



Instituto tecnológico de Culiacán

Sistema de detección de placas vehiculares

Materia:

Topicos de Inteligencia Artificial

Integrantes:

Ramirez Medina Cristian Andrea

Castro Figueroa Daniel Sebastian

Maestro:

Dr. Mora Felix Zuriel Dathan

Repositorio: <https://github.com/DanielSdc/TopicosIA/tree/main/Unidad4>

Índice

1. Introducción	3
2. Objetivo General	4
3. Objetivos Específicos	4
4. Descripción del Problema	5
5. Justificación	5
6. Documentación Técnica y manual de instalación	6
6.1. Backend API reconocimiento matriculas (ANPR)	6
6.1.1. Manual de Instalación	6
6.1.2. Especificaciones Técnicas	7
6.2. Backend JAVA Spring Boot (Lógica de negocio)	11
6.2.1. Manual de Instalación	11
6.2.2. Especificaciones Técnicas	12
6.3. Frontend App móvil	15
6.3.1. Manual de Instalación	15
6.3.2. Especificaciones Técnicas	16
7. Manual Usuario	18
7.1. Acceso al Sistema	18
7.2. Navegación principal	19
7.3. Reportar Incidencia	20
7.4. Gestión de perfil y vehículos	22

7.4.1. Vincular Cuenta de Propietario	22
7.4.2. Consultar historial incidencias	23
7.5. Problemas frecuentes	24

1. Introducción

El proyecto aborda el desarrollo de un sistema integral de detección de matrículas, el cual combina técnicas avanzadas de visión artificial, un sistema de gestión de bases de datos y un módulo de vinculación de información. Este sistema busca identificar de manera precisa las matrículas vehiculares a partir de imágenes y asociarlas con la información de sus propietarios registrada en una base de datos.

La tecnología de Reconocimiento Automático de Placas es una aplicación crucial de la visión por computador, esencial para la vigilancia y gestión vehicular. Históricamente, estos sistemas han evolucionado desde los primeros prototipos en el Reino Unido en 1979, hasta convertirse en herramientas capaces de escanear matrículas a altas velocidades y en diversas condiciones. El valor de estas herramientas radica en su capacidad para agilizar la gestión de acceso, mejorar la seguridad en zonas de parqueo, y brindar un apoyo invaluable en la vigilancia y la búsqueda de vehículos de interés[1].

2. Objetivo General

Implementar un sistema automatizado completo para el reconocimiento de matrículas vehiculares (ANPR) en tiempo real, con el fin de validar y asociar de forma precisa los identificadores de placa detectados con la información de los propietarios almacenada en una base de datos central para generar incidencias ó en caso contrario señalar los fallos al usuario.

3. Objetivos Específicos

1. Entrenar un modelo de visión artificial capaz de detectar matricula vehiculares con una precisión de al menos 80 %.
2. Desarrollar una API que integre el modelo detector de placas y un OCR que sea capaz de recibir imágenes y regrese la matrícula (si es que hay una) con tiempos de respuesta cortos.
3. Construir un modelo relacional de base de datos que permita almacenar usuarios, propietarios, vehículos e incidencias, asegurando integridad referencial y soporte para reglas de negocio como el bloqueo automático de tarjetas tras la acumulación de tres infracciones
4. Diseñar y configurar un backend en Java Spring Boot encargado de la lógica de negocio, gestión de usuarios, registro de incidencias y control de acceso, asegurando la correcta integración con la base de datos.
5. Desarrollar una aplicación móvil que permita a los usuarios capturar imágenes, enviar datos al servicio ANPR, registrar incidencias con geolocalización y consultar sus infracciones asociadas.

4. Descripción del Problema

este proyecto busca resolver es la ineficiencia y el descontrol en el acceso vehicular, lo que genera congestión y lentitud en el flujo de tráfico, detectar y notificar de manera inmediata infracciones vehiculares que generan una falta de cumplimiento normativo y una nula capacidad de alerta proactiva.

