UNIVERSIDAD NACIONAL DEL COMAHUE FACULTAD DE INGENIERÍA
DEPARTAMENTO DE ELETROTÉCNIA

DISEÑO E IMPLEMENTACIÓN DEL PROTOTIPO DE UN SISTEMA DE ESTACIONAMIENTO INTELIGENTE

Martin, Santiago Emanuel
Saez, Lautaro Andres

Ante la Facultad de Ingeniería de la Universidad Nacional del Comahue para acceder al título de:
INGENIERO ELECTRÓNICO

Direción
Director: Ing. Mendieta, Dario F.
Codirector: Dr. Moreyra, Marcelo L.

Neuquén, 29 de octubre de 2023

Agradecimientos

Queremos agradecer a nuestras familias, especialmente a nuestros padres, ya que sin su apoyo incondicional a lo largo de estos años no hubiésemos podido culminar esta etapa.

A nuestros compañeros, por formar parte de nuestro camino en la facultad de comienzo a fin.

A Dario Mendieta y Marcelo Moreyra por acompañarnos durante la carrera como en este último tramo, sus conocimientos y aportándonos su confianza para que sigamos creciendo como profesionales.

Resumen

A lo largo de este trabajo se detalla el procedimiento utilizado para desarrollar un prototipo de estacionamiento inteligente, basado en algoritmos de OCR mediante redes neuronales.

Para el desarrollo del algoritmo de OCR se utilizaron dos redes neuronales: la principal procesa una imagen de una patente y estima los caracteres, mientras que la secundaria es un detector clasificador que procesa la fotografía y entrega la imagen de la patente.

El prototipo contempla un servidor web y un sistema embebido que se instala en las entradas/salidas de un establecimiento. El sistema embebido cuenta con un sensor ultrasónico y una cámara USB. Con la finalidad de analizar costos se propusieron dos prototipos: una versión que es capaz de procesar las imágenes de forma local, basada en NVIDIA Jetson TX1, denominado como SL, y una versión simplificada que captura la imagen y la envía al servidor para posteriormente ser procesada en la Raspberry Pi 3B+, llamado SL mini.

El servidor web cuenta con una arquitectura de microservicios y fue diseñado para que sea fácilmente escalable. Este permite almacenar la información que recolectan los sistemas SL, adicionalmente los administradores pueden acceder a la información, y además pueden configurar los sistemas SL desde la web.

Finalmente, ya con el prototipo funcional, se realizaron ensayos a las distintas partes del sistema, con la finalidad de obtener su capacidad actual de procesamiento y sus limitaciones.

Palabres Clave

OCR, redes neuronales, sistemas autónomos, detección de patentes, Javascript.

Abstract

This work details the procedure used to develop a smart parking prototype throughout this thesis, based on OCR algorithms using neural networks.

For the OCR algorithm development, two neural networks were employed: the main network processes an image of a license plate and estimates the characters, while the secondary one is a classifier detector that processes the photograph and provides the license plate image.

The prototype includes a web server and an embedded system that is installed at the entrances/exits of a facility. The embedded system features an ultrasonic sensor and a USB camera. In order to analyze costs, two prototypes were proposed: one version capable of processing images locally based on NVIDIA Jetson TX1, referred to as SL, and a simplified version that captures the image and sends it to the server for later processing based on Raspberry Pi 3B+, called SL mini.

The web server employs a microservices architecture and was designed to be easily scalable. It allows storing the information collected by the SL systems, and administrators can access this information and configure the SL systems through the web.

Finally, with the functional prototype in place, tests were conducted on different parts of the system to determine its current processing capacity and limitations.

Keywords

OCR, neuronal networks, autonomous systems, license plate detection, Javascript.Autonomous systems, license plate detection.

Índice general

Agradecimientos I

Resumen II

Abstract III

1. Introducción y objetivos **2**

1.1. Fundamentos 2

1.2. Objetivos 3

1.2.1. Objetivo general 3

1.2.2. Objetivos específicos 3

1.3. Metodología 4

1.4. Estructura de la tesis 5

2. Diseño del sistema **7**

2.1. Descripción general del sistema 7

2.2. Algoritmos de entrada y salida 8

2.2.1. Algoritmo de entrada 8

2.2.2. Algoritmo de salida 9

2.2.3. El problema de múltiples establecimientos 10

2.3. Partes del sistema 10

2.3.1. Interacción con el cliente 11

3. Detección óptica de caracteres **12**

3.1. Algoritmos de OCR 12

3.1.1. Redes LSTM . . . 13
3.1.2. Redes CNN . . . 14
3.2. Definición de red neuronal . . . 15
3.2.1. La neurona . . . 15
3.3. Redes neuronales . . . 16
3.4. Redes neuronales de convolución . . . 17
3.4.1. Convolución en 2D . . . 19
3.4.2. Capa de agrupación . . . 19
3.4.3. Técnica de Batch-Normalization . . . 20
3.5. CCN-OCR . . . 20
3.6. Requerimientos . . . 22
3.6.1. Obtención del recorte de la patente . . . 23
3.6.2. Transformación a escala de grises . . . 24
3.6.3. Redimensionamiento de la imagen . . . 24
3.7. Implementación del algoritmo en Python 3 . . . 25
3.7.1. Prueba de rendimiento . . . 26

4. Diseño e implementación de los sistemas SL y SL mini 27

4.1. Análisis de requisitos . . . 27
4.1.1. Sistemas actuales . . . 27
4.1.1.1. Sistemas tradicionales de tickets . . . 28
4.1.1.2. Sistemas de radiofrecuencia . . . 28
4.1.2. Requisitos de los sistemas SL . . . 29
4.2. Selección de placas . . . 30
4.2.1. Características del SL mini . . . 30
4.2.2. Características del SL . . . 31
4.3. Evaluación y selección de sensores . . . 32
4.3.1. Selección de cámara . . . 33
4.3.2. Selección de activador . . . 33
4.3.2.1. Barrera infrarroja . . . 33
4.3.2.2. Placa de presión . . . 34

4.3.2.3. Sensor de distancia . . . 34
4.4. Diseño y ensamble . . . 35
4.5. Implementación de los drivers . . . 35
4.5.1. Lectura de distancia . . . 35
4.5.2. Utilidades . . . 36
4.5.3. Diferencias entre SL y SL mini . . . 37

5. Diseño e implementación del servidor 38
5.1. Actualidad del desarrollo web . . . 39
5.2. Docker . . . 39
5.2.1. Contenedores . . . 40
5.2.2. Docker Compose . . . 40
5.2.2.1. version . . . 40
5.2.2.2. services . . . 40
5.2.2.3. intranet . . . 42
5.3. Protocolos de comunicación . . . 42
5.3.1. HTTP . . . 42
5.3.2. MQTT . . . 43
5.4. Servicios . . . 43
5.4.1. Base de datos . . . 43
5.4.2. Broker MQTT . . . 43
5.4.3. Back-end . . . 44
5.4.3.1. Api Key . . . 45
5.4.3.2. JWT . . . 46
5.4.4. Análisis de imágenes . . . 46
5.4.5. Front-end . . . 46
5.4.5.1. Páginas para los usuarios . . . 46
5.4.5.2. Páginas para los administradores . . . 49
5.4.6. Nginx . . . 49

6. Pruebas de integración y fiabilidad 50
6.1. Pruebas sobre el algoritmo de OCR . . . 50

6.1.1. Pruebas de distancia y ángulo 50
6.1.2. Pruebas de exterior 52
6.1.3. Análisis de resultados 53
6.2. Pruebas sobre el servidor 55
6.2.1. Integración de los sistemas SL y servidor 55
6.2.2. Creación de registros 55
6.3. Estado del prototipo 56

7. Conclusiones **57**
7.1. Conclusiones generales 57
7.2. Posibles mejoras 58

Bibliografía **60**

Índice de figuras

2.1. Modelo de caja negra del sistema. 8
2.2. Sistema con sensor de distancia y cámara. 8
2.3. Diagramas de flujo para la entrada y la salida de un vehículo del estacionamiento. 9
2.4. Sistema separado en SL y servidor. 10
2.5. Esquema de comunicación Cliente-Servidor-SL. 11

3.1. Diagrama del bloque LSTM. 14
3.2. Evolución patentes Argentinas a lo largo de los años. 15
3.3. Comparación neurona con neurona artificial. 15
3.4. Esquema simplificado de una red neuronal, donde los círculos representan una neurona junto con la función de activación. 16
3.5. Funciones de activación más utlizadas. 17
3.6. Ejemplificación de una red neuronal de convolución. 18
3.7. Esquema de funcionamiento de una capa convolucional 2D [10]. . . . 19
3.8. Ejemplo de funcionamiento de Maxpooling. 20
3.9. Arquitectura de la CNN-OCR de 1.3 millones de parámetros optimizados para CPU. 21
3.10. Comparación entre una capa de Flatten y una de Global Pooling. . . 21
3.11. Ejemplo de imagen obtenida e imagen requerida por la CNN-OCR. . 22
3.12. Esquema general de un detector de una y dos etapas. [11]. 24
3.13. En (a) se observa la imagen original, mientras que en (b) y (c) la imagen convertida a escala de grises por los diferentes métodos. . . . 24
3.14. Algoritmo general de OCR. 25

3.15. Flujo de una fotografía hasta obtener los caracteres de la patente. . . 25

4.1. Sistema tradicional de acceso por barreras [15]. 28
4.2. Sistema de acceso por RFID [16]. 29
4.3. Raspberry Pi 3B+. 31
4.4. NVIDIA Jetson TX1 development kit. 32
4.5. Cámara utilizada. 33
4.6. Sensor de distancia ultrasónico US-100. 34
4.7. Vistas del modelo 3D del contendor. 36
4.8. Contenedor ensamblado y montado sobre un trípode. 37

5.1. Esquema final del servidor. 38
5.2. Ejemplificación de un JSON. 39
5.3. Ejemplificación de una archivo *docker-compose.yml* 41
5.4. Esquema de la base de datos. 44
5.5. Documentación generada por Swagger. 45
5.6. Inicio de sesión. 47
5.7. Estacionamientos del cliente. 47
5.8. Formulario para actualizar un estacionamiento. 47
5.9. Vista de permisos. 47
5.10. Vista de los detalles. 48
5.11. Formulario para actualizar los datos del sistema SL. 48
5.12. Visualización de datos para un cliente con 2 estacionamientos. 48
5.13. Vista de registros. 48
5.14. Detalles de los sistemas SL según cliente y estacionamiento. 49
5.15. Formulario para agregar un sistema SL. 49

6.1. Sistema de referencia. 51
6.2. Fotografías a diferentes distancias a $0°$. 52
6.3. Fotografías a diferentes ángulos a $50cm$. 52
6.4. Ejemplificación de las imágenes obtenidas durante la prueba de exterior. 54

6.5. Vistas del SL mini y servidor junto con el vehículo de durante las pruebas de integración. 55
6.6. Prototipos en su estado actual. 56

Índice de tablas

3.1. Tiempos obtenidos del procesamiento de diez imágenes. 26

4.1. Resumen de características de las placas embebidas. 32

6.1. Resumen de la prueba de distancia a 0° grados. 51
6.2. Resumen de la prueba de ángulos a 50 cm. 51
6.3. Resumen de las patentes reconocidas. 53

Capítulo 1

Introducción y objetivos

1.1. Fundamentos

La creciente flota de vehículos circulantes a nivel mundial y principalmente en Argentina que ha aumentado un 1,88 % entre 2020 y 2021 [1], logrando un total de 14.840.010 a finales del 2021 lo que trae aparejado un problema para encontrar lugar de estacionamiento. Particularmente en la ciudad de Neuquén, esta problemática se ve amplificada por la constante llegada de vehículos de ciudades aledañas, principalmente por razones de estudio o trabajo, llegando casi a duplicar la flota de vehículos circulantes en el día a día [2]. Este problema no es único de esta ciudad, a lo largo de todo el mundo ya se han encontrado situaciones similares [3, 4].

Gracias a los avances en diversas tecnologías tales como: robótica, simulación, sistemas de integración, IOT (Internet of things), IA (Inteligencia artificial), Ciberseguridad, Big data, entre otras, la industria llegó a una nueva era: Industria 4.0, donde la tecnología es cada vez más utilizada para realización de tareas y resolución de problemas buscando una industria autónoma [5].

Muchas de estas ideas de industrias inteligentes, comenzaron a ser extrapoladas en otros aspectos de la vida cotidiana, dando paso a la idea de ciudades inteligentes. Esto se debe principalmente al ahorro de recursos que pueden brindar. Dentro de los servicios fundamentales que debe afrontar una ciudad inteligente están: eficiencia energética y medioambiente, gestión de infraestructuras y edificios públicos, seguridad pública y movilidad urbana. Es en esta última donde se busca generar un aporte

y una base para futuros trabajos. La idea general es integrar nuevas tecnologías a una idea antigua, es decir, modificar el sistema de control de estacionamiento clásico, buscando la automatización. Este avance presenta mejoras tanto para los usuarios del sistema, que desean estacionar y pueden hacerlo de una forma más eficiente, gracias a la eliminación de intermediarios. Por otro lado, favorece al dueño o administrador, ya que le permite llevar un control de rendimiento mucho más sencillo, planificar de una mejor manera sus inversiones, dando datos estadísticos de uso reales de sus usuarios.

