOfListsOfMatchingChars =
buscarListaDeListasDeCaracteresCoincidentes(listaDePosibleCaracteresWithCurrentMatchesRemov
ed)    # recursive call

        for recursivelistaDeCaracteresCoincidentes in recursiveListOfListsOfMatchingChars:
# for each list of matching chars found by recursive call
            listOfListsOfMatchingChars.append(recursivelistaDeCaracteresCoincidentes)

```

```

# add to our original list of lists of matching chars
# end for

    break          # exit for

# end for

    return listOfListsOfMatchingChars
# end function

#####
#####
def buscarListaDeCaracteresCoincidentes(possibleCaracter, listOfChars):
    # the purpose of this function is, given a possible char and a big list of
    possible chars,
    # find all chars in the big list that are a match for the single possible char,
    and return those matching chars as a list
    listaDeCaracteresCoincidentes = []          # this will be the return value

    for possibleMatchingChar in listOfChars:    # for each char in big list
        if possibleMatchingChar == possibleCaracter: # if the char we attempting to find
            matches for is the exact same char as the char in the big list we are currently checking
            # then we should not include it in the
            list of matches b/c that would end up double including the current char
            continue          # so do not add to list of matches and
            jump back to top of for loop
        # end if

        # compute stuff to see if chars are a match
        fltdistanciaEntreCaracteres = distanciaEntreCaracteres(possibleCaracter,
        possibleMatchingChar)

        fltanguloEntreCaracteres = anguloEntreCaracteres(possibleCaracter,
        possibleMatchingChar)

        fltChangeInArea = float(abs(possibleMatchingChar.intBoundingRectArea -
        possibleCaracter.intBoundingRectArea)) / float(possibleCaracter.intBoundingRectArea)

        fltChangeInWidth = float(abs(possibleMatchingChar.intBoundingRectWidth -
        possibleCaracter.intBoundingRectWidth)) / float(possibleCaracter.intBoundingRectWidth)
        fltChangeInHeight = float(abs(possibleMatchingChar.intBoundingRectHeight -
        possibleCaracter.intBoundingRectHeight)) / float(possibleCaracter.intBoundingRectHeight)

        # check if chars match
        if (fltdistanciaEntreCaracteres < (possibleCaracter.fltdiagonalSize *
        MAX_DIAG_SIZE_MULTIPLE_AWAY) and
            fltanguloEntreCaracteres < MAX_ANGLE_BETWEEN_CHARS and
            fltChangeInArea < MAX_CHANGE_IN_AREA and
            fltChangeInWidth < MAX_CHANGE_IN_WIDTH and
            fltChangeInHeight < MAX_CHANGE_IN_HEIGHT):

            listaDeCaracteresCoincidentes.append(possibleMatchingChar)          # if the
            chars are a match, add the current char to list of matching chars
        # end if
    # end for

    return listaDeCaracteresCoincidentes          # return result
# end function

#####
#####
# use Pythagorean theorem to calculate distance between two chars
def distanciaEntreCaracteres(firstChar, secondChar):
    intX = abs(firstChar.intCenterX - secondChar.intCenterX)
    intY = abs(firstChar.intCenterY - secondChar.intCenterY)

    return math.sqrt((intX ** 2) + (intY ** 2))
# end function

#####
#####
# use basic trigonometry (SOH CAH TOA) to calculate angle between chars
def anguloEntreCaracteres(firstChar, secondChar):
    fltAdj = float(abs(firstChar.intCenterX - secondChar.intCenterX))
    fltOpp = float(abs(firstChar.intCenterY - secondChar.intCenterY))

    if fltAdj != 0.0:          # check to make sure we do not divide by
        zero if the center X positions are equal, float division by zero will cause a crash in

```



```

Python
    fltAngleInRad = math.atan(fltOpp / fltAdj)      # if adjacent is not zero,
calculate angle
    else:
        fltAngleInRad = 1.5708                    # if adjacent is zero, use this as
the angle, this is to be consistent with the C++ version of this program
        # end if

    fltAngleInDeg = fltAngleInRad * (180.0 / math.pi)      # calculate angle in degrees

    return fltAngleInDeg
# end function

#####
# if we have two chars overlapping or too close to each other to possibly be separate chars,
remove the inner (smaller) char,
# this is to prevent including the same char twice if two contours are found for the same
char,
# for example for the letter 'O' both the inner ring and the outer ring may be found as
contours, but we should only include the char once
def eliminarCaracteresSuperpuestosInternos(listaDeCaracteresCoincidentes):
    listaDeCaracteresCoincidentesWithInnerCharRemoved = list(listaDeCaracteresCoincidentes)
# this will be the return value

    for caracterActual in listaDeCaracteresCoincidentes:
        for otherChar in listaDeCaracteresCoincidentes:
            if caracterActual != otherChar:          # if current char and other char are not
the same char . . .
                                                    # if current
char and other char have center points at almost the same location . . .
                if distanciaEntreCaracteres(caracterActual, otherChar) <
(caracterActual.fltDiagonalSize * MIN_DIAG_SIZE_MULTIPLE_AWAY):
                    # if we get in here we have found overlapping chars
                    # next we identify which char is smaller, then if that char
was not already removed on a previous pass, remove it
                    if caracterActual.intBoundingRectArea < otherChar.intBoundingRectArea:
# if current char is smaller than other char
                        if caracterActual in
listaDeCaracteresCoincidentesWithInnerCharRemoved:          # if current char was not
already removed on a previous pass . . .