5. Justificación

El desarrollo de este Sistema Integral de Detección de Matrículas (ANPR) se justifica, no solo para ser un supervisor de cumplimiento que utiliza visión artificial y código abierto para identificar la falta y mandar una alerta directa, sino como una herramienta clave para agilizar la gestión de acceso y optimizar el flujo vehicular, logrando una solución segura y eficiente

6. Documentación Técnica y manual de instalación

En la siguiente sección se desarrollara el procedimiento que se debe llevar a cabo para la instalación correcta del sistema, cabe aclarar que al no contar con servidor dedicado para el modelo de ANPR y la logica del sistema, el proceso es el mismo tanto para el usuario final como para el técnico, y además que el sistema solo funciona en un ambiente local, aunque es posible desplegarlo si es que se desea y se cuenta con los recursos necesarios.

6.1. Backend API reconocimiento matriculas (ANPR)

Se detallan los pasos necesarios para desplegar el servicio de Inteligencia Artificial encargado de la detección y lectura de matrículas vehiculares, así como sus especificaciones técnicas.

6.1.1. Manual de Instalación

Requisitos: Antes de comenzar, asegúrese de que el entorno host tenga instalado el siguiente software:

- **Python:** Versión 3.12 o superior.
- **PIP:** Gestor de paquetes de Python.
- **Git:** Para clonar el repositorio.
- **(Opcional) Soporte GPU:** Drivers NVIDIA y CUDA Toolkit si se desea aceleración por hardware (aunque el sistema es totalmente funcional en CPU).

Configuración del Entorno: Paso 1: Obtener el Código Fuente

```
git clone <url_repositorio>
```

Paso 2: Instalación de Librerías El proyecto cuenta con un archivo requirements.txt que gestiona todas las dependencias de IA y servidor web. Se recomienda ejecutar este paso dentro de un entorno virtual (venv).

```
python -m venv venv
```

```
pip install -r requirements.txt
```

Nota: La instalación inicial descargará librerías pesadas como torch (PyTorch), ultralytics y paddlepaddle. Asegúrese de tener una conexión a internet estable.

Paso 3: Verificación de Modelos Asegúrese de que la carpeta model contenga el archivo best.pt. Este archivo contiene los pesos de la red neuronal entrenada. Si no existe, el sistema fallará al iniciar.

6.1.2. Especificaciones Técnicas

Esta sección describe la arquitectura interna del servicio para facilitar su mantenimiento y comprensión por parte del usuario técnico.

Arquitectura del Sistema

El proyecto implementa una arquitectura de microservicio RESTful utilizando FastAPI, diseñada para ser consumida por otros backends o aplicaciones móviles. El flujo de procesamiento sigue un patrón de "Pipeline" de visión por computadora:

1. Capa de API (app.py):

- Actúa como el controlador de entrada.
- Recibe la imagen en formato binario (multipart/form-data).
- Realiza la decodificación de bytes a matrices numéricas (NumPy/OpenCV).
- Gestiona las respuestas HTTP y el manejo de errores (400 Bad Request si la imagen es inválida).

2. Capa de Servicio (anpr_service.py):

- Contiene la lógica de inteligencia artificial.
- **Patrón Singleton:** Inicializa los modelos pesados solo una vez en el constructor `__init__` para optimizar el rendimiento.
- **Pipeline de Procesamiento:**
 - a) **Detección:** Utiliza YOLOv11 (best.pt) para inferir las coordenadas (Bounding Box) de la matrícula en la imagen completa.
 - b) **Pre-procesamiento:** Recorta la región de interés (ROI) de la imagen original basándose en la detección.
 - c) **Reconocimiento (OCR):** Pasa el recorte a PaddleOCR para extraer el texto.
 - d) **Heurística de Selección:** Aplica reglas de negocio (Expresiones Regulares) para filtrar falsos positivos y seleccionar el texto que cumple con el formato de una matrícula (prioridad a cadenas alfanuméricas de 4-10 caracteres).