La idea de un estacionamiento inteligente, está siendo explorada desde otros enfoques en diversos países a lo largo del mundo, por ejemplo, control de espacios por sensores de ultrasonido [6], plataformas inteligentes de estacionamiento [7], entre otros. El enfoque de este trabajo se centrará en la visión artificial, mediante la detección y reconocimiento de caracteres de imágenes tomadas desde una cámara. Se buscará reconocer y almacenar la patente de cada vehículo que ingresa y egresa de un espacio.

1.2. Objetivos

1.2.1. Objetivo general

Diseñar e implementar un sistema de estacionamiento inteligente mediante el reconocimiento de caracteres de las patentes a través de visión artificial, con la finalidad de llevar un seguimiento de los vehículos que ingresan y egresan del recinto, y un control de los tiempos de entrada y salida.

1.2.2. Objetivos específicos

1. Desarrollo y evaluación de los algoritmos necesarios para el reconocimiento de patentes basados en algoritmos de OCR (reconocimiento óptico de caracteres).

2. Implementación de los algoritmos desarrollados en el hardware embebido (Raspberry Pi y NVIDIA Jetson TX1) para su funcionamiento en tiempo real.

3. Evaluación de sensores a utilizar.

4. Desarrollo de una aplicación capaz de correr en la nube, la cual comprende una base de datos, y una página web que permita la visualización de los mismos.

5. Análisis de costo-rendimiento para la implementación de los sensores.

6. Diseño y desarrollo de prototipo de estructuras auxiliares para instalación de hardware.

1.3. Metodología

En primer lugar se diseña el sistema permitiendo obtener los requisitos del sistema. A grandes rasgos, se pueden diferenciar tres tareas principales: implementar un algoritmo de detección óptica de caracteres para obtener la patente de los vehículos, diseñar e implementar un sistema capaz de capturar la fotografía para posteriormente procesarla, y desarrollar una aplicación WEB para permitir el acceso a la información a los dueños del establecimiento.

El desarrollo de la obtención de la patente de un vehículo a partir de una fotografía. La conversión fotografía-patente se realiza utilizando un algoritmo de detección óptica de caracteres usando redes neuronales convolucionales. Posteriormente se desarrolla una implementación del algoritmo en Python 3, para poder utilizarlo en un servidor web, y en sistemas embebidos.

El diseño de las placas secundarias se lleva a cabo con Raspberry Pi 3B+ y NVIDIA Jetson TX1, con la idea de desarrollar dos versiones del sistema, los cuales son denominados como SL mini y SL respectivamente. En esta etapa se definen los sensores necesarios para la tarea de obtener la fotografía de los vehículos. Además se hacen las implementaciones de software necesarias para realizar la tarea.

Para el desarrollo del servidor se eligió una variedad de tecnologías, principalmente escritas en Python 3 y JavaScript, ya que son muy utilizadas en la actualidad. Esta etapa incluye el diseño de un panel para los dueños de los establecimientos, permitiendo acceder a la información y cambiar algunas configuraciones básicas de

los sistemas SL. Además los administradores cuentan con un panel específico para administrar los sistemas SL, permitiendo crear nuevos sistemas.

Para estudiar el rendimiento del sistema se realizan diferentes pruebas tales como: prueba de exterior, distancia y ángulos variables y creación de registros de entrada y salida. La prueba en exterior permite ver el rendimiento del algoritmo de detección óptica de caracteres en un ambiente para el cual no fue diseñado. El análisis de distancia y ángulos hace énfasis en el comportamiento del sistema en entornos controlados, y se realiza con la finalidad de estudiar los límites del algoritmo de detección óptica de caracteres. Por última la prueba sobre el servidor, permite observar si el comportamiento de la aplicación web es correcta, permitiendo crear registros y visualizar los datos de forma correcta.

1.4. Estructura de la tesis

El informe de este proyecto integrador profesional cuenta con siete capítulos. En este primer capítulo se introducen las ideas generales del proyecto, los objetivos y como se llevaran a cabo.

En el segundo se desarrolla el sistema de forma general, como el algoritmo a implementar, se detallan los protocolos de comunicación a utilizar y la arquitectura de la aplicación WEB.

En el tercer capítulo se implementa el algoritmo de detección óptica de caracteres, dando los fundamentos teóricos necesarios para comprender como funciona el mismo. Este capítulo termina con el algoritmo implementado en Python 3.

En el cuarto capítulo se desarrolla e implementan los sistemas SL, realizando un análisis de sensores a utilizar, diseñando un contenedor para los sensores y se realiza la implementación de los drivers de los sistemas SL.

En el quinto capítulo se implementa la aplicación WEB partiendo del esquema definido en el capítulo 2.

En el sexto capítulo se detallan las pruebas realizadas tanto en el proceso de detección de caracteres como en el rendimiento del servidor.

Por último se describen las conclusiones alcanzadas y posibles mejoras que se

pueden realizar para obtener un mejor rendimiento del sistema.

Capítulo 2

Diseño del sistema

En este capítulo se mostrará el desarrollo general del sistema, comenzando por una visión superficial del sistema hasta obtener la versión final del sistema. En capítulos posteriores se mostrará el proceso de diseño e implementación de cada parte del mismo.

2.1. Descripción general del sistema

El requisito fundamental de este trabajo fue poder controlar el sistema de entrada y salida de vehículos de al menos un recinto. Por lo tanto primero se explicará el caso particular de cómo funciona el sistema para el caso de un solo lugar. Adicionalmente existen otros requisitos preestablecidos, como:

- Medir el tiempo del vehículo dentro del establecimiento.
- Que la entrada y salida sea automática basada en la patente.

De estos requisitos se desprende la forma más general del sistema, que detecta los caracteres del vehículo, permite la identificación del vehículo, y basado en la hora de entrada y salida, calcula el tiempo que estuvo el vehículo. En la Fig. 2.1 se observa un modelo de caja negra del sistema. Existe un gran inconveniente a la hora de obtener el texto de la patente de una imagen y es el tiempo de procesamiento. Es por ello que se utilizará un sensor de distancia para garantizar que existe un objeto. La versión correspondiente del sistema se puede observar en la Fig. 2.2.

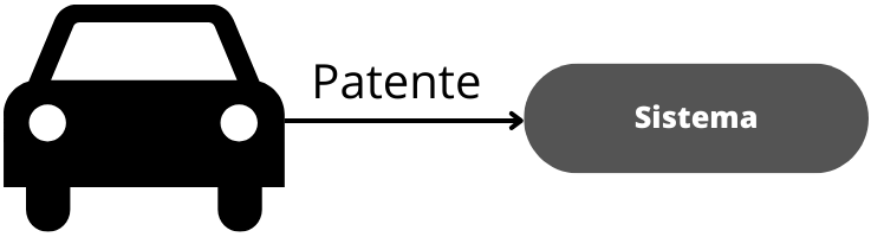


Figura 2.1: Idea general del sistema a implementar, en esta versión se toma al bloque de sistema como una caja negra que solo necesita la patente para realizar su tarea.

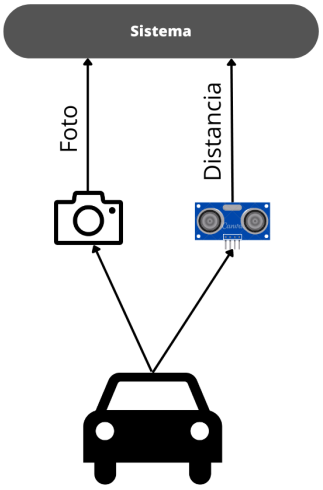


Figura 2.2: Sistema con sensor de distancia y cámara.

Debido a que los parques de estacionamiento, como los de los supermercados, tienen entradas y salidas diferenciadas, el sistema deberá tener un comportamiento diferente según donde se encuentre. Es por ello que se implementarán dos algoritmos, uno para manejar las entradas y otro para manejar las salidas.

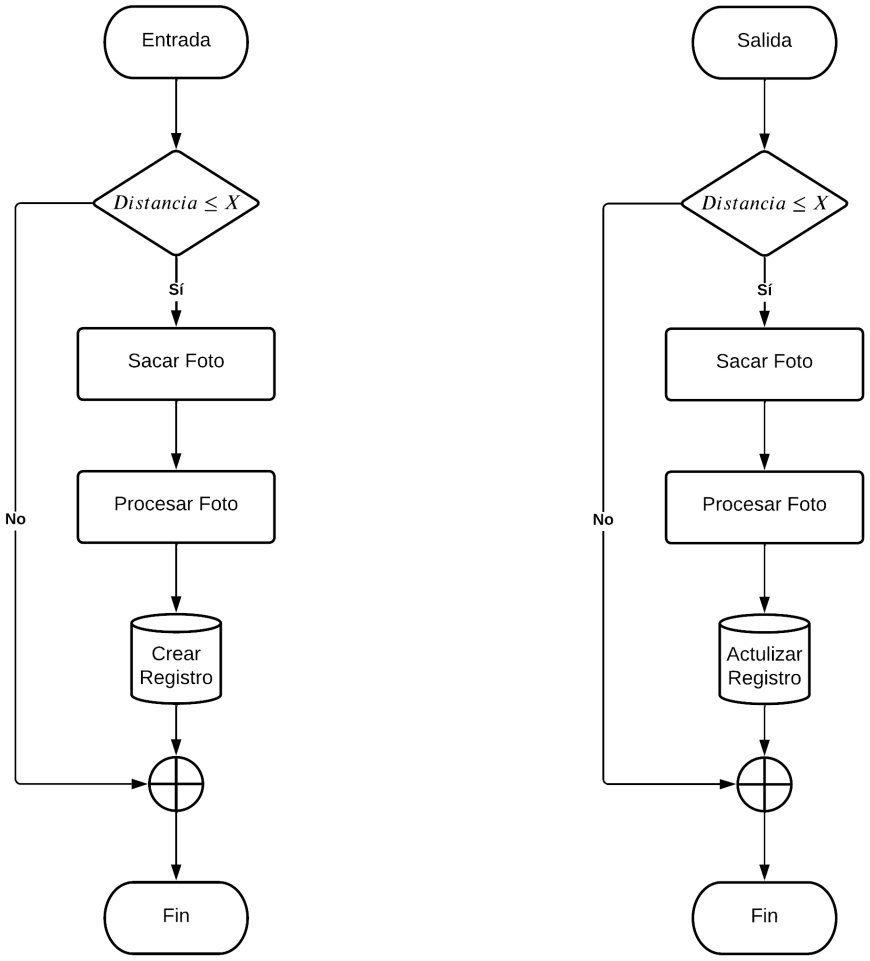
2.2. Algoritmos de entrada y salida

Una vez establecido como se verá el sistema, se debe definir su comportamiento cuando un vehículo entra o cuando sale.

2.2.1. Algoritmo de entrada

La toma de muestra, es decir, una fotografía, se da cuando el sistema detecta que tiene un objeto a menos de Xcm. Luego, se procesa la imagen, para obtener la patente en formato de texto. En caso de encontrar una patente, se almacena la

fecha, junto con la patente y la fotografía Fig. 2.3a.



(a) Entrada. (b) Salida.

Figura 2.3: Diagramas de flujo para la entrada y la salida de un vehículo del estacionamiento.

2.2.2. Algoritmo de salida

El algoritmo de salida es idéntico en la toma de muestra, pero antes de guardar la información de salida (foto y fecha), verifica que exista un registro de entrada correspondiente y se actualiza el mismo Fig. 2.3b.

2.2.3. El problema de múltiples establecimientos

Ambos algoritmos poseen un pequeño inconveniente a la hora de extender el proyecto en más de un recinto. Es por ello que cada barrera se vincula a un lugar, y los registros se buscan y actualizan en función del lugar que tiene asignado cada barrera.

2.3. Partes del sistema

Una vez diseñado y comprendido como está compuesto el sistema a grandes rasgos, es importante entender las partes del mismo y lograr diferenciar que parte del algoritmo se ejecutará en el sistema SL y cuál en el servidor. Para la comunicación del SL con el servidor se optó por utilizar 2 protocolos de comunicación: por un lado HTTP para el envío y actualización de registros de entrada/salida, y MQTT para cambios en la configuración de las barreras, en el capítulo 5 se realizará una explicación de cada protocolo. En la Fig. 2.4 se observa como interactúan los vehículos con la plataforma, la flecha en rojo se utiliza para destacar el envío de datos por MQTT.