listaDeCaracteresCoincidentesWithInnerCharRemoved.remove(caracterActual)      # then
remove current char
                        # end if
                    else:
# else if other char is smaller than current char
                        if otherChar in listaDeCaracteresCoincidentesWithInnerCharRemoved:
# if other char was not already removed on a previous pass . . .

listaDeCaracteresCoincidentesWithInnerCharRemoved.remove(otherChar)          # then remove
other char
                        # end if
                    # end if
                # end if
            # end if
        # end for
    # end for

    return listaDeCaracteresCoincidentesWithInnerCharRemoved
# end function

#####
# aquí es donde aplicamos el reconocimiento de caracteres real
def reconocerCaracteresEnPlaca(imgUmbral, listaDeCaracteresCoincidentes):
    strChars = ""      # este será el valor de retorno, los caracteres en la placa

    height, width = imgUmbral.shape

    imgUmbralColor = np.zeros((height, width, 3), np.uint8)

    listaDeCaracteresCoincidentes.sort(key = lambda matchingChar: matchingChar.intCenterX)#
ordenar caracteres de izquierda a derecha

    cv2.cvtColor(imgUmbral, cv2.COLOR_GRAY2BGR, imgUmbralColor)# hacer la versión en color
de la imagen de umbral para que podamos dibujar contornos en color en ella

```

```

    for caracterActual in listaDeCaracteresCoincidentes:# para cada carácter en la placa
        pt1 = (caracterActual.intBoundingRectX, caracterActual.intBoundingRectY)
        pt2 = ((caracterActual.intBoundingRectX + caracterActual.intBoundingRectWidth),
(caracterActual.intBoundingRectY + caracterActual.intBoundingRectHeight))

        cv2.rectangle(imgUmbralColor, pt1, pt2, Main.SCALAR_RED, 2) # dibuja un cuadro rojo
alrededor del carácter

        # recortar char fuera de la imagen de umbral
        imgROI = imgUmbral[caracterActual.intBoundingRectY :
caracterActual.intBoundingRectY + caracterActual.intBoundingRectHeight,
caracterActual.intBoundingRectX :
caracterActual.intBoundingRectX + caracterActual.intBoundingRectWidth]

        imgROIResized = cv2.resize(imgROI, (RESIZED_CHAR_IMAGE_WIDTH,
RESIZED_CHAR_IMAGE_HEIGHT)) # N° de imagen de cambio de tamaño, esto es necesario para el
reconocimiento carácter

        npaROIResized = imgROIResized.reshape((1, RESIZED_CHAR_IMAGE_WIDTH *
RESIZED_CHAR_IMAGE_HEIGHT)) # aplanar la imagen en una matriz numérica 1d

        npaROIResized = np.float32(npaROIResized)# convertir de 1d numpy array of ints a 1d
numpy array de flotadores

        retval, npaResults, neigh_resp, dists = kNearest.findNearest(npaROIResized, k = 1)
# finalmente podemos llamar a findNearest !!!

        strcaracterActual = str(chr(int(npaResults[0][0])))# obtener carácter de los
resultados

        strChars = strChars + strcaracterActual # agregar el carácter actual a la cad
# end for

    if Main.MostrarPasos == True: # show steps
#####
        cv2.imshow("10", imgUmbralColor)
        # end if # show steps #####ena
completa
#####

    return strChars
# end function

```

Módulo: DetectarPlaca.py

Funciones y Técnicas:

```

# DetectPlates.py

import cv2
import numpy as np
import math
import Main
import random

import Preprocesamiento
import DetectarCaracter
import PosiblePlaca
import PosibleCaracter

# module level variables
#####
PLATE_WIDTH_PADDING_FACTOR = 1.3
PLATE_HEIGHT_PADDING_FACTOR = 1.5

#####
def detectarPlacaEnEscena(imgEscenaOriginal):
    """
    :rtype:
    """
    listaPosiblePlacas = []# este será el valor de retorno

```

```

height, width, numChannels = imgEscenaOriginal.shape

imgEscalaDeGrisesScene = np.zeros((height, width, 1), np.uint8)
imgUmbralScene = np.zeros((height, width, 1), np.uint8)
imgContornos = np.zeros((height, width, 3), np.uint8)

cv2.destroyAllWindows()

if Main.MostrarPasos == True: # show steps
#####
cv2.imshow("0", imgEscenaOriginal)
# end if # show steps
#####

imgEscalaDeGrisesScene, imgUmbralScene =
Preprocesamiento.preprocesamiento(imgEscenaOriginal)# preproceso para obtener imágenes en
escala de grises y umbral

if Main.MostrarPasos == True: # show steps
#####
cv2.imshow("Imagen en escala de grises", imgEscalaDeGrisesScene)
cv2.imshow("Imagen umbralizada", imgUmbralScene)
# end if # show steps
#####