Tecnologías y Modelos de IA

- **Framework WEB:** FastAPI (Python) sobre servidor Uvicorn.
- **Detección de objetos:** El núcleo del sistema de detección se basa en un modelo YOLOv11 Nano (`yolo11n.pt`), el cual fue re-entrenado específicamente para la tarea de localización de matrículas.

Entorno de Entrenamiento: El proceso se llevó a cabo en la plataforma Google Colab, aprovechando la aceleración por hardware mediante una GPU NVIDIA T4. Esto permitió reducir significativamente los tiempos de cómputo requeridos para el ajuste de pesos de la red neuronal.

Dataset Utilizado: Se empleó un conjunto de datos público de Roboflow Universe ("License Plate Recognition"), el cual consta de **10,125 imágenes** etiquetadas, distribuidas de la siguiente manera para garantizar una evaluación robusta:

- **Entrenamiento (Train):** 70 % de las imágenes.
- **Pruebas (Test):** 20 % de las imágenes.
- **Validación (Val):** 10 % de las imágenes.

Configuración del Entrenamiento: Para el ajuste fino (fine-tuning) del modelo, se establecieron los siguientes hiperparámetros:

- **Épocas:** 50 (Iteraciones completas sobre el dataset).
- **Tamaño de Lote (Batch Size):** 16 imágenes por iteración.
- **Resolución de Imagen:** 640x640 píxeles (estándar para YOLO).

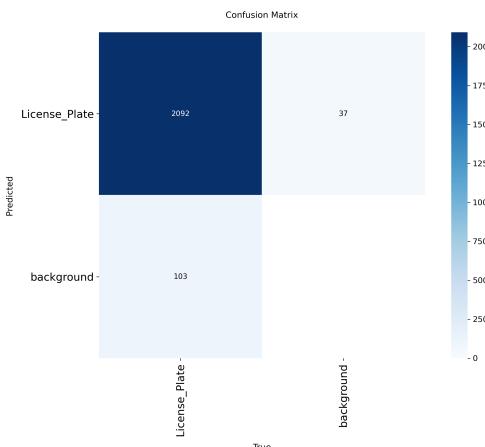


Figura 1: Matriz de confusión modelo

El resultado de este proceso es el archivo de pesos `best.pt`, el cual selecciona automáticamente la configuración que obtuvo las mejores métricas de precisión (mAP) durante las 50 épocas, y con una precisión del %98.67, garantizando así el rendimiento óptimo en inferencia.

- Input: Imagen completa.
- Output: Coordenadas [x1, y1, x2, y2] y confianza de detección.
- **OCR:** PaddleOCR.
 - Input: Recorte de imagen (Crop).
 - Output: Texto crudo y confianza de lectura.

Documentación de API (Endpoints) La API sigue el estándar RESTful y genera documentación automática interactiva bajo el estándar OpenAPI (Swagger UI).

- URL de Documentación: <http://localhost:8000/docs>
- Formato de Intercambio: JSON.

Endpoint Principal: POST /read_plate

- Descripción: Procesa una imagen subida y retorna la matrícula detectada.
- Body: form-data con campo file (archivo de imagen).
- Respuesta Exitosa (200 OK):

```
{  
    "plate_text": "HA9354A"  
}
```

- Respuesta Sin Detección (200 OK):

```
{  
    "plate_text": null,  
    "message": "No plate detected",  
    "candidates": []  
}
```

Heurística para obtener matrícula Para mitigar errores comunes en sistemas de visión por computadora, el servicio implementa la siguiente lógica de selección de candidatos:

- **Limpieza:** Todo texto detectado se convierte a mayúsculas y se eliminan caracteres especiales (solo A-Z y 0-9).
- **Prioridad 1 (Ideal):** Se busca un candidato que contenga tanto letras como números (formato estándar de matrícula).
- **Prioridad 2 (Respaldo):** Si no existe el anterior, se busca un candidato que contenga al menos dígitos (para evitar leer nombres de estados como "SINALOA").
- **Selección Final:** En caso de múltiples candidatos dentro de la misma prioridad, se elige el que tenga la mayor puntuación de confianza (Score) otorgada por el motor OCR

6.2. Backend JAVA Spring Boot (Lógica de negocio)

Este apartado detalla los pasos necesarios para desplegar el backend de la aplicación en un entorno local o servidor.

6.2.1. Manual de Instalación

Requisitos: Antes de comenzar, asegúrese de que el entorno host tenga instalado el siguiente software:

1. **Java Development Kit (JDK):** Versión 21 o superior (LTS recomendado).
2. **Maven:** Herramienta de construcción.
3. **Git:** Para clonar el repositorio.
4. **Base de Datos:** MySQL 8.0+.
5. **Cuenta de Firebase:** Proyecto activo en Google Firebase para la gestión de autenticación.

Configuración del entorno

Paso 1: Obtener el Código Fuente

```
git clone <url_repositorio>
```

Paso 2: Configuración de Credenciales (Firebase) El sistema requiere una clave de servicio para validar tokens de usuario.

1. Acceda a la Consola de Firebase >Configuración del Proyecto >Cuentas de servicio.
2. Genere una nueva clave privada (archivo .json).
3. Renombre el archivo descargado a firebase_service.json.
4. Mueva este archivo a la ruta: firebase_service.json.

Paso 3:Configuración de la Base de Datos Edite el archivo application.properties: Configuración de MySQL (Producción/Desarrollo):

```
spring.datasource.url=jdbc:mysql://localhost:3306/incidencias_db?createDatabaseIfNotExist=true&useSSL=false&serverTimezone=UTC
```

```
spring.datasource.username=tu_usuario o root
```

```
spring.datasource.password=tu_contraseña
```

```
spring.jpa.hibernate.ddl-auto=update
```

Ejecución directa con Maven Wrapper

Este comando descargará las dependencias y levantará el servidor embebido Tomcat en el puerto 8080:

```
mvnw spring-boot:run
```

6.2.2. Especificaciones Técnicas

Esta sección describe la arquitectura interna para facilitar el mantenimiento y escalabilidad futura por parte del equipo de desarrollo.

Arquitectura del Sistema

El proyecto implementa una Arquitectura RESTful basada en el patrón MVC (Model-View-Controller) de Spring Boot 3.5.7, organizada en capas lógicas (Controlador, Servicio, Repositorio) para garantizar la separación de responsabilidadesCapa de Controladores (/controller): Expone los endpoints REST. Maneja las peticiones HTTP y delega la lógica a los servicios.

- **Capa de Servicio (/service):** Contiene la lógica de negocio pura.
- **Capa de Persistencia (/repository):** Interfaces que extienden de JpaRepository para la comunicación con la base de datos mediante Hibernate.
- **Capa de Seguridad (/security):** Filtro personalizado JwtAuthenticationFilter que intercepta cada petición para validar el token Bearer contra Firebase Auth.

Esquema de Base de Datos (Entidad-Relación)

El sistema utiliza un el siguiente modelo relacional:

1. Usuarios (usuarios):

- o Almacena credenciales básicas.
- o Relación 1:1 con Propietarios.

2. Propietarios (propietarios):

- o Contiene datos personales y el estado de acceso (tarjetaPase: boolean).
- o Relación 1:N con Automóviles.

3. Automóviles (automoviles):

- o Identificados por matricula.
- o Relación 1:N con Incidencias.

4. Incidencias (incidencias):

- o Registro histórico de multas con latitud, longitud y fecha.