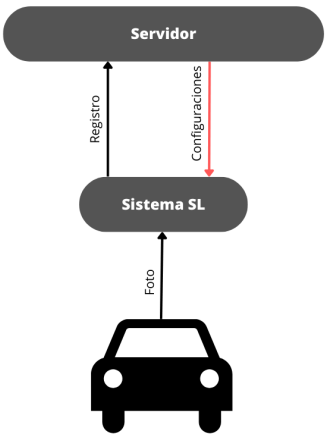


Figura 2.4: Sistema separado en SL y servidor.

Esta es la versión que se desarrollará en los capítulos siguientes.

2.3.1. Interacción con el cliente

Debido a que el enfoque del diseño implica un cliente que desea utilizar el sistema se desarrolla una plataforma web para simplificar el manejo de configuración y administración del sistema. En la Fig. 2.5 se observa la arquitectura a implementar en el servidor, teniendo en cuenta las interacciones y su manejo entre los sistemas SL y los clientes.

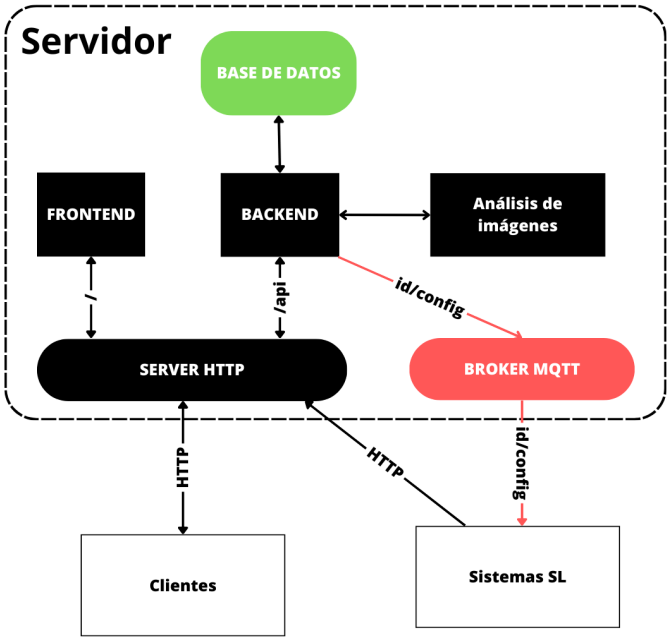


Figura 2.5: Esquema de comunicación Cliente-Servidor-SL.

Capítulo 3

Detección óptica de caracteres

En este capítulo se explicará que es la detección óptica de caracteres (OCR), y se buscará obtener una idea básica de los distintos algoritmos que se pueden implementar para utilizar esta técnica. Luego se presentará la red convolucional entrenada por Ankandrew [8], con la idea de finalizar en un algoritmo en Python 3 que permita obtener los caracteres de una patente mediante una imagen.

3.1. Algoritmos de OCR

El OCR, por sus siglas en inglés Optical Character Recognition o en español Reconocimiento Óptico de Caracteres, es una técnica que permite obtener texto en formato ASCII a partir de una imagen. Para lograrlo se debe inspeccionar la imagen buscando formas características de los símbolos como los del abecedario. La tecnología más utilizada en esta última década para realizar el OCR son las Redes Neuronales donde se destacan dos tipos: las redes LSTM, por sus siglas en inglés Long-Short Term Memory, que son una variedad de Redes neuronales recurrentes, y las redes neuronales de convolución o CNN por sus siglas en inglés Convolutional Neural Network, o en español Red Neuronal Convolucional.

Si bien el fin último de ambos tipos de red es obtener los caracteres a partir de la imagen, la forma en la que trabajan difiere significativamente.

3.1.1. Redes LSTM

Las redes LSTM almacenan información de estados anteriores mediante bucles, lo que les permite realizar predicciones de estados futuros usando información pasada almacenada y la información del estado actual. En la Fig. 3.1 se observa un bloque LSTM básico, donde se pueden apreciar las diversas funciones que lo componen. Los bloques σ corresponden a las funciones de activación que en general son sigmoides, los bloques g y h son funciones de activación diferentes que en la mayoría de los casos suelen ser tangentes hiperbólicas. Las señales principales son:

- $y^{(t-1)}$: corresponde al estado oculto anterior.
- x^t: es la entrada del estado actual.
- $c^{(t-1)}$: representa el estado previo del bloque.
- c^t: representa el estado actual del bloque.
- y^t: corresponde al estado oculto actual.

Cada bloque LSTM se puede dividir en tres secciones diferentes que son:

- Compuerta de memoria: representado por la función σ, en ella se decide a que información se le debe prestar atención y cuál debe ser ignorada, en otras palabras, es la encargada de ponderar la importancia de la entrada actual respecto a la información pasada.
- Compuerta de entrada: esta se obtiene de la multiplicación de las señales z e i, es la encargada de actualizar el estado del bloque con la información ponderada de la compuerta de memoria por el vector z, cuyos valores se encuentran entre -1 y 1.
- Compuerta de salida: aquí se determina el valor actual del estado oculto y queda contenida la información de los estados previos, este valor es utilizado para realizar la predicción de estados futuros.

En forma resumida se puede decir que la compuerta de memoria determina que información de los estados anteriores es relevante, la compuerta de entrada decide que información es necesaria añadir al estado actual, y la compuerta de salida determinada el estado oculto actual.

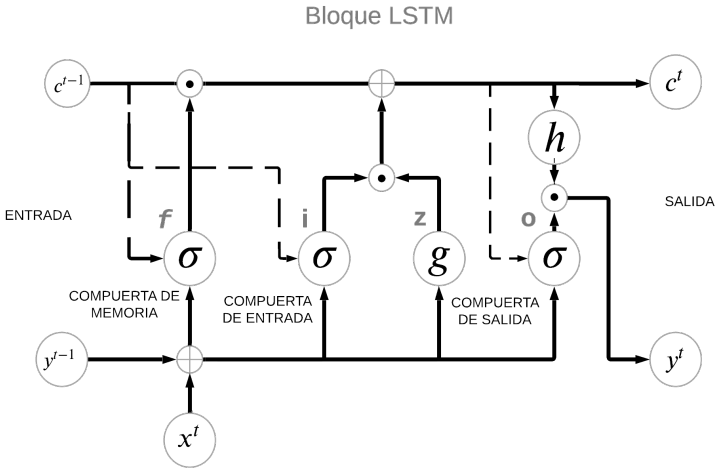


Figura 3.1: Diagrama del bloque LSTM.

Actualmente, este tipo de redes son las más utilizadas para realizar reconocimiento de caracteres, pero cuentan con la desventaja de un alto costo computacional, por lo que se desaconseja su uso en sistemas embebidos.

3.1.2. Redes CNN

La otra alternativa que se puede utilizar son las CNN, que mediante filtros bidimensionales y una serie de capas totalmente conectadas se predice el texto más probable. Al no requerir almacenar información ni trabajar de forma recursiva, la implementación de este tipo de redes se puede hacer en equipos con un hardware de menor potencia, menor tamaño y, por consiguiente, menor costo. La mayor desventaja es que la cantidad de caracteres debe estar previamente definido.

Las patentes argentinas han sufrido tres modificaciones a lo largo de los años, que se pueden ver en la Fig. 3.2. En la actualidad, se pueden encontrar vehículos con las patentes lanzadas en 1994 y 2015. Si se considera un séptimo carácter en las de 1994 como un guion bajo (_), es posible utilizar una red neuronal del tipo CNN frente a una LSTM.

Figura 3.2: Evolución patentes Argentinas a lo largo de los años.

3.2. Definición de red neuronal

Se entiende como red neuronal a un algoritmo computacional que imita el funcionamiento del cerebro humano. En este caso el trabajo se centrará en la tarea de clasificación de datos. Debido a la semejanza que existe entre los algoritmos de Deep Learning [9] (algoritmos de aprendizaje profundo) y el cerebro humano, a la unidad fundamental de estos sistemas se la denomina neurona.

3.2.1. La neurona

La neurona es la unidad fundamental dentro de la red, y su nombre viene de la similitud que existe entre este elemento y una neurona humana, esto se puede apreciar en Fig. 3.3. La tarea de la neurona es procesar la información que le entra para producir un valor de salida, a este proceso se lo conoce como sinapsis.

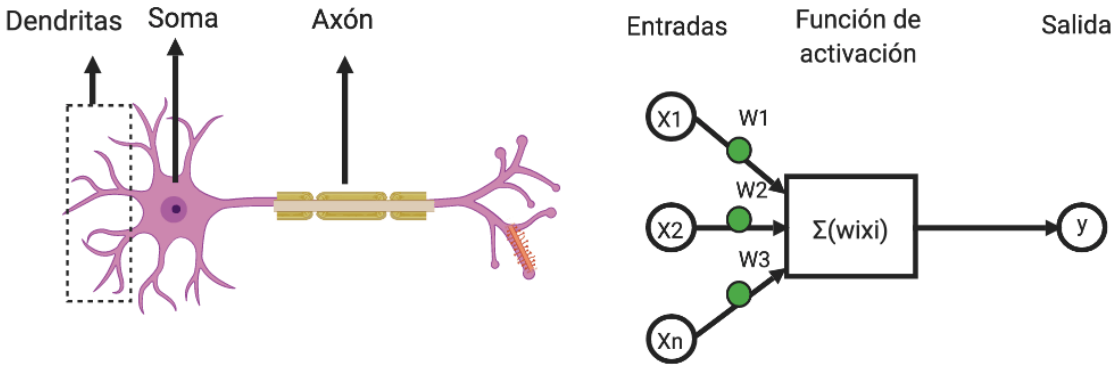


Figura 3.3: Comparación neurona con neurona artificial.

La sinapsis entre neuronas se modela como una suma ponderada que puede ser expresada como $y = W^T \cdot X + b$ donde y es el valor de salida de la neurona, W

representa los pesos de la neurona y se define como un vector columna de dimensión N, b es un bias y X es el vector columna de entrada cuya dimensión es N. Si bien la neurona es muy útil, sola no es de mucha utilidad, es por ello que a los arreglos en paralelo de neuronas se los denomina capa, a la conexión de capas de neuronas simples se las denomina capas totalmente conectadas, ya que toda la información de la capa anterior se envía a la capa siguiente.

3.3. Redes neuronales

A la hora de resolver problemas más complejos es usual necesitar más de una capa, es por ello que se interconectan para formar una red neuronal como se observa en la Fig. 3.4.

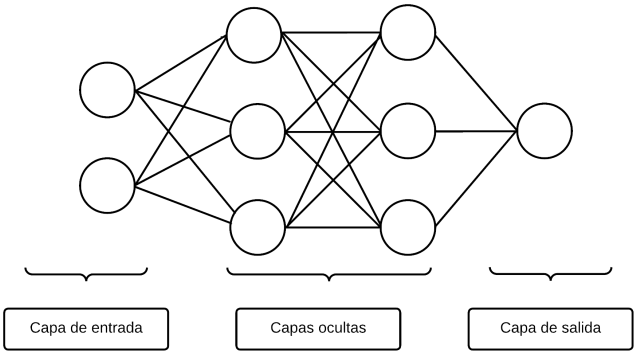
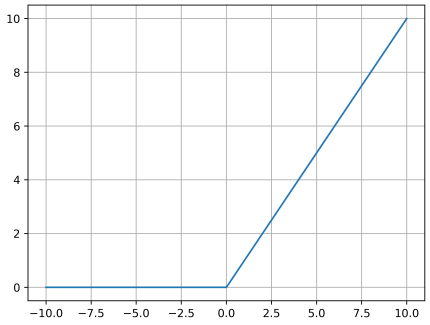


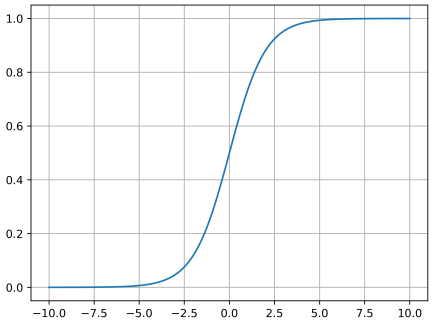
Figura 3.4: Esquema simplificado de una red neuronal, donde los círculos representan una neurona junto con la función de activación.

Debido a que en esencia el proceso que realiza la neurona es una transformación lineal, al interconectar capas la resultante sigue siendo una transformación lineal. Este problema de linealidad se soluciona aplicando una transformación no lineal, la cual se suele llamar función de activación, luego de que la información es procesada por la neurona, obteniendo $y = f(W^T X + b)$. Las funciones de activación más conocidas se pueden observar en la Fig. 3.5.