# encuentra todos los caracteres posibles en la escena,
# esta función primero encuentra todos los contornos, luego solo incluye los
contornos que podrían ser caracteres (sin comparación con otros caracteres todavía)
listaDePosiblesCaracteresEnEscena =
encontrarPosiblePersonajesEnLaEscena(imgUmbralScene)

if Main.MostrarPasos == True: # show steps
#####
print("paso 2 - len(listaDePosiblesCaracteresEnEscena) = " + str(
len(listaDePosiblesCaracteresEnEscena))) # 131 with MCLRNf1 image

imgContornos = np.zeros((height, width, 3), np.uint8)

contours = []

for posibleCaracter in listaDePosiblesCaracteresEnEscena:
contours.append(posibleCaracter.contour)
# end for

cv2.drawContours(imgContornos, contours, -1, Main.SCALAR_WHITE)
cv2.imshow("2b", imgContornos)
# end if # show steps
#####

# dada una lista de todos los caracteres posibles, busque grupos de caracteres
coincidentes
# en los siguientes pasos, cada grupo de caracteres coincidentes intentará ser
reconocido como una placa
listaDeListasDePersonajesCoincidentesEnLaEscena =
DetectarCaracter.buscarListaDeListasDeCaracteresCoincidentes(listaDePosiblesCaracteresEnEscena)

if Main.MostrarPasos == True: # show steps
#####
print("paso 3 - listaDeListasDePersonajesCoincidentesEnLaEscena.Count = " + str(
len(listaDeListasDePersonajesCoincidentesEnLaEscena))) # 13 with MCLRNf1 image

imgContornos = np.zeros((height, width, 3), np.uint8)

for listaDeCaracteresCoincidentes in
listaDeListasDePersonajesCoincidentesEnLaEscena:
intRandomBlue = random.randint(0, 255)
intRandomGreen = random.randint(0, 255)
intRandomRed = random.randint(0, 255)

contours = []

for matchingChar in listaDeCaracteresCoincidentes:
contours.append(matchingChar.contour)
# end for

```

```

        cv2.drawContours(imgContornos, contours, -1, (intRandomBlue, intRandomGreen,
intRandomRed))
        # end for

        cv2.imshow("3", imgContornos)
        # end if # show steps
#####

        for listaDeCaracteresCoincidentes in listaDeListasDePersonajesCoincidentesEnLaEscena: #
para cada grupo de caracteres coincidentes
            posiblePlaca = extraerPlaca(imgEscenaOriginal, listaDeCaracteresCoincidentes) #
intento de extraer la placa

            if posiblePlaca.imgPlaca is not None: # si se encontró la placa
                listaPosiblePlacas.append(posiblePlaca) # agregar a la lista de posibles placas
            # end if
        # end for

        print("\n" + str(len(listaPosiblePlacas)) + " posible placa encontrada") # 13 with
MCLRNF1 image

        if Main.MostrarPasos == True: # show steps
#####
            print("\n")
            cv2.imshow("4a", imgContornos)

            for i in range(0, len(listaPosiblePlacas)):
                p2fRectPoints =
cv2.boxPoints(listaPosiblePlacas[i].ubicacionDeLaPlacaEnLaEscena)

                cv2.line(imgContornos, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]),
Main.SCALAR_RED, 2)
                cv2.line(imgContornos, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]),
Main.SCALAR_RED, 2)
                cv2.line(imgContornos, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]),
Main.SCALAR_RED, 2)
                cv2.line(imgContornos, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]),
Main.SCALAR_RED, 2)

                cv2.imshow("4a", imgContornos)

                print("possible plate " + str(i) + ", haga clic en cualquier imagen y presione
una tecla para continuar . . .")

                cv2.imshow("4b", listaPosiblePlacas[i].imgPlaca)
                cv2.waitKey(0)
            # end for

            print("\ndetección de placa completa, haga clic en cualquier imagen y presione una
tecla para comenzar el reconocimiento de caracteres . . .\n")
            cv2.waitKey(0)
        # end if # show steps
#####

        return listaPosiblePlacas
# end function

#####
#####
def encontrarPosiblePersonajesEnLaEscena(imgUmbral):
    listaDePosibleCaracteres = [] # este será el valor de retorno

    intRecuentoDeCaracteresPosibles = 0

    imgUmbralCopy = imgUmbral.copy()