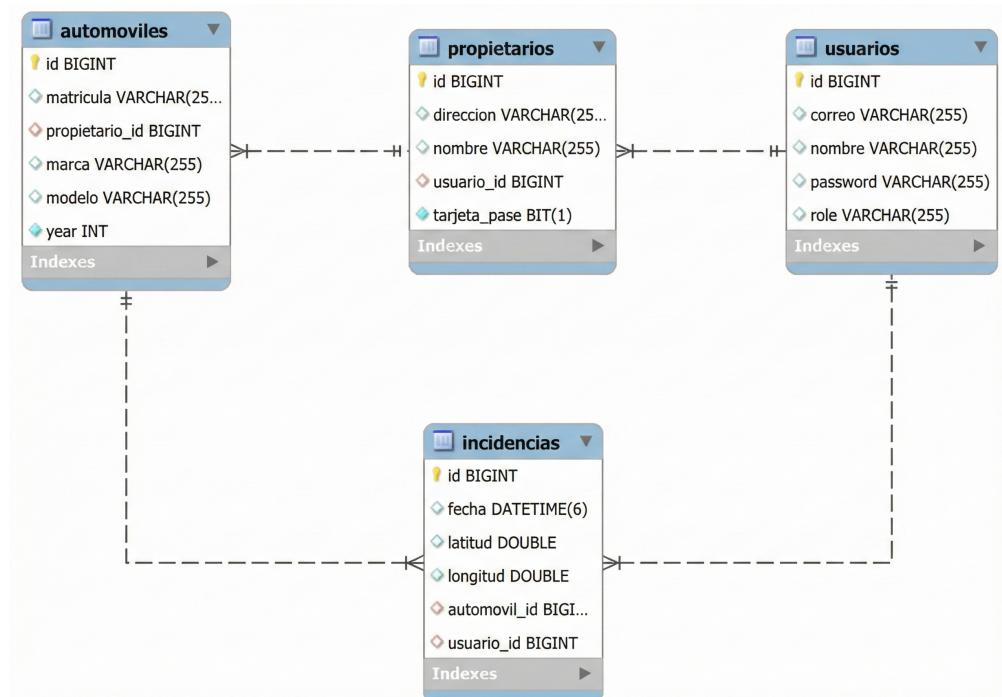


Figura 2: Modelo Base de datos

Documentación de API (Endpoints)

La API sigue el estándar RESTful y está documentada automáticamente con OpenAPI 3.0 (Swagger).

- **URL de Documentación:** <http://localhost:8080/swagger-ui/index.html>
- **Seguridad:** Esquema Bearer Auth (JWT).

Endpoints Principales:

- POST /api/incidencias: Registra una nueva multa. Dispara el proceso de verificación de bloqueo de tarjeta (con 3 incidencias se bloquea).
- GET /api/usuarios/perfil: Retorna la información del usuario y el estado actual de su tarjeta de pase (Habilitada/Deshabilitada/Perfil de propietario sin vincular).
- POST /api/propietarios/id/vincular: Vincula un usuario de la App con un registro de propietario existente.

Reglas de Negocio Críticas

El sistema implementa una lógica reactiva en el servicio de incidencias, al registrar una incidencia (IncidenciaService), el sistema consulta el historial del propietario asociado. Si el conteo total de incidencias es mayor o igual a 3, se invoca PropietarioService.deshabilitarTarjeta(id), bloqueando el acceso físico inmediatamente, el cual solo puede volver a ser habilitado manualmente en la base de datos.

6.3. Frontend App móvil

Este apartado detalla los pasos necesarios para desplegar, configurar y ejecutar la aplicación móvil encargada de la captura de evidencia y gestión de incidencias por parte del usuario final.

6.3.1. Manual de Instalación

Requisitos: Antes de comenzar, asegúrese de que el entorno de desarrollo tenga instalado lo siguiente:

- **Node.js:** Versión 20 LTS o superior (Requerido para NPM).
- **Expo Go:** Aplicación instalada en el dispositivo físico (Android/iOS) para pruebas en tiempo real.
- **Red Local:** El dispositivo móvil y el servidor donde corren los backends deben estar conectados a la misma red WiFi (si es que se encuentran de forma local).