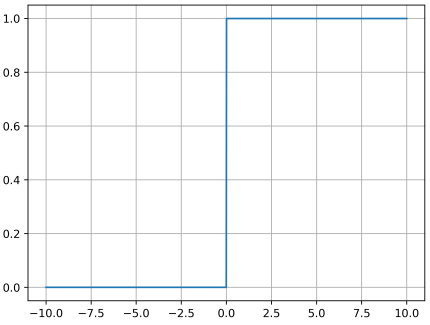
En resumen una red funciona de la siguiente manera, se realiza la multiplicación de la entrada por los pesos de la red, luego los resultados son ingresados a la función de activación para quitar las linealidades. Posteriormente esta información se pasa a



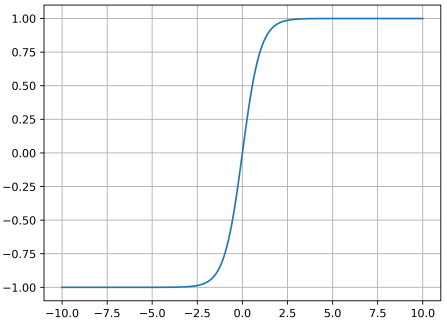
(a) ReLU: $f(x) = \max(0, x)$



(b) Sigmoide: $f(x) = 1/(1 + e^{-x})$



(c) Escalón: $\mu(x)$



(d) Tangente hiperbólica: $\tanh(x)$

Figura 3.5: Funciones de activación más utlizadas.

la capa siguiente y se realiza el mismo proceso. Esto se repite hasta llegar a la capa de salida donde, en tareas de clasificación, se suelen colocar tantas neuronas como salidas posibles.

3.4. Redes neuronales de convolución

Si bien hasta ahora se han analizado solo las capas totalmente conectadas, existen una gran cantidad de capas diferentes. En este trabajo se presenta una breve introducción a las redes de convolución o CNN.

La función principal de las capas de convolución es poder extraer información de una imagen, es por ello que nace la semejanza entre las CNN y la corteza visual de los animales. Este tipo de redes poseen características que las hacen únicas, por lo que se mencionan las partes que la componen. En la Fig. 3.6 se observa un esquema. A continuación se mencionan las partes más importantes de las CNN junto a una

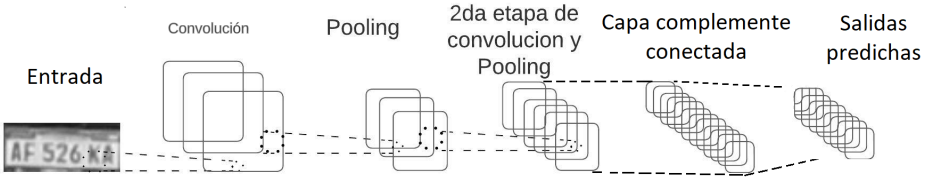


Figura 3.6: Ejemplificación de una red neuronal de convolución.

breve descipción para el caso de imágenes en escala de grises:

- Capa convolucional: compuesta por filtros entrenables de dimensiones predefinidas. Cada filtro realiza una multiplicación punto a punto recorriendo toda la imagen y produciendo un mapa de activación bidimensional, es decir, otra imagen. En la Fig. 3.7 se ve el funcionamiento de esta capa. Cada uno de los filtros se activarán según la característica que aprenda la red.

- Capa de agrupación o Pooling: se coloca entre las capas de convolución, su trabajo consiste en reducir el tamaño de cada mapa de activación, lo que reduce la complejidad para evitar el sobreajuste de los parámetros.

- Función de activación.

- Capa de aplanamiento o Flatten: esta capa transforma las imágenes o matrices de dimensión $N \times N$ en un vector columna N^2M, donde M es la cantidad de filtros de la capa anterior.

- Capa completamente conectada o Fully Connected: Se encargan de procesar la información de los filtros para obtener la cantidad de salidas esperadas.

Si bien la descripción de las capas de convolución, agrupación y función de activación se presentan de forma separada, se suelen implementar juntas, ya que es común tener varias capas de convolución conectadas en serie, permitiendo extraer información más específica de la red.

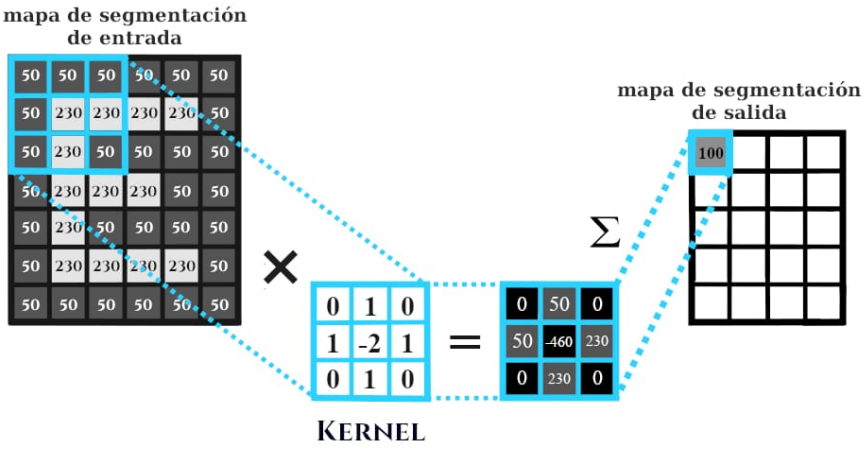


Figura 3.7: Esquema de funcionamiento de una capa convolucional 2D [10].

3.4.1. Convolución en 2D

La convolución bidimensional es similar a la 1D. En este caso la respuesta al impulso $h[n, m]$ se denomina filtro o kernel y, análogo al caso 1D, recorre todo el arreglo de entrada para producir un nuevo arreglo denominado mapa de características. La dimensión del kernel se suele definir como $(2k+1) \times (2k+1)$ usualmente se utilizan de 3×3 y 5×5.

Si se define a $h[n, m]$ como un filtro de dimensión $(2k + 1) \times (2k + 1)$ e I una imagen a escala de grises, cada celda (i, j) de la imagen de salida O es el resultado de la convolución entre h e I dado por

$$O(i, j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u, v]I[i - u, j - v] \tag{3.1}$$

3.4.2. Capa de agrupación

La capa de agrupación o pooling se encarga de reducir el tamaño de la imagen ejecutando una función a una submatriz de $n \times n$ de la imagen pasando de tener n^2 elementos a un elemento, las funciones de polling más utilizadas son:

- MaxPooling: se queda con el valor máximo de la submatriz.
- AveragePooling: calcula la media de los elementos.

Otra forma de entender a la capa de pooling es que realiza un submuestreo de la imagen de salida de la capa de convolución que devuelve un valor segun las funciones nombradas anteriormente.

En la Fig. 3.8 se puede apreciar el comportamiento de una capa de maxpooling. Luego de la capa de convolución, se divide el mapa de activación en pequeñas matrices de 2×2 y se extrae el valor máximo de cada submatriz, reduciendo la matriz de un tamaño de salida de 5×5 a una de 3×3, en este caso.

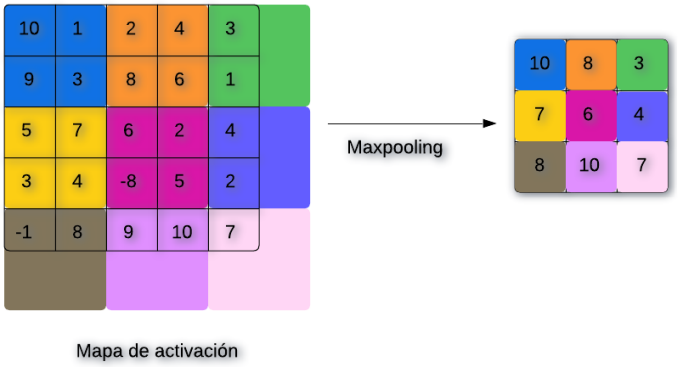


Figura 3.8: Ejemplo de funcionamiento de Maxpooling.

3.4.3. Técnica de Batch-Normalization

La técnica de Batch-Normalization se utiliza para reducir la covarianza de los datos de entrada. Esto se realiza normalizando los datos de entrada en un rango de 0 a 1 evitando trabajar con números grandes. Si bien este proceso se suele realizar con los datos de entrada, es útil realizarlo entre capas antes de la función de activación para trabajar siempre con datos normalizados, lo que permite un ajuste de parámetros más simple. Finalmente antes de pasar a la capa siguiente se aplica la función de activación.

3.5. CCN-OCR

Para este trabajo se utilizó una CNN diseñada y entrenada por Ankadrew [8], cuya arquitectura se puede ver en Fig. 3.9. De la arquitectura de la red se destaca la capa de Global Average Pooling, la cual remplaza a la capa de aplanamiento y

calcula la media por cada imagen de salida de la capa anterior, produciendo un valor por cada imagen . La comparación entre estos métodos de aplanamiento se observa en la Fig. 3.10.

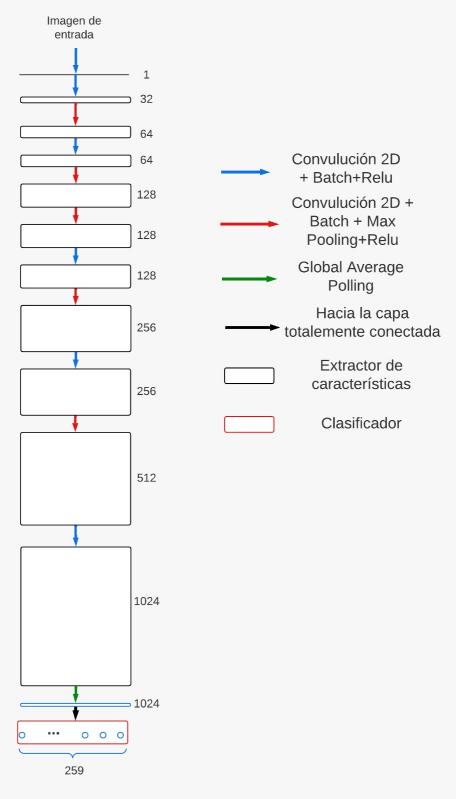


Figura 3.9: Arquitectura de la CNN-OCR de 1.3 millones de parámetros optimizados para CPU.

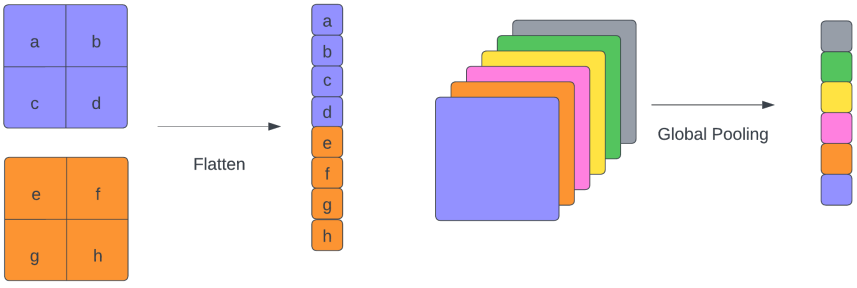


Figura 3.10: Comparación entre una capa de Flatten y una de Global Pooling.

Otro elemento a destacar de la red son las siete capas densas conectadas en paralelo, que en la Fig. 3.9 fueron representadas como una única capa densa para mejorar la visualización del esquema, debido a que, como se anticipó en secciones

anteriores, las patentes poseen siete caracteres. En esencia cada capa densa es idéntica, solo se diferencian sus pesos que son aprendidos durante el entrenamiento de la misma. Cada capa posee en total treinta y siete caracteres de salida, esto surge de las veintiseis letras del abecedario, los diez números del 0 al 9 y el símbolo de guion bajo (_) utilizado en el caso de las patentes antiguas que solo poseen seis caracteres, la salida de cada neurona de la capa de clasificación se puede interpretar como la probabilidad estimada de dicho carácter.

Como aclaración para el resto del trabajo, se referirá a esta red como CNN-OCR.

3.6. Requerimientos

Si bien la CNN es muy útil para la tarea a resolver, posee una serie de requisitos previos en la imagen de entrada. Según el autor, la CNN necesita como entrada una imagen de la patente en blanco y negro de dimensiones 70×140 pixeles. Por lo que es necesario realizarle un tratamiento previo a las imágenes antes de ingresarlas por la CNN-OCR. El tratamiento necesario se puede observar en la Fig. 3.11. Entonces existen tres pasos previos luego de sacar la captura antes de ingresarla a la CNN-OCR:

1. Obtener un recorte de la patente.
2. Transformar la imagen a escala de grises.
3. Redimensionar la imagen al tamaño requerido.

(a) Imagen obtenida por la cámara. (b) Imagen requerida por la CNN-OCR.

Figura 3.11: Ejemplo de imagen obtenida e imagen requerida por la CNN-OCR.

3.6.1. Obtención del recorte de la patente

Debido a la importancia de esta etapa y a su complejidad, se realiza una explicación extendida, con la finalidad de tener una mejor comprensión del funcionamiento del proceso de recorte necesario para que la CNN-OCR pueda funcionar de manera óptima. Siguiendo la recomendación de [8], se utilizó una red complementaria para detectar la ubicación de la patente y luego realizar el recorte. La red recomendada para esta tarea es el detector de una etapa Yolo (You Only Look Once) en su versión 4 o YoloV4 [11], como se muestra en la Fig. 3.12. Esta red es un sistema de código abierto el cual se centra en la detección de objetos. Utiliza una arquitectura Darknet [12], escrita en C.