    _, contours, npaHierarchy = cv2.findContours(imgUmbralCopy, cv2.RETR_LIST,
cv2.CHAIN_APPROX_SIMPLE) # encuentra todos los contornos

    height, width = imgUmbral.shape
    imgContornos = np.zeros((height, width, 3), np.uint8)

    for i in range(0, len(contours)): # for each contour

        if Main.MostrarPasos == True: # show steps
#####

```

```

        cv2.drawContours(imgContornos, contours, i, Main.SCALAR_WHITE)
        # end if # show steps
#####

        posibleCaracter = PosibleCaracter.PossibleCaracter(contours[i])

        if DetectarCaracter.comprobarSiEsPosible(posibleCaracter): # if
contour is a possible char, note this does not compare to other chars (yet) . . .
            intRecuentoDeCaracteresPosibles = intRecuentoDeCaracteresPosibles + 1
# increment count of possible chars
            listaDePosibleCaracteres.append(posibleCaracter) # and
add to list of possible chars
        # end if
    # end for

    if Main.MostrarPasos == True: # show steps
#####
        print("\npaso 2 - len(contours) = " + str(len(contours))) # 2362 with MCLRNF1
image
        print("paso 2 - intRecuentoDeCaracteresPosibles = " +
str(intRecuentoDeCaracteresPosibles) # 131 with MCLRNF1 image
            cv2.imshow("2a", imgContornos)
        # end if # show steps
#####

    return listaDePosibleCaracteres
# end function

#####
#####
def extraerPlaca(imgOriginal, listaDeCaracteresCoincidentes):
    posiblePlaca = PosiblePlaca.PossiblePlaca() # este sera el valor de retorno

    listaDeCaracteresCoincidentes.sort(key = lambda matchingChar: matchingChar.intCenterX)
# ordenar caracteres de izquierda a derecha según la posición x

    # calcular el punto central de la placa
    fltCentroDePlacaX = (listaDeCaracteresCoincidentes[0].intCenterX +
listaDeCaracteresCoincidentes[len(listaDeCaracteresCoincidentes) - 1].intCenterX) / 2.0
    fltCentroDePlacaY = (listaDeCaracteresCoincidentes[0].intCenterY +
listaDeCaracteresCoincidentes[len(listaDeCaracteresCoincidentes) - 1].intCenterY) / 2.0

    ptCentroDePlaca = fltCentroDePlacaX, fltCentroDePlacaY

    # calcular el ancho y la altura de la placa
    intAnchoDeLaPlaca =
int((listaDeCaracteresCoincidentes[len(listaDeCaracteresCoincidentes) - 1].intBoundingRectX
+ listaDeCaracteresCoincidentes[len(listaDeCaracteresCoincidentes) -
1].intBoundingRectWidth - listaDeCaracteresCoincidentes[0].intBoundingRectX) *
PLATE_WIDTH_PADDING_FACTOR)

    intAlturaTotalDelCaracter = 0

    for matchingChar in listaDeCaracteresCoincidentes:
        intAlturaTotalDelCaracter = intAlturaTotalDelCaracter +
matchingChar.intBoundingRectHeight
    # end for

    fltAlturaPromedioDelCaracter = intAlturaTotalDelCaracter /
len(listaDeCaracteresCoincidentes)

    intAltoDeLaPlaca = int(fltAlturaPromedioDelCaracter * PLATE_HEIGHT_PADDING_FACTOR)

    # calcular el ángulo de corrección de la región de la placa
    fltOpuesto = listaDeCaracteresCoincidentes[len(listaDeCaracteresCoincidentes) -
1].intCenterY - listaDeCaracteresCoincidentes[0].intCenterY
    fltHipotenusa =
DetectarCaracter.distanciaEntreCaracteres(listaDeCaracteresCoincidentes[0],
listaDeCaracteresCoincidentes[len(listaDeCaracteresCoincidentes) - 1])
    fltAnguloDeCorreccionEnRad = math.asin(fltOpuesto / fltHipotenusa)
    fltAnguloDeCorreccionEnGrados = fltAnguloDeCorreccionEnRad * (180.0 / math.pi)

    # Empaque el punto central de la región de la placa, el ancho y la altura, y el
ángulo de corrección en el miembro recto girado variable de la placa
    posiblePlaca.ubicacionDeLaPlacaEnLaEscena = ( tuple(ptCentroDePlaca),
(intAnchoDeLaPlaca, intAltoDeLaPlaca), fltAnguloDeCorreccionEnGrados )

```

```

        # los pasos finales son para realizar la rotación real

        # obtener la matriz de rotación para nuestro ángulo de corrección calculado
        rotationMatrix = cv2.getRotationMatrix2D(tuple(ptCentroDePlaca),
        fltAnguloDeCorreccionEnGrados, 1.0)

        height, width, numChannels = imgOriginal.shape      # desempaquetar el ancho y alto de
        la imagen original

        imgGirada = cv2.warpAffine(imgOriginal, rotationMatrix, (width, height))      # rotar
        toda la imagen

        imgRecortada = cv2.getRectSubPix(imgGirada, (intAnchoDeLaPlaca, intAltoDeLaPlaca),
        tuple(ptCentroDePlaca))

        posiblePlaca.imgPlaca = imgRecortada      # copia la imagen de la placa recortada en
        la variable de miembro aplicable de la placa posible

        return posiblePlaca
    # end function

```

Módulo: Main.py

Funciones y Técnicas:

```

import cv2
import os
import psutil
import time
from sklearn.metrics import precision_score, recall_score, accuracy_score

import DetectarCaracter
import DetectarPlaca
import PosiblePlaca

# Variables del módulo
SCALAR RED = (0.0, 0.0, 255.0)
MostrarPasos = False

def main():
    # Medir el consumo inicial de recursos
    cpu_inicial = psutil.cpu_percent(interval=1)
    memoria_inicial = psutil.virtual_memory().percent
    tiempo_inicial = time.time()

    # Entrenamiento KNN
    KNNentrenamientoExitoso = DetectarCaracter.cargarDatosKNNyEntrenamientoKNN()

    if not KNNentrenamientoExitoso:
        print("\nError: KNN el entrenamiento no fue exitoso\n")
        return

    # Ruta de la imagen y placa real
    ruta_imagen = "E:/v2-ORIGINAL/mrp-pal titulo-no tocar/METODO DE RECONOCIMIENTO DE
    PLACAS DE MOTOTAXI/DPLACAS/DIA/mototaxi_h_dia (1).jpeg"
    placa_real = "3897ba" # Esta es la placa correcta para la imagen

    # Verificar que la ruta y la imagen existen
    if not os.path.exists(ruta_imagen):
        print(f"Error: La imagen no existe en la ruta especificada: {ruta_imagen}")
        return

    # Intentar cargar la imagen
    imgEscenaOriginal = cv2.imread(ruta_imagen)

    # Validar que la imagen se cargó correctamente
    if imgEscenaOriginal is None:
        print(f"\nError: La imagen no pudo ser leída en la ruta '{ruta_imagen}'\n")
        return
    else:
        print("La imagen se ha cargado correctamente y está lista para ser procesada.")

    # Detección de posibles placas
    listaPosiblePlacas = DetectarPlaca.detectarPlacaEnEscena(imgEscenaOriginal)

```

```

listaPosiblePlacas = DetectarCaracter.detectarCaracteresEnPlacas(listaPosiblePlacas)

cv2.imshow("Imagen Escena Original", imgEscenaOriginal)

# Mostrar el número de posibles placas encontradas
print(f"\n{len(listaPosiblePlacas)} posible(s) placa(s) encontrada(s)\n")

if len(listaPosiblePlacas) == 0:
    print("\nNo se detectaron placas de matrícula\n")
else:
    # Ordenar las placas según la longitud de los caracteres detectados
    listaPosiblePlacas.sort(key=lambda posiblePlaca: len(posiblePlaca.strChars),
reverse=True)
    licPlaca = listaPosiblePlacas[0] # Placa con mayor cantidad de caracteres
detectados

    cv2.imshow("imgPlaca", licPlaca.imgPlaca)
    cv2.imshow("imgUmbral", licPlaca.imgUmbral)

    # Evaluar si se detectaron los caracteres de la placa
    if len(licPlaca.strChars) == 0:
        print("\nNo se detectaron caracteres\n\n")
        return

    # Mostrar los caracteres detectados (número de placa)
    print(f"Número de placa detectado: {licPlaca.strChars}")

    # Dibujar rectángulo rojo alrededor de la placa
    dibujarRectanguloRojoEnPlaca(imgEscenaOriginal, licPlaca)

    # ---- Cálculo de métricas basado en comparación carácter por carácter ----
    caracteres_detectados = licPlaca.strChars.strip() # Eliminar espacios en blanco
    longitud_placa_real = len(placa_real)
    longitud_placa_detectada = len(caracteres_detectados)

    # Ajustar comparación para placas detectadas más largas
    if longitud_placa_detectada > longitud_placa_real:
        caracteres_detectados = caracteres_detectados[:longitud_placa_real] # Limitar
al tamaño de la placa real

    # Inicializar variables para el cálculo de las métricas
    VP = 0 # Verdaderos Positivos (caracteres correctos detectados)
    FP = 0 # Falsos Positivos (caracteres incorrectos detectados)
    FN = 0 # Falsos Negativos (caracteres no detectados)

    # Comparar carácter por carácter entre la placa detectada y la real
    for i in range(min(longitud_placa_real, longitud_placa_detectada)):
        if caracteres_detectados[i] == placa_real[i]:
            VP += 1
        else:
            FP += 1

    # Si la placa detectada tiene más caracteres que la real, esos son falsos positivos
    if longitud_placa_detectada > longitud_placa_real:
        FP += (longitud_placa_detectada - longitud_placa_real)

    # Si la placa real tiene más caracteres que los detectados, esos son falsos
negativos
    if longitud_placa_real > longitud_placa_detectada:
        FN += (longitud_placa_real - longitud_placa_detectada)