Configuración del entorno

Paso 1: Obtener el código Fuente

```
git clone <url_repositorio>
```

Paso 2: Instalación de Librerías npm install Esto descargará librerías como expo-camera, expo-location, firebase y axios.

Configuración de conectividad: Dado que la aplicación móvil consume servicios locales, es necesario configurar las direcciones IP de los servidores.

Archivo de Configuración de API: Edite el archivo src/config/api.js y reemplace '192.168.1.XX' con la IP Local de su computadora anfitriona

Configuración de Firebase: Asegúrese de que el archivo src/config.firebaseio.js contenga las credenciales (API Key) correspondientes a su proyecto de Firebase Console para permitir la autenticación.

Ejecución del sistema

Para iniciar el servidor de desarrollo: npx expo start –clear

1. Aparecerá un código QR en la terminal.
2. Escanee este código utilizando la cámara (iOS) o la app Expo Go (Android).
3. La aplicación se compilará y ejecutará en su dispositivo.

6.3.2. Especificaciones Técnicas

Esta sección describe la arquitectura del frontend y cómo orquesta la comunicación entre los servicios de IA y de Negocio.

Arquitectura del Sistema

La aplicación móvil está construida sobre el framework React Native utilizando Expo. Funciona como un cliente que interactúa con tres sistemas externos:

1. **Google Firebase:** Para delegar la autenticación y obtención de Tokens de Seguridad.
2. **API Python:** Para el procesamiento de imágenes (ANPR) en tiempo real.
3. **Backend Java:** Para la lógica y persistencia de datos.

Tecnologías y Librerías Clave

- **Core:** React Native / Expo SDK 50+.
- **Navegación:** React Navigation (Stack) para el manejo de pantallas.
- **Hardware:**
 - expo-camera: Acceso a la cámara trasera para captura de evidencia.
 - expo-location: Acceso al GPS para geo-referenciar la incidencia.
 - expo-secure-store: Almacenamiento encriptado local para persistir el Token JWT de sesión.
- **Red:** Axios para peticiones HTTP (REST).

Estructura del proyecto

El código sigue una arquitectura modular basada en pantallas y servicios:

- **/src/screens:**

- LoginScreen.js: Maneja el inicio de sesión con Firebase y la sincronización automática del usuario con el Backend Java.
- HomeScreen.js: Dashboard principal que muestra el estado de la tarjeta de pase y menú de navegación.
- CameraScreen.js: Captura la imagen, la optimiza (compresión JPG 0.7) y la envía al servicio de Python.
- ConfirmScreen.js: Presenta los resultados del OCR, permite corrección manual y envía la incidencia final a Java.
- MisIncidenciasScreen.js: Historial de reportes enviados y recibidos.

- **/src/config:** Configuraciones centralizadas de API y Firebase.

Integración API's

La aplicación móvil consume los siguientes endpoints documentados anteriormente en los manuales de backend:

Servicio	Método	Endpoint	Propósito
Python	POST	/read_plate	Envía imagen binaria (multipart/form-data), recibe texto JSON.
Java	POST	/api/incidencias	Envía JSON con matrícula, latitud, longitud y Token Bearer.
Java	GET	/api/incidencias/enviadas	Obtener incidencias creadas por el usuario.
Java	GET	/api/incidencias/recibidas	Obtener incidencias asociadas a automóviles del usuario.
Java	GET	/api/automoviles/mis-autos	Obtener vehículos asociados al usuario logeado.
Java	POST	/api/propietarios/id/vincular	Vincular perfil usuario con un propietario existente.
Java	POST	/api/usuarios/registrar	Sincroniza el UID de Firebase con la base de datos PostgreSQL.
Java	GET	/api/usuarios/perfil	Mostrar datos del usuario logeado.