Los detectores modernos basados en redes neuronales convolucionales (CNN) contienen tres partes importantes: Backbone, Neck y Head que trabajan en conjunto para realizar la detección de objetos en imágenes.

- Backbone: es una arquitectura de deep learning que actúa como extractor de características en la imagen de entrada.
- Neck: su funcionalidad es tomar mapas de características de diferentes partes del Backbone y agregarle características, para facilitar la clasificación de los objetos.
- Head: es la parte encargada de reconocer objetos, básicamente busca una región en la que pueda haber un objeto, sin distinguir cual sea. Usualmente se compone de dos partes. La primera utiliza subimagenes de tamaño fijo, y la segunda varía el tamaño de las subimagenes para adecuarse mejor al tamaño de cada objeto. En la Fig. 3.12 se muestra un Head genérico separado en Dense Prediction y Sparse Prediction.

Para este trabajo el Backbone fue entrenado por el mismo autor de la CNN-OCR para que reconozca patentes argentinas [8]. Debido a que la Darknet está escrita en C fue necesario compilarla, esto se hizo siguiendo los pasos del repositorio de AlexeyAB [12].

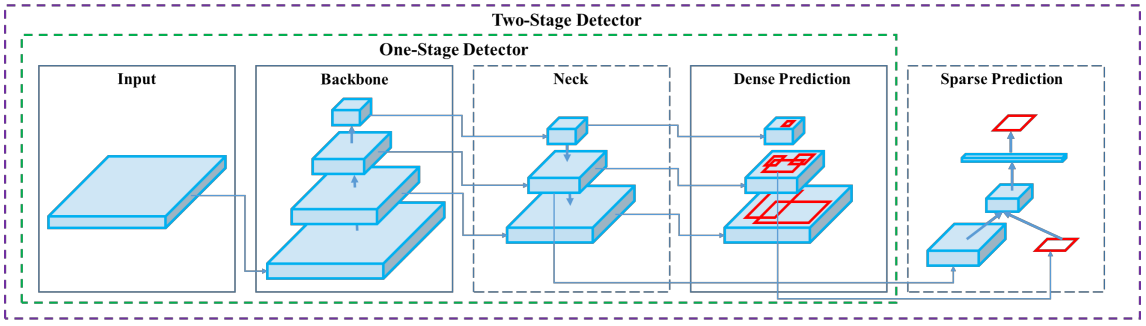


Figura 3.12: Esquema general de un detector de una y dos etapas. [11].

3.6.2. Transformación a escala de grises

El proceso de conversión a escala de grises se puede hacer de diversas maneras. La forma más utilizada es realizar un promedio de las capas de color Rojo (R), Verde (G), Azul (B) quedando descripto por la ec. 3.2. Otra forma es emplear un promedio ponderado para compensar la sensibilidad relativa del ojo humano la cual se puede observar en la ec. 3.3. En la Fig. 3.13 se observa que la conversión a escala de grises teniendo en cuenta la sensibilidad del ojo humano obtiene un mejor resultado.

$$I = \frac{R + G + B}{3} \tag{3.2}$$

$$I = 0{,}2989R + 0{,}587G + 0{,}114B \tag{3.3}$$

(a) Imagen original. (b) Promedio. (c) Promedio ponderado.

Figura 3.13: En (a) se observa la imagen original, mientras que en (b) y (c) la imagen convertida a escala de grises por los diferentes métodos.

3.6.3. Redimensionamiento de la imagen

Para redimensionar la imagen se utiliza un algoritmo de interpolación bilineal la cual no se explicará debido a que su complejidad está fuera del alcance de este

trabajo.

3.7. Implementación del algoritmo en Python 3

Para la implementación del algoritmo se utilizó el lenguaje Python 3, ya que cuenta con una gran cantidad de librerías necesarias para el funcionamiento del sistema completo. La descripción general del algoritmo se puede observar en Fig. 3.14.

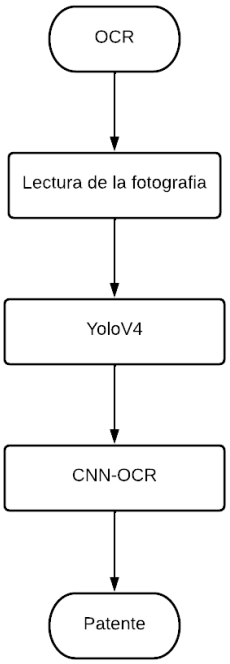


Figura 3.14: Algoritmo general de OCR.

Figura 3.15: Flujo de una fotografía hasta obtener los caracteres de la patente.

La lectura u obtención de una fotografía se realizó mediante OpenCV [13] ya que permite utilizar una gran variedad de fuentes tales como cámaras y archivos. Luego se procesa mediante la YoloV4, que devuelve la ubicación relativa de la patente en

	Raspberry Pi 3b+	Jetson TX1	I5-11600K	I5-1135G7
Tiempo [s]	85.78	6.15	1.05	1.20

Tabla 3.1: Tiempos obtenidos del procesamiento de diez imágenes.

la imagen en un archivo de texto plano lo que permite obtener la patente recortada. Antes de pasar el recorte a la CNN-OCR se lo transforma a escala de grises y se redimensiona a 70×140 utilizando una interpolación bilineal y se normalizan los valores de la imagen entre 0 y 1 usando OpenCV. Finalmente se procesa la imagen por la CNN-OCR obteniendo la patente en texto. En capítulos posteriores se explicará con mayor detalle que sucede con la imagen, y la patente obtenida (Fig. 3.15).

3.7.1. Prueba de rendimiento

Con la finalidad de dimensionar los tiempos de procesamiento del algoritmo de OCR, se procesaron diez imágenes y luego, se calculó el promedio para obtener el tiempo por imagen en diferentes sistemas. Los resultados se encuentran en Tab. 3.1. Se observa que la Raspberry Pi es 1395 % más lenta que la Jetson TX1 para esta tarea, mientras que los otros sistemas poseen tiempo similares.

Capítulo 4

Diseño e implementación de los sistemas SL y SL mini

En este capítulo se realizará una explicación del diseño de los sistemas SL y SL mini, analizando las diferentes alternativas de placas, junto con sensores seleccionados y el modelado en 3D de la caja que alojará los mismos. Por último se detallará la implementación de los drivers escritos en Python 3, considerando elecciones de diseño.

4.1. Análisis de requisitos

Para definir algunos requerimientos es necesario evaluar las alternativas existen en el mercado de sistemas de acceso vehicular, con que placas de desarrollo y sensores encontramos en el ámbito local y proponer una solución capaz de suplir las desventajas o mejorar ventajas para los usuarios.

4.1.1. Sistemas actuales

Los sistemas actuales más comunes hoy en día son:

4.1.1.1. Sistemas tradicionales de tickets

Este sistema de control de ingreso y egreso suele generar molestias en muchos de los usuarios, por la necesidad de realizar alguna acción extra, como presionar un pulsador para obtener un ticket con los datos de entrada. Este sistema se puede apreciar en Fig. 4.1. Los tickets poseen un gran inconveniente, ya que si el usuario lo pierde se le cobra un valor fijo que, en general, es mayor al tiempo de la estadía. Estos sistemas cuentan con el inconveniente de que la impresión del ticket se realiza por impresión térmica, lo que requiere un papel termosensible. Además, la emisión de papel de un solo uso genera un impacto ambiental a tener en cuenta. En promedio un usuario tarda 11 segundos en entrar o salir del establecimiento [14].

A continuación se resumen las desventajas de este sistema:

1. Tiempo de acceso.
2. Emisión de papeles de un solo uso.
3. Necesidad de acciones por parte del usuario.

Figura 4.1: Sistema tradicional de acceso por barreras [15].

4.1.1.2. Sistemas de radiofrecuencia

Con el avance y el abaratamiento de los costos en la electrónica surgieron nuevos métodos que permitieron a los usuarios prescindir de la necesidad de un ticket o una persona que les permita el acceso. El método principal es el uso de controles remotos que al ser accionados, activan el mecanismo y abren el paso del vehículo.

Otro sistema que está ocupando gran parte del mercado en los últimos años es el que integra a la barrera un sistema de identificador de radiofrecuencia o RFID, que se puede implementar usando de radiofrecuencia (RF) en el vehículo, tarjetas o llaveros RF. Además es necesario instalar un receptor RF en la entrada para producir el enlace y permitir el acceso. La gran desventaja de este sistema es tener que instalar un emisor RF por cada vehículo o guardar la tarjeta de lectura.

Los problemas de este sistema son:

1. Necesidad de instalar un recetor de RF en los puntos de acceso.

2. Instalar un emisor RF por cada vehículo.

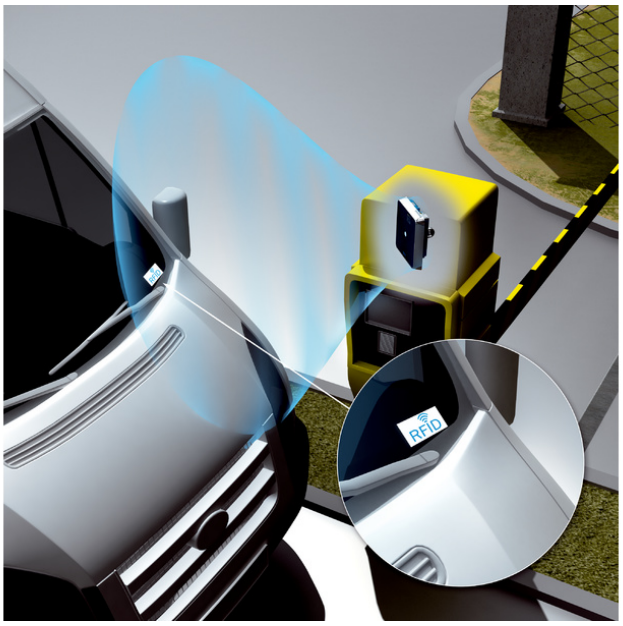


Figura 4.2: Sistema de acceso por RFID [16].

4.1.2. Requisitos de los sistemas SL

Los métodos anteriormente nombrados son muy útiles en el día a día, pero cuentan con una serie de inconvenientes que pueden ser atenuados mediante el OCR como el descrito en el capítulo 3. Dicha técnica utiliza un distintivo único de cada vehículo, como en el sistema de RFID, pero la distancia de actuación está dada por la distancia máxima de reconocimiento del algoritmo y los sensores empleados.

Debido a la necesidad de realizar OCR, se requerirá una cámara. Con la finalidad de poder acoplar algún sistema de activación para la cámara se exigirá que la placa elegida soporte protocolos como I2C, SPI y UART. En cuanto a la comunicación, como se implementará junto con un servidor web existe la necesidad de conexión Ethernet o Wifi. Finalmente para facilitar futuras actualizaciones se requiere que la placa pueda correr un sistema operativo basado en GNU/Linux.

En resumen los requerimientos son:

1. Colocación de una cámara.
2. Soporte a protocolos I2C, SPI, UART.
3. Conexión Ethernet o Wifi.
4. Tiempo de acceso menor a 11 segundos.
5. Sistema operativo basado en GNU/Linux.

4.2. Selección de placas

Teniendo en cuenta los requisitos planteados en la sección anterior se presentan varias opciones posibles: placas de la empresa Raspberry Pi, embebidos de la serie STM32, NVIDIA Jetson e incluso placas de la marca Arduino en sus versiones más potentes, por nombrar las más conocidas. Aquí es donde surge el primer inconveniente para tomar la elección de placa, cuál sería la mejor opción que cumpla tanto las necesidades y que sea accesible para poder realizar el proyecto. Luego de realizar una investigación de placas a las que se podía acceder se optó por dos modelos: uno basado en Raspberry Pi 3 B+ a la cual se la llamó SL mini, y el otro basado en NVIDIA Jetson TX1 denominada SL.

4.2.1. Características del SL mini

La placa del SL mini es una Raspberry Pi 3 B+ [17], Fig. 4.3, la cual cuenta con un procesador Broadcom BCM2837B0 y una CPU cuadcore Cortex A53 acompañada

de 1 GB de RAM LPDDR2, junto con un sistema operativo llamado Raspberry Pi OS el cual está basado en Debian.

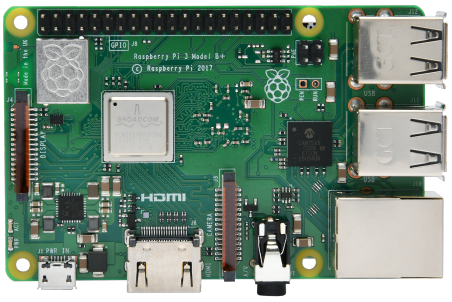


Figura 4.3: Raspberry Pi 3B+.