    # Calcular precisión, recall y exactitud
    precision = VP / (VP + FP) if (VP + FP) > 0 else 0
    recall = VP / (VP + FN) if (VP + FN) > 0 else 0
    exactitud = VP / longitud_placa_real if longitud_placa_real > 0 else 0

    # Mostrar métricas
    print(f"Precisión: {round(precision * 100, 2)}%")
    print(f"Recall: {round(recall * 100, 2)}%")
    print(f"Exactitud: {round(exactitud * 100, 2)}%")

    # Mostrar la imagen con la placa
    escribirCaracteresDeMatriculaEnLaImagen(imgEscenaOriginal, licPlaca)
    cv2.imshow("Imagen Escena Original", imgEscenaOriginal)
    cv2.imwrite("imgOriginalScene.png", imgEscenaOriginal)

# Medir consumo final de recursos

```

```

cpu_final = psutil.cpu_percent(interval=1)
memoria_final = psutil.virtual_memory().percent
tiempo_final = time.time()

consumo_cpu = round(cpu_final)
consumo_memoria = round(memoria_final)
tiempo_respuesta = round(tiempo_final - tiempo_inicial, 2)

# Mostrar métricas de rendimiento
print(f"Consumo de CPU: {consumo_cpu}%")
print(f"Consumo de Memoria: {consumo_memoria}%")
print(f"Tiempo promedio de respuesta: {tiempo_respuesta} segundos")

cv2.waitKey(0)
return

# Función para dibujar el rectángulo rojo en la placa
def dibujarRectanguloRojoEnPlaca(imgEscenaOriginal, licPlaca):
    p2fRectPoints = cv2.boxPoints(licPlaca.ubicacionDeLaPlacaEnLaEscena)

    cv2.line(imgEscenaOriginal, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]),
SCALAR_RED, 2)
    cv2.line(imgEscenaOriginal, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]),
SCALAR_RED, 2)
    cv2.line(imgEscenaOriginal, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]),
SCALAR_RED, 2)
    cv2.line(imgEscenaOriginal, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]),
SCALAR_RED, 2)

# Función para escribir los caracteres detectados en la imagen
def escribirCaracteresDeMatriculaEnLaImagen(imgEscenaOriginal, licPlaca):
    # Aquí va el código para escribir los caracteres sobre la imagen
    pass

if __name__ == "__main__":
    main()

```

Módulo: PosibleCaracter.py

Clase: PosibleCaracter

```

# PossibleChar.py

import cv2
import numpy as np
import math

#####
#####
class PosibleCaracter:

    # constructor
#####
    def __init__(self, _contour):
        self.contour = _contour

        self.boundingRect = cv2.boundingRect(self.contour)

        [intX, intY, intWidth, intHeight] = self.boundingRect

        self.intBoundingRectX = intX
        self.intBoundingRectY = intY
        self.intBoundingRectWidth = intWidth
        self.intBoundingRectHeight = intHeight

        self.intBoundingRectArea = self.intBoundingRectWidth * self.intBoundingRectHeight

        self.intCenterX = (self.intBoundingRectX + self.intBoundingRectX +
self.intBoundingRectWidth) / 2
        self.intCenterY = (self.intBoundingRectY + self.intBoundingRectY +
self.intBoundingRectHeight) / 2

        self.fltDiagonalSize = math.sqrt((self.intBoundingRectWidth ** 2) +
(self.intBoundingRectHeight ** 2))

```



```

        self.fltAspectRatio = float(self.intBoundingRectWidth) /
float(self.intBoundingRectHeight)
        # end constructor

# end class

```

Módulo: PosiblePlaca.py

Clase: PosiblePlaca

```

# PossiblePlate.py

import cv2
import numpy as np

#####
class PosiblePlaca:

    # constructor
#####
    def __init__(self):
        self.imgPlaca = None
        self.imgEscalaDeGrises = None
        self.imgUmbral = None

        self.ubicacionDeLaPlacaEnLaEscena = None

        self.strChars = ""
        # end constructor

# end class

```

Módulo: Preprocesamiento.py

Funciones y Técnicas:

```

# Preprocess.py

import cv2
import numpy as np
import math

# module level variables
#####
GAUSSIAN_SMOOTH_FILTER_SIZE = (5, 5)
ADAPTIVE_THRESH_BLOCK_SIZE = 19
ADAPTIVE_THRESH_WEIGHT = 9