Cuadro 1: Servicios y Endpoints del sistema

7. Manual Usuario

7.1. Acceso al Sistema

Al abrir la aplicación por primera vez, encontrará la pantalla de autenticación.

Pasos para ingresar:

1. **Si ya tiene cuenta:** Ingrese su correo electrónico y contraseña en los campos correspondientes y presione el botón azul ".Entrar".
2. **Si es nuevo usuario:**
 - Toque el enlace ".¿No tienes cuenta? Regístrate." en la parte inferior.
 - Ingrese su Nombre Completo, Correo y una Contraseña segura (mínimo 6 caracteres).
 - Presione Registrarse".



Figura 3: Ventana Login

7.2. Navegación principal

Una vez haya iniciado sesión, verá el Panel de Control (Dashboard). Este contiene los siguientes elementos:

- **Tarjeta de Estado (Superior):** Le indica si su perfil tiene permisos de acceso (Tarjeta Habilitada/Deshabilitada) o si necesita vincular su cuenta de propietario.
- **Menú de Acciones:**
 - **Reportar Incidencia:** Para escanear un vehículo infractor.
 - **Mis Vehículos:** Para ver los autos registrados a su nombre..
 - **Infracciones Recibidas:** Para ver el historial de reportes.
- **Cerrar Sesión:** Botón inferior para salir de la cuenta de forma segura.



Figura 4: Menú Principal

7.3. Reportar Incidencia

Esta es la función principal del sistema. Siga estos pasos para registrar una infracción:

Paso 1: Escaneo de Matrícula

1. En el menú principal, toque el botón "Reportar Incidencia".
2. Se activará la cámara. Si es la primera vez, otorgue el permiso a la aplicación.
3. Apunte a la matrícula del vehículo.
4. Presione el botón circular para capturar (la imagen sera procesada por el modelo, lo cual puede tomar un par de segundos).

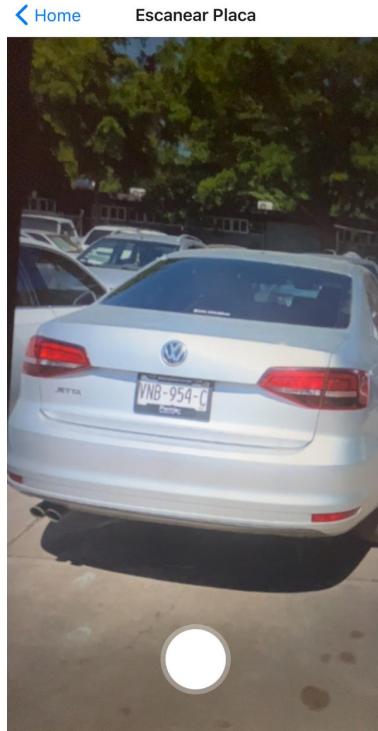


Figura 5: Ventana para capturar una incidencia

Paso 2: Validación y Envío

1. Si la lectura fue exitosa, verá la pantalla de Confirmación.
2. Verifique la Matrícula: El recuadro de texto mostrará lo que el sistema leyó (Ej: ABC123). Si hay un error, puedes volver a tomar una nueva imagen bajo mejores circunstancias.
3. Verifique la Ubicación: El sistema mostrará sus coordenadas GPS actuales. Si es la primera vez, otorgue permiso a la aplicación.
4. Presione el botón rojo **Registrar Multa**. (Si el vehículo no existe en la base de datos, recibirá un aviso indicando que no se pudo procesar ya que el vehículo no se encuentra registrado.)