Otro de los puntos importantes que destacan de la placa son los pines GPIO, pines de entrada/salida de propósito general por sus siglas en inglés, lo que la hace sumamente sencilla a la hora de utilizarla junto a sensores comerciales. La amplia conectividad integrada que posee fue un punto que la destacó sobre otras posibles placas de desarrollo, ya que cuenta con puertos de conexión USB 2.0, puerto Ethernet, conexión Wifi 2,4 y 5,8 GHz y comunicación Bluetooth 4.2, que suple la necesidad de brindar conexión a internet de manera nativa sin necesidad de contar con periféricos extras que pueden encarecer el sistema.

La disponibilidad de la placa en el mercado fue un punto importante a considerar, ya que se considera una futura implementación del sistema a mediana o gran escala o la necesidad de recambio por rotura. Por otro lado, este hardware permite migrar a un modelo más actual como la versión 4, sin tener que realizar grandes cambios en los drivers diseñados.

4.2.2. Características del SL

La placa del SL es una Nvidia Jetson TX1 [18], Fig. 4.4, la cual cuenta con un procesador cuadcore Cortex A57 y 4GB de RAM LPDDR4. Además contiene 256 núcleos Nvidia Maxwell, lo que la vuelve una opción excelente en lo que se refiere al trabajo con imágenes y videos.

Para realizar la implementación se contó con el kit de desarrollo provisto por la empresa Nvidia. En él se pueden encontrar todas las conexiones mencionadas en

el SL mini, lo que permite el paso de los sensores de una placa a otra con suma facilidad. Este modelo cuenta con un sistema operativo JetPack 4.6.3 basado en Ubuntu 18.04.

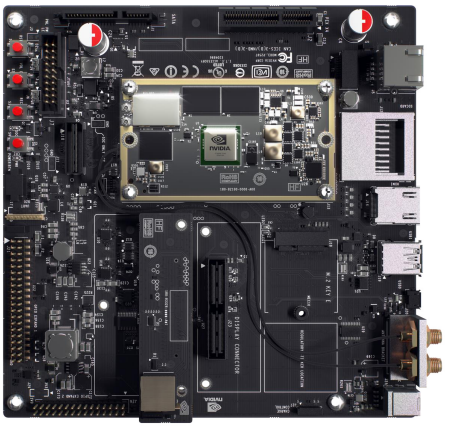


Figura 4.4: NVIDIA Jetson TX1 development kit.

En contra parte con el modelo mini, la placa del modelo SL está pensada para proyectos basados en redes neuronales y el procesamiento de imágenes, por lo que es posible integrar el algoritmo de OCR dentro del sistema, obteniendo un tiempo de respuesta de aproximadamente unos 6 segundos (Tab. 3.1), tiempo que se consideró aceptable.

Modelo	CPU	RAM	GPU
Raspberry Pi 3b+	4 a 1.4GHz	1GB LPDDR2	2 a 400MHz
Jeston TX1	4 a 1.7GHz	4GB LPDDR4	256 a 1GHz

Tabla 4.1: Resumen de características de las placas embebidas.

4.3. Evaluación y selección de sensores

En este apartado se comentará la selección de la cámara y el proceso de selección del sistema de actuación que, como se adelantó en el capítulo 2, se decantó por un sensor de distancia.

4.3.1. Selección de cámara

Una cualidad importante para la selección del tipo de cámara fue la conexión, que permitiera implementar el mismo modelo en ambas versiones de placa. Si bien, tanto Raspberry como NVIDIA proveen de módulos de cámara, debido a que los conectores no eran los mismos se descartaron estas opciones. La decisión final terminó siendo una cámara web con interfaz USB, debido a que hoy en día son fáciles de conseguir y son relativamente económicas. Particularmente el modelo utilizado es el que se observa en Fig. 4.5, el cual es un modelo genérico con una resolución de 1280×720 y $0{,}9MP$, siendo estas características aceptables para la tarea requerida.

Figura 4.5: Cámara utilizada.

4.3.2. Selección de activador

Con la finalidad de minimizar la captura de imágenes sin vehículos, se decidió colocar un sensor capaz de señalar que existe un objeto. A continuación se nombran las opciones analizadas:

4.3.2.1. Barrera infrarroja

Este sistema cuenta con un receptor y un emisor, separados lo suficiente para que un vehículo pase entre ellos. La captura se da cuando el vehículo interrumpe el paso de luz entre el receptor y el emisor.

El sistema de barrera infrarroja cuenta con una serie de inconvenientes que dificultan su implementación como:

1. Potencia mínima para activar el receptor.

2. Potencia máxima que puede emitir el trasmisor.

3. Ruido producido por la luz ambiente u otras fuentes lumínicas.

4. Apantallamiento del receptor debido a partículas de suciedad.

4.3.2.2. Placa de presión

La placa de presión cuenta de un accionador que luego de ser pulsado activa el sistema. Debido a que no fue posible armar un sistema de placa a presión, y tampoco se consiguió una opción económicamente viable, sumado a que las dimensiones pueden variar dependiendo el estacionamiento esta opción fue descartada.

4.3.2.3. Sensor de distancia

Los sensores de distancia son muy versátiles y existen una amplia variedad en el mercado es por ello que fueron seleccionados para la implementación de los prototipos. En particular para este trabajo se utilizó un sensor ultrasónico US-100.

Los sensores ultrasónicos funcionan emitiendo una onda ultrasónica por el transmisor y medir el tiempo que esta tarda en llegar al receptor.

El US-100 el cual se puede apreciar en la Fig. 4.6, posee un rango de medición de $2cm$ a $350cm$, compensado por temperatura, voltaje de alimentación entre $3V$-$5V$ y protocolo de comunicación UART. Por lo que es fácilmente integrable en ambas placas por medio del puerto GPIO sin necesidad de utilizar hardware extra.

Figura 4.6: Sensor de distancia ultrasónico US-100.

4.4. Diseño y ensamble

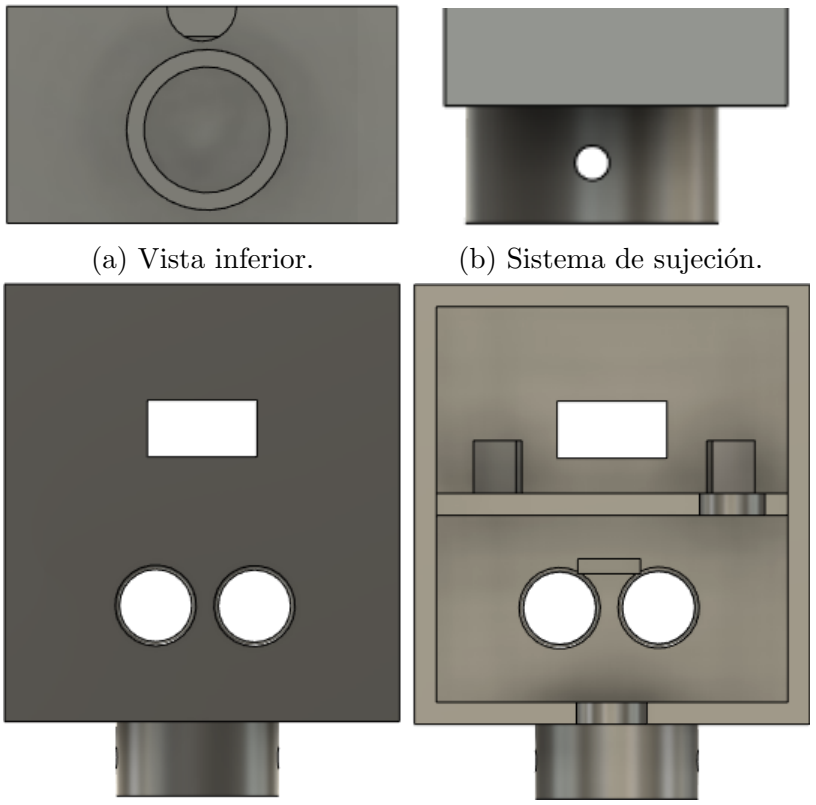
Para la implementación se definieron una serie de requisitos tales como la distribución de los sensores, la distancia desde el piso a la que se va a instalar, cómo se va a instalar en los establecimientos, entre otros. Las vistas del modelo 3D se pueden observar en la Fig. 4.7. El sistema de anclaje utilizado se puede observar en Fig. 4.7a y Fig. 4.7b, este sistema permite colocar un cilindro y ajustarlo utilizando los orificios del mismo, una de las ventajas de este sistema es poder ajustar la orientación del dispositivo. Se estableció que la distancia apropiada para la instalación es de 60cm de altura, ya que permite tomar una captura completa del vehículo y mejora el rendimiento del algoritmo de OCR. Se optó por colocar el sensor de proximidad en la zona inferior, ya que garantiza que la lectura que se tome sea de la trompa del automóvil, por lo que la cámara queda en la parte superior, como puede observarse en las Fig. 4.7c y Fig. 4.7d. Adicionalmente se diseñó una tapa rectangular que permita sellar el contenedor. En la Fig. 4.8 se observa el contenedor ensamblado y montado sobre un trípode.

4.5. Implementación de los drivers

Para terminar el diseño de los sistemas SL y SL mini, se explicará el diseño de los drivers. Durante la etapa de diseño el eje fundamental fue crear piezas de códigos reutilizables, priorizando el diseño modular del sistema. Es por ello que se implementaron una serie de módulos que permiten ser actualizados de manera sencilla y aíslan responsabilidades en diferentes partes. La implementación se realizó en Python 3. A continuación se realizará una breve explicación de las librerías creadas.

4.5.1. Lectura de distancia

Para la lectura de distancia se implementó una librería llamada *us100*, la cual utiliza *pyserial* [19] para la comunicación por puerto serie. Esta librería posee una función que devuelve la distancia promedio de un objetivo, para ello se utilizó una lista anidada de las últimas 5 muestras.



(a) Vista inferior. (b) Sistema de sujeción.

(c) Vista frontal. (d) Vista trasera.

Figura 4.7: Vistas del modelo 3D del contendor.

4.5.2. Utilidades

Se creó una librería llamada *slutils* la cual tiene como responsabilidad la conexión con el servidor MQTT, el envío de registros de entrada/salida, así como administrar la configuración del sistema y la captura de imágenes.

Para la conexión con el servidor MQTT[1] se utilizó *paho-mqtt* [20], escuchando el tópico $id/config$ donde el id es un identificador único por cada placa.

En cuanto al envío de los registros este se realiza por POST mediante HTTP utilizando la librería *requests* [21].

La captura de imágenes se realizó mediante la librería de *OpenCV*, que permite una fácil conexión con la cámara USB logrando tomar fotografías rápidamente.

[1]Este protocolo se explicará con mayor detalle en el siguiente capítulo.

(a) Contendor real ensamblado. (b) Sistema de sujeción del contenedor.

Figura 4.8: Contenedor ensamblado y montado sobre un trípode.

4.5.3. Diferencias entre SL y SL mini

Como fue nombrado con anterioridad, la mayor diferencia entre ambos sistemas es que el sistema SL es capaz de procesar la imagen de manera local. Esto implica que una conversión de los modelos de Tensorflow [22] utilizados para la CNN-OCR a una versión capaz de correr en ARM, por lo que se compilaron los modelos a Tensorflow Lite, utilizando las herramientas que Tensorflow provee para estos casos. Por otro lado, se requiere compilar la YoloV4 para arquitectura ARM, esto se realizó utilizando el compilador de CMake.

Capítulo 5

Diseño e implementación del servidor

En este capítulo se explicará el diseño y la implementación del server. En la Fig. 5.1 se observa la versión final del servicio, mostrando la interacción entre los Clientes, los sistemas SL y el Server.

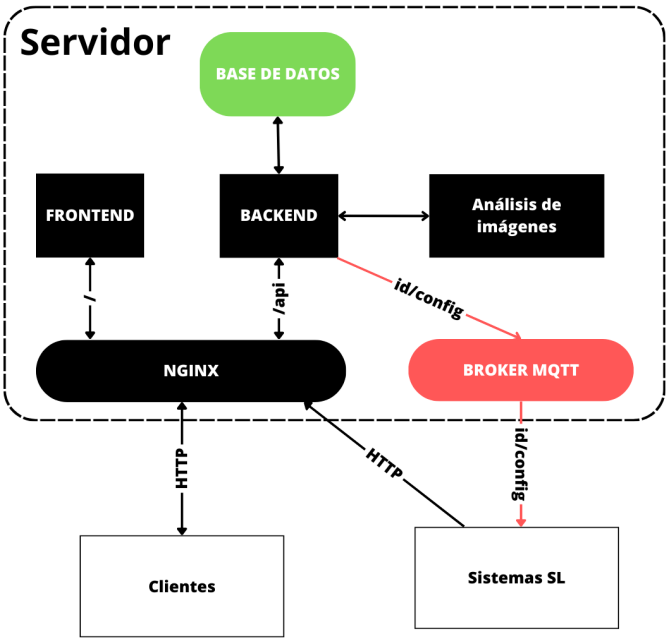


Figura 5.1: Esquema final del servidor.