#####
def preprocesamiento(imgOriginal):
    imgEscalaDeGrises = extraerValor(imgOriginal)

    imgMaxContrastGrayscale = maximizarContraste(imgEscalaDeGrises)

    height, width = imgEscalaDeGrises.shape

    imgBlurred = np.zeros((height, width, 1), np.uint8)

    imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale, GAUSSIAN_SMOOTH_FILTER_SIZE, 0)

    imgUmbral = cv2.adaptiveThreshold(imgBlurred, 255.0, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, ADAPTIVE_THRESH_BLOCK_SIZE, ADAPTIVE_THRESH_WEIGHT)

    return imgEscalaDeGrises, imgUmbral
# end function

#####
def extraerValor(imgOriginal):
    height, width, numChannels = imgOriginal.shape

    imgHSV = np.zeros((height, width, 3), np.uint8)

    imgHSV = cv2.cvtColor(imgOriginal, cv2.COLOR_BGR2HSV)

```

```

    imgHue, imgSaturation, imgValue = cv2.split(imgHSV)

    return imgValue
# end function

#####
#####
def maximizarContraste(imgEscalaDeGrises):

    height, width = imgEscalaDeGrises.shape

    imgTopHat = np.zeros((height, width, 1), np.uint8)
    imgBlackHat = np.zeros((height, width, 1), np.uint8)

    structuringElement = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

    imgTopHat = cv2.morphologyEx(imgEscalaDeGrises, cv2.MORPH_TOPHAT, structuringElement)
    imgBlackHat = cv2.morphologyEx(imgEscalaDeGrises, cv2.MORPH_BLACKHAT,
    structuringElement)

    imgEscalaDeGrisesPlusTopHat = cv2.add(imgEscalaDeGrises, imgTopHat)
    imgEscalaDeGrisesPlusTopHatMinusBlackHat = cv2.subtract(imgEscalaDeGrisesPlusTopHat,
    imgBlackHat)

    return imgEscalaDeGrisesPlusTopHatMinusBlackHat
# end function

```

Anexo 7. Ejecución del método a una placa de mototaxi.

```
In [2]: runfile('E:/v2-ORIGINAL/mrp-pal
titulo-no tocar/METODO DE RECONOCIMIENTO DE
PLACAS DE MOTOTAXI/Main.py', wdir='E:/v2-
ORIGINAL/mrp-pal titulo-no tocar/METODO DE
RECONOCIMIENTO DE PLACAS DE MOTOTAXI')
Reloaded modules: DetectarCaracter, Main,
DetectarPlaca, Preprocesamiento,
PosiblePlaca, PosibleCaracter
La imagen se ha cargado correctamente y esta
lista para ser procesada.

2 posible placa encontrada

2 posible(s) placa(s) encontrada(s)