Figura 6: Ventana para registrar una incidencia

7.4. Gestión de perfil y vehículos

7.4.1. Vincular Cuenta de Propietario

Si al entrar ve una tarjeta amarilla que dice "Tarjeta de pase no vinculada":

1. Toque la tarjeta amarilla o vaya a "Mis Vehículos".
2. Verá un formulario pidiendo su ID de Propietario.
3. Ingrese el número de ID de propietario (idealmente este debería de ser proporcionado por el área encargada de las tarjetas) y presione Confirmar.
4. Si es correcto, la tarjeta cambiará a color verde o rojo según su estado.



Figura 7: Ventana para vincular el perfil de usuario con un propietario

7.4.2. Consultar historial incidencias

Para ver las infracciones que han recibido o creado:

1. Toque el botón **Infracciones Recibidas** en el menú principal.
2. Use las pestañas superiores para alternar entre:
 - Recibidas (Rojo): Multas aplicadas a sus vehículos.
 - Enviadas (Azul): Reportes que usted ha generado.
3. Deslice hacia abajo para actualizar la lista.

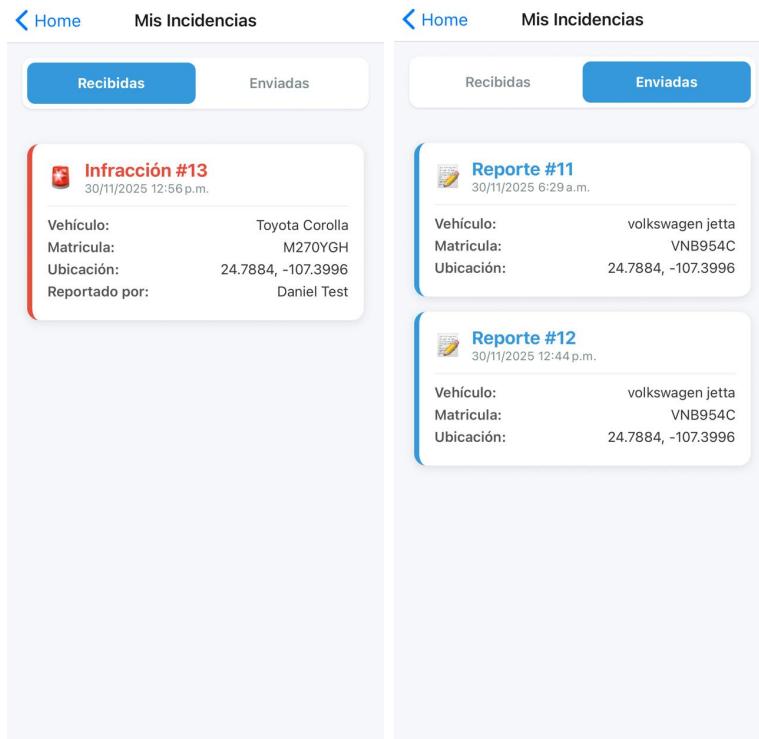


Figura 8: Infracciones recibidas y creadas

7.5. Problemas frecuentes

Problema	Causa	Solución
No se detectó ninguna matrícula	El dispositivo no tiene internet o no alcanza al servidor.	Intente tomar la foto de nuevo acercándose más y asegurando buena iluminación.
Error de conexión	El dispositivo no tiene internet o no alcanza al servidor.	Verifique su conexión Wi-Fi/Datos.
Vehículo no encontrado	La matrícula leída no está registrada en el padrón vehicular.	Verifique que la matrícula sea correcta. Si lo es, el auto no está en el sistema y no se puede multar.
La App se cierra al abrir la cámara	Permisos denegados.	Vaya a Configuración del Teléfono >Aplicaciones >Tránsito App >Permisos y active la Cámara.

Cuadro 2: Problemas frecuentes y su respectiva solución

Referencias

- [1] J. A. Vásquez and J. A. Melo Morales. Sistema automático de reconocimiento de placas vehiculares. <https://repository.ucc.edu.co/server/api/core/bitstreams/e8644af8-354c-4f98-b3e4-2ee17508fdae/content>, noviembre 2018.