5.1. Actualidad del desarrollo web

En la web existen dos grandes pilares: el front-end es la parte de la aplicación web con la que interactúan los usuarios o clientes. El back-end que es la parte que se conecta con la base de datos y ejecuta las validaciones necesarias para leer o escribir información. En general, el front-end y el back-end suelen ser dos proyectos separados y pueden ser construidos en diferentes lenguajes. Los lenguajes más utilizados para desarrollo back-end son: JavaScript (JS), Python, Ruby, entre otros [23].

En la actualidad el formato más utilizado para trasmitir información entre el back-end y el front-end es el formato JSON de sus siglas en inglés JavaScript Object Notation o traducido como notación de objetos de JavaScript. En la Fig. 5.2 se observa un ejemplo de JSON. Este tipo de notación se caracteriza por almacenar los datos en forma clave-valor, permitiendo un acceso a la información rápido y sencillo.

Figura 5.2: Ejemplificación de un JSON.

Hoy en día la web se ha transformado en la base de cualquier organización. Hace años las organizaciones solían usar aplicaciones de escritorio, pero en la actualidad las aplicaciones web han dominado el mercado.

5.2. Docker

Docker es una tecnología open source desarrollada por Docker Inc, la cual permite correr aplicaciones de forma aislada en contenedores.

5.2.1. Contenedores

Un contenedor funciona casi como una máquina virtual, pero sin entorno gráfico y compartiendo el Kernel [24] con el sistema que lo hospeda, lo que hace que los contenedores sean más livianos y mucho más rápidos que una máquina virtual, ya que estas últimas emulan todo el sistema e incluyen un puente entren el kernel del sistema padre y el sistema emulado. Cada contenedor suele estar especializado para cumplir una determinada tarea y, en general, se los denomina servicios.

5.2.2. Docker Compose

A pesar que Docker es una tecnología muy utilizada, por si sola puede ejecutar un único contenedor a la vez, existen herramientas para la administración de múltiples contenedores en simultáneo. La más conocida, y muy utilizada en el mundo del desarrollo es Kubernetes [25]. Debido a que el sistema aun no es lo suficientemente grande y no se cuenta con múltiples servidores para alojar los servicios, la implementación se realizó mediante Docker Compose [26], cumple un rol muy similar a Kubernetes, pero sin agregar tanta complejidad debido a poder administrar múltiples servidores.

El proceso de configuración de Docker Compose se hace creando un archivo de extensión YAML como el de la Fig. 5.3. Se puede observar que existen dos apartados principales.

5.2.2.1. version

Este atributo le indica a Docker Compose la versión del archivo de configuración que se está utilizando, en general, esta versión impacta directamente en qué atributos se pueden utilizar en el resto del archivo de configuración.

5.2.2.2. services

Se definen los servicios que se necesitarán. En la Fig. 5.3 se muestran tres contenedores *api*, *front*, *db*.

Dentro de cada contenedor se define el atributo *image* el cual indica que imagen se utilizará. Dicha imagen puede ser privada o se puede utilizar una imagen pública

Figura 5.3: Ejemplificación de una archivo *docker-compose.yml*

de Docker Hub que es un repositorio de imágenes de Docker. Las imágenes son en esencia un sistema operativo configurado para algunos entornos, por ejemplo la imagen *node:18-alpine* posee un sistema Alpine, que ya viene instalado con NodeJS en su versión 18, imagen muy utilizada para correr aplicaciones escritas en JS.

Otro campo que se destaca es el *working_dir* o directorio de trabajo, que es donde se almacenará el código de la aplicación. Es importante destacar que esta ruta se toma desde el contenedor, y no tiene por qué ser una ruta válida desde el sistema Host.

El atributo *volumes* es un array que vincula una carpeta del sistema Host con una carpeta del contenedor, permitiendo acceder a la información del contenedor. Esto es muy utilizado para realizar backup desde el sistema Host.

Por último, *ports* es un array que vincula los puertos del sistema anfitrión a los puertos del contenedor, lo que es sumamente útil cuando se desea escuchar un puerto específico para manejar peticiones HTTP, por ejemplo.

5.2.2.3. intranet

Si bien en el archivo de configuración no aparece la configuración de *intranet*, Docker Compose genera una red interna para interconectar los contenedores, lo que permite que el servicio *api* pueda acceder a la base de datos del servicio *db* mediante la ruta *mongodb://db:27017/sl*. La mayor ventaja de esta red es que hace inaccesible la base de datos para usuarios que no estén dentro de la red de Docker, disminuyendo la vulnerabilidad del sistema.

5.3. Protocolos de comunicación

Como fue nombrado en el capítulo 2, el sistema utiliza dos protocolos de comunicación: HTTP y MQTT, este apartado se realiza una breve introducción a cada protocolo.

5.3.1. HTTP

HTTP por sus siglas en inglés Hypertext Transfer Protocol o protocolo de transferencia de hipertextos, es la forma en la que se comunican los navegadores web para consultar datos a un servidor. Para este trabajo es importante tener en cuenta las siguientes características:

1. Half-Duplex: El cliente es quien pide la información y el servidor es quien responde dicha petición.

2. Cuenta con métodos específicos para indicarle al servidor la acción para llevar a cabo: GET (leer), POST (crear), PUT (actualizar), DELETE (eliminar), entre otros.

Este protocolo se utiliza debido a que permite a los usuarios visualizar información de sus recintos desde una aplicación web, además permite enviar la información desde los sistemas SL cuando entra o sale un vehículo.

5.3.2. MQTT

MQTT por sus siglas en inglés Message Queing Telemetry Transport o transporte de telemetría mediante mensajes, es un protocolo muy utilizado para aplicaciones de IoT (Internet of Things), debido a: bajo consumo energético, ancho de banda reducido, escalabilidad y confiabilidad. Este protocolo es basado en publicación/suscripción: Los clientes pueden publicar mensajes en tópicos, los mensajes son enviados a todos los clientes que estén suscriptos al tópico del mensaje, esto permite establecer una comunicación Full-Duplex entre el back-end de la aplicación y los sistemas SL.

5.4. Servicios

Una vez que fue descrito cómo se hará la implementación y su arquitectura, se procederá a entender cómo fue creada la aplicación web en su totalidad, describiendo cada parte del sistema.

5.4.1. Base de datos

La base de datos elegida fue Mongo DB, debido a que es una de las bases de datos más utilizada. Mongo esta basada en documentos, por lo tanto no posee vínculos fuertes entre los diferentes esquemas. Los datos se almacenan en BSON (binary JSON), el cual hace más fácil la transferencia de datos. Por otro lado, permite escalar de manera horizontal de forma más sencilla, ya que se suelen usar identificadores únicos y no incrementales, como es frecuente en bases de datos del tipo SQL.

El esquema de base de datos se observa en la Fig. 5.4, las fechas violetas indican una relación entre los documentos, mientras que las grises indican que existe un documento embebido en el documento padre.

5.4.2. Broker MQTT

Un broker MQTT es una aplicación back-end que coordina los mensajes entre los diferentes clientes. El broker más conocido es Mosquitto, el cual utiliza la versión 5.0 del protocolo. El único tópico implementado tiene la forma de "*id*/config"donde

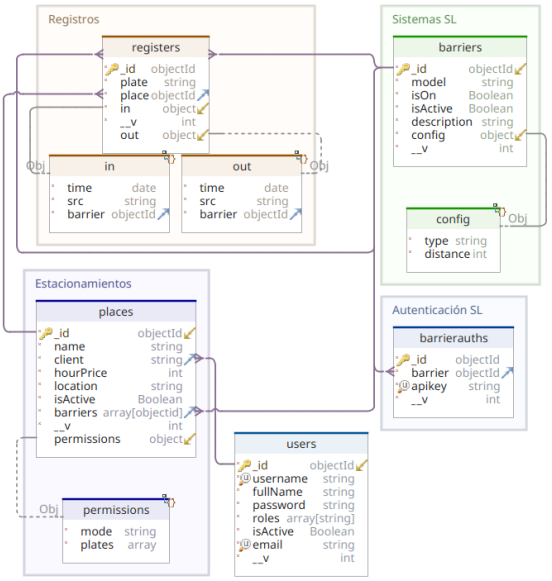


Figura 5.4: Esquema de la base de datos.

id es un identificador único de los sistemas SL, esto permite que existan tantos tópicos como sistemas SL estén funcionando, y solo pueden escuchar los cambios de configuración de cada una de ellas, lo que permite ahorrar procesamiento de datos en los sistemas SL.

5.4.3. Back-end

El back-end está implementado en NestJS [27]. Su principal función es administrar la información porque se conecta con la base de datos para realizar las tareas de lectura y escritura de documentos.

Con la idea de generar la mayor documentación posible de este servicio se utilizó una librería llamada Swagger la cual, junto a NestJS, genera documentación automática de las rutas HTTP. Esto facilita la tarea de crear una nueva aplicación así como la de incluir una nueva persona al proyecto. En la Fig. 5.5 se observa como se ve la documentación generada por Swagger.

Debido a la necesidad de autenticación de usuarios y de los sistemas SL, se implementó PassportJS que permite la creación de sistemas de validación de forma sencilla. Los métodos de autenticación implementados fueron:

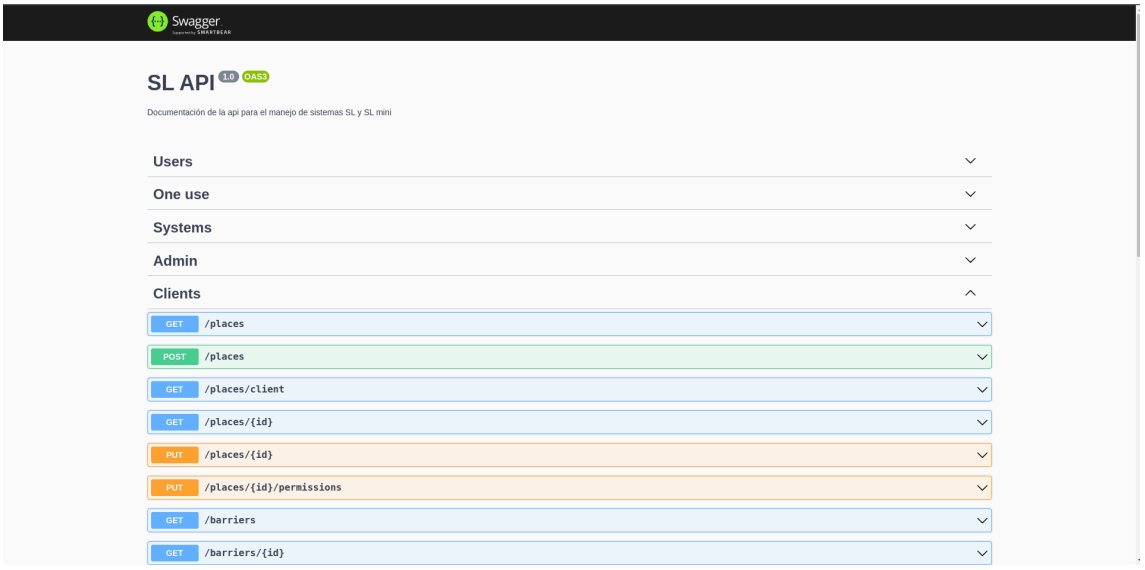


Figura 5.5: Documentación generada por Swagger.

1. Usuario y contraseña: Utilizado para los usuarios que entran desde la web.

2. Api Key de terceros: Una forma de validar si la aplicación que pide datos está autorizada o no.

3. Api Key para sistemas SL: Permite validar si la información del registro de entrada/salida está siendo enviada por un SL/SL mini autorizado. Además permite saber qué sistema es la que envía los datos, diminuyendo el tamaño de la petición.

4. Json Web Token o JWT: Se utiliza una vez que el usuario entra a la app, y es un hash [1] generado por el servidor.

5.4.3.1. Api Key

Las Api Key son llaves que generadas en servidor que permiten el accesso a determinados endpoints [2]. Estas sirven para validar las peticiones entrantes al servidor, permitiendo dar acceso a la información o denegar la consulta HTTP.

[1] Es el resultado de una operación criptográfica que genera un identificador único e irrepetible a partir de una información dada.

[2] Son las URLs que consultan desde el front-end.

5.4.3.2. JWT

JSON Web Token o simplemente JWT es un estándar qué está dentro del documento RFC 7519 [28]. Su función principal es poder propagar información entre dos partes de forma segura. En la práctica, los JWT son muy fáciles de desencriptar por lo que se desaconseja incluir información sensible del usuario en el mismo.

5.4.4. Análisis de imágenes

Este servicio fue implementado con Flask el cual permite crear aplicaciones back-end en Python de forma fácil y rápida. La función de este servicio es obtener la patente de la fotografía tomada por un SL mini, es por ello que solo tiene una ruta e internamente ejecuta el algoritmo visto en el capítulo 3.

5.4.5. Front-end

El front-end fue construido en ReactJS, que es una de las librerías de JavaScript más utilizadas en la industria. La idea principal de este servicio es brindar un centro de control de los sistemas SL, además de permitir la visualización de registros y datos. Además permite a los administradores crear nuevos sistemas SL y visualizar sus parámetros.

5.4.5.1. Páginas para los usuarios

Los usuarios puede registrarse y entrar a la plataforma como se ve en la Fig. 5.6. Luego de ingresar pueden visualizar sus estacionamientos (Fig. 5.7).

Es posible configurar diversos parámetros del estacionamiento, como el nombre, ubicación y precio por hora (Fig. 5.8). Para agregar versatilidad al sistema y permitir su uso para diferentes tareas se implementó un sistema de bloqueo de patentes (Fig. 5.9), que permite denegar el acceso a todas las patentes salvo las que estén en la lista o bloquear solo estas últimas. Adicionalmente pueden visualizar los sistemas SL por barrio (Fig. 5.10) y desde esta vista pueden configurar los parámetros de las mismas en tiempo real (Fig. 5.11).

Para la visualización y control de datos se crearon dos rutas. El Dashboard permite ver información resumida: horas estacionadas y cantidad de vehículos estacionados, separados por estacionamiento, filtrar los datos según fecha de inicio y fin, además permite agrupar los registros día, mes o año (Fig. 5.12). Por otro lado existe una vista de los registros crudos, dando información particular de cada registro (Fig. 5.13)

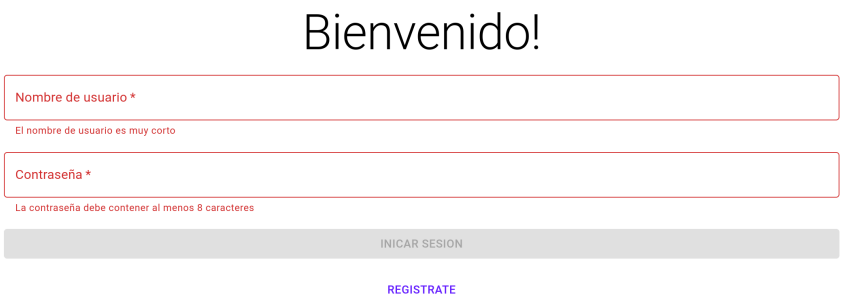


Figura 5.6: Inicio de sesión.

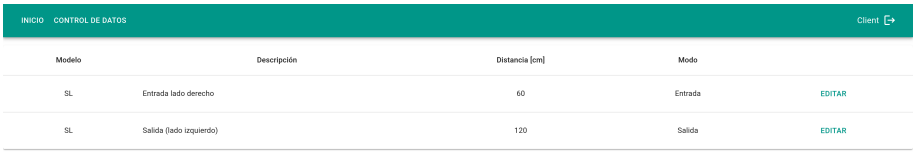


Figura 5.7: Estacionamientos del cliente.

Figura 5.8: Formulario para actualizar un estacionamiento.

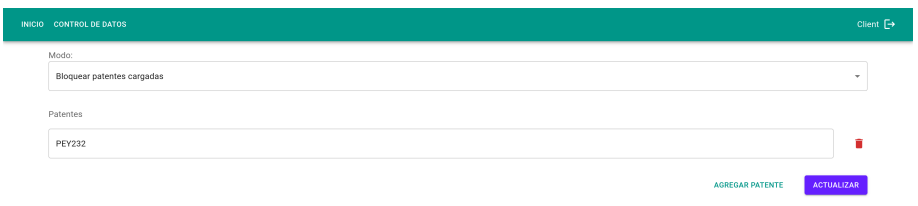


Figura 5.9: Vista de permisos.

Figura 5.10: Vista de los detalles.

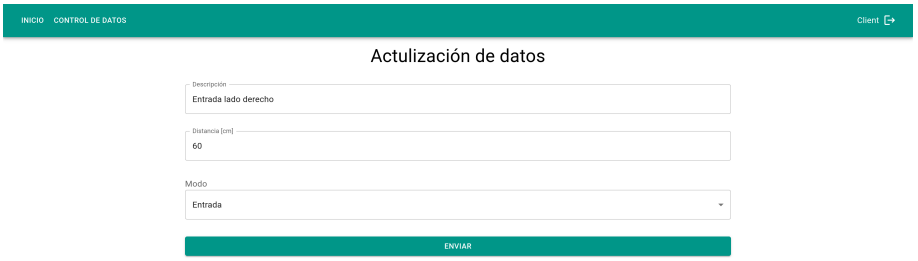


Figura 5.11: Formulario para actualizar los datos del sistema SL.

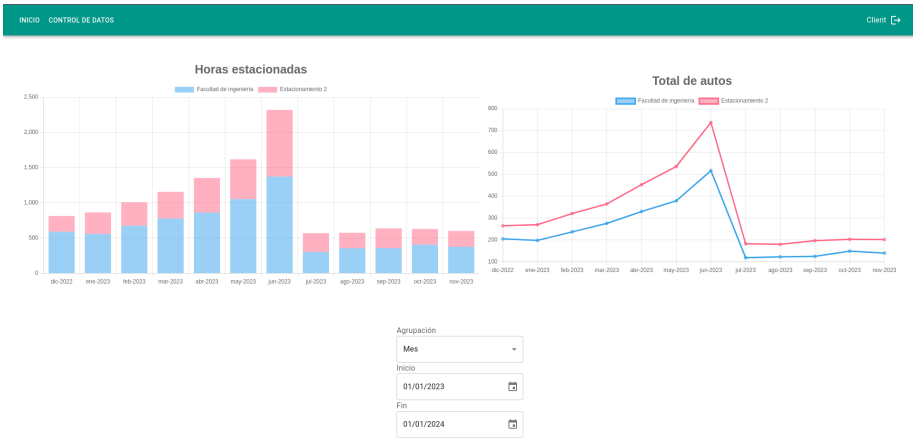


Figura 5.12: Visualización de datos para un cliente con 2 estacionamientos.

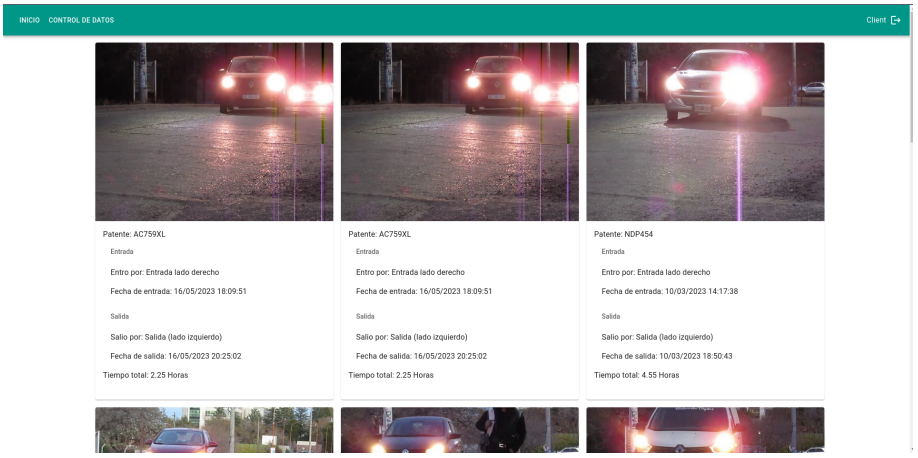


Figura 5.13: Vista de registros.

5.4.5.2. Páginas para los administradores

Los administradores poseen una página para visualizar los sistemas SL según cliente y lugar (Fig. 5.14). Por último, poseen una página para agregar sistemas SL (Fig. 5.15).

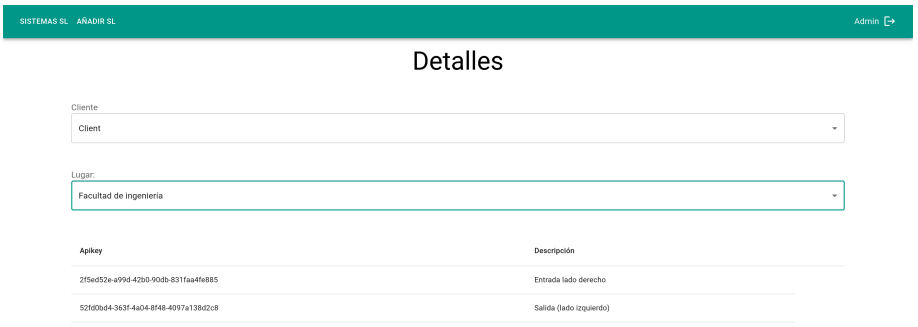


Figura 5.14: Detalles de los sistemas SL según cliente y estacionamiento.

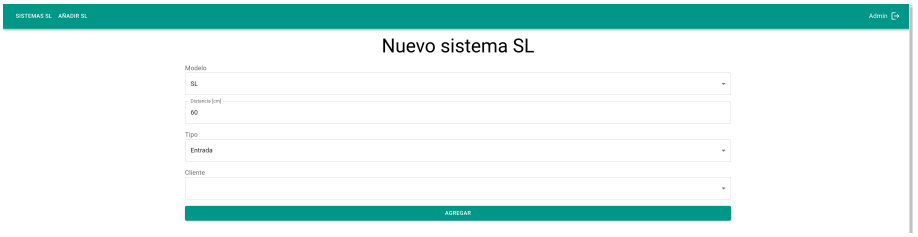


Figura 5.15: Formulario para agregar un sistema SL.

5.4.6. Nginx

Nginx es un servidor web de código abierto que permite la implementación de proxy inverso[3], cache de HTTP, y balanceo de cargo. Su implementación se utiliza como proxy inverso entre el back-end y el front-end.

[3]Es una aplicación back-end que se situa delante de otros servicios web, aumentando la seguridad, el rendimiendo y la fiabilidad del servidor.

Capítulo 6

Pruebas de integración y fiabilidad

El eje fundamental de este capítulo es mostrar el rendimiento del algoritmo y la integración final en diferentes circunstancias, y probar que el mismo es capaz de desenvolverse de forma satisfactoria en la mayoría de escenarios posibles.

6.1. Pruebas sobre el algoritmo de OCR

Para corroborar el funcionamiento del algoritmo se ensayaron diversas pruebas, con la finalidad de testear el comportamiento del algoritmo sobre diferentes entornos de trabajo, partiendo desde un ambiente ideal para llegar a uno alejado de él en el cual no se puede garantizar el correcto funcionamiento.

6.1.1. Pruebas de distancia y ángulo

Para realizar esta serie de pruebas se colocó un Volkswagen Up modelo 2015 con patente PEY232 en una posición fija y se procedió a medir con una cinta métrica de error 1 cm. Para las mediciones se consideró que la cámara y la patente están paralelas y enfrentadas. Este escenario define el ángulo de 0° y cambia solamente cuando la cámara gira en dirección paralela al suelo, mientras que el origen se tomó desde el centro de la misma. En Fig. 6.1 se puede apreciar un esquema del sistema de ejes. La medición consistió en la captura de una fotografía con la cámara de los sistemas SL a una distancia y ángulo determinados. Luego, cada medición se procesó

Distancia	Predicción
50	PEY232
100	PEY232
150	PEY232
200	PEY232
250	PEY232
300	PEY232

Tabla 6.1: Resumen de la prueba de distancia a 0° grados.

Ángulo	Predicción
-60	PEY232
-30	PEY232
0	PEY232
30	PEY232
60	PEY232

Tabla 6.2: Resumen de la prueba de ángulos a 50 cm.

con el algoritmo de OCR. Debido al espacio con el que se contaba, se realizaron dos pruebas: la primera consistió en fijar el angulo en 0° e ir tomando muestras cada 50cm, mientras que en la segunda se fijó la distancia en 50 cm y se varió el ángulo cada 30°. Las muestras obtenidas para la primera prueba pueden verse en la Fig. 6.2 y los resultados obtenidos en la Tab. 6.1, mientras que para la segunda prueba las muestras se observan en la Fig. 6.3 y los resultados en la Tab. 6.2.

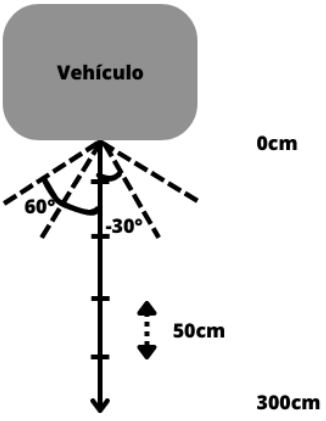


Figura 6.1: Sistema de referencia.

(a) 50cm (b) 100cm (c) 150cm

(d) 200cm (e) 250cm (f) 300cm

Figura 6.2: Fotografías a diferentes distancias a 0°.

(a) $-60°$ (b) $-30°$ (c) 0° (d) 30° (e) 60°

Figura 6.3: Fotografías a diferentes ángulos a $50cm$.

6.1.2. Pruebas de exterior

Esta prueba se realizó con una cámara Kanon PowerShot SX150 IS [29] en el ingreso al estacionamiento de la Facultad de Ingeniería de la Universidad Nacional del Comahue entre las 8 y 9 de la mañana, el martes 6 de junio del 2023. La prueba se realizó colocando la cámara en un trípode con una altura de 70cm respecto