Número de placa detectado: 3897BA
Precisión: 100.0%
Recall: 100.0%
Exactitud: 100.0%
Consumo de CPU: 2%
Consumo de Memoria: 64%
Tiempo promedio de respuesta: 1.37 segundos
```

Anexo 8. Evaluación de rendimiento de algoritmos, técnica y métodos.

Etapas	Método/ Algoritmo	Beneficios	Limitaciones	Precisión (%)	Exactitud (%)	Evaluación	Autores
Preprocesamiento	Gaussian Blur	Reduce ruido y suaviza la imagen	Puede perder características importantes	85	80	Esencial para reducción de ruido antes de la detección de bordes	[38]
	Median Filter	Eficaz para suprimir ruido	No adecuado para variaciones extremas	84.2	80.9	Reducción específica de ruido	[38]
	Mean	Suavizado general de imagen	Puede reducir detalles importantes	82.5	81	Mejora en detalles sin ruido	[38]
	Convolve	Mejora la claridad de la imagen	Puede introducir distorsiones si no se aplica correctamente	83.5	81.4	Claridad y mejora de detalles	[38]

	Mediam	Reduce el ruido de "sal y pimienta"	No adecuado para imágenes con variaciones extremas	84.2	80.9	Reducción específica de ruido	[38]
	Edge Detector	Alta precisión para detección de bordes	Sensible al ruido	97.8	96.2	Mejor precisión en detección de bordes	[11]
	Canny	Alta precisión en detección de bordes	Sensible al ruido y variaciones de iluminación	97.8	96.2	Mejor precisión en detección de bordes	[11]
Procesamiento	Sobel	Simplicidad y eficiencia en la detección de bordes	Menor precisión que Canny, menos efectivo en bordes complejos	94.5	92.8	Detección eficiente para estructuras simples	[11]
	Roberts Edge Detection	Rápido y sencillo para imágenes en escala de grises	Sensible al ruido, menor precisión en bordes curvos	91.7	90.5	Detección de bordes para detalles rápidos	[11]

Segmentación	ACO	Optimizador de detección rápida	Requiere ajuste preciso de parámetros	89.0	87.0	Optimización efectiva en tiempo de procesamiento	[39]
	OTSU	Umbralización adaptativa efectiva	Menor precisión en escenas con variaciones de iluminación	92.7	90.5	Separación clara de objetos en imagen	[40]
	ISODATA	Ajusta dinámicamente los umbrales	Requiere buena iluminación para resultados óptimos	91	89	Ajuste dinámico para umbralización	[40]
	MaxEntropy	Determina umbrales para separar objetos complejos	Menos efectivo en imágenes con bajos contrastes	90.2	88.5	Umbralización para objetos complejos	[40]
	Li	Método adaptativo de umbralización	Requiere imágenes claras para mejores resultados	91.2	90	Umbralización adaptativa robusta	[40]

Extracción de características	HOG	Captura características robustas de bordes	Requiere un preprocesamiento adecuado	96.1	95	Reconocimiento robusto y efectivo	[39]
	CCA	Eficaz para detectar objetos conectados	Limitado si hay ruido significativo	95.3	94.1	Detección de componentes conectados	[39]
	Hough Transform	Identificación precisa de líneas y curvas	Ineficaz si hay demasiados elementos dispersos	94	93.2	Detección precisa de formas geométricas	[39]
Clasificación	KNN	Simplicidad y eficiencia en clasificación	Sensible a ruido y variaciones menores en la entrada	86.3	85	Clasificación rápida para distancias cortas	[39]
	SVM	Alta precisión para datos complejos con márgenes claros	Requiere ajuste de parámetros, computacionalmente costoso	95.5	94	Alta precisión en ambientes variados	[39]

Multilayer Perceptron (MLP)	Eficaz en clasificación multiclase	Necesita tiempo de entrenamiento extenso	93.0	95.0	Excelente para clasificación con varias capas ocultas	
ANN	Capacidad para modelar datos no lineales	Necesita datos extensivos para entrenamiento	92.1	90.8	Eficiencia en clasificación no lineal	[39]
Random Forest	Buena precisión y manejo de datos no balanceados	Requiere ajustes y es computacionalmente intensivo	93.2	91.8	Clasificación robusta y precisa	[39]
Red Neuronal Artificial (ANN)	Modela datos no lineales, adaptabilidad	Requiere gran cantidad de datos de entrenamiento	90.0	83.0	Eficiencia en reconocimiento complejo	[17]

	AdaBoost	Combina múltiples clasificadores débiles para crear uno fuerte	Sensible a ruido y sesgos en datos	89.0	83.0	Mejora la precisión de clasificación a partir de modelos débiles	[17]
	Red Bayes	Clasificación probabilística basada en la teoría de Bayes	Requiere independencia de predictores, menos preciso con datos correlacionados	86.5	85.0	Eficiencia en datos limitados y probabilidad	Alsudani et al., 2023
	Naive Bayes	Simple, rápido, efectivo para problemas probabilísticos	Suposición de independencia puede no ser realista	84.7	83.0	Rápido para decisiones simples con datos claros	Nosseir et al., 2020
	OCR	Eficaz para caracteres claros	Problemas con ruido y fuentes complejas	83.5	82.1	Reconocimiento bajo condiciones controladas	Nosseir et al., 2020
Reconocimiento	MLP	Mejor para patrones de datos complejos	Requiere largo tiempo de entrenamiento	85.6	84.3	Reconocimiento mejorado para patrones variados	Nosseir et al., 2020

LPRN	Alto	Requiere una formación sólida	90.	85.2	Eficiente	[41]
------	------	-------------------------------	-----	------	-----------	------

Nota: Algoritmos selectos de artículos científicos. Fuente: Elaboración propia.

Anexo 9. Imagen sobre la detección de una placa de rodaje

The image shows the Spyder Python IDE interface. The editor window displays the following code:

```
11 variables del módulo
12 LAR_RED = (0.0, 0.0, 255.0)
13 trarPasos = False
14
15 def main():
16     # Medir el consumo inicial de recursos
17     cpu_inicial = psutil.cpu_percent(interval=1)
18     memoria_inicial = psutil.virtual_memory().percent
19     tiempo_ini
20
21 # Entrenam
22 KNNentrenam
23
24 if not KNN
25     print(
26     return
27
28 # Ruta de
29 ruta_image
30 placa_real
31
32 # Verifica
33 if not os.
34     print(
35     return
36
37 # Intentar
38 imgEscenaO
39
40 # Validar
41 if imgEsce
42     print(
43     return
44 else:
45     print(
46
47 # Detecció
48 listaDoch
```

The image window shows a photograph of a motorcycle with a license plate that has been detected and highlighted with a red bounding box. The license plate number is 3897-BA.

The terminal window shows the following output:

```
In [1]: runfile('E:/v2-ORIGINAL/mrp-pal
titulo-no tocar/METODO DE RECONOCIMIENTO DE
PLACAS DE MOTOTAXI/Main.py', wdir='E:/v2-
ORIGINAL/mrp-pal titulo-no tocar/METODO DE
RECONOCIMIENTO DE PLACAS DE MOTOTAXI')
La imagen se ha cargado correctamente y
está lista para ser procesada.

2 posible placa encontrada

2 posible(s) placa(s) encontrada(s)

Número de placa detectado: 3897BA
Precisión: 66.67%
Recall: 100.0%
Exactitud: 66.67%
Consumo de CPU: 13%
Consumo de Memoria: 76%
Tiempo promedio de respuesta: 1.35 segundos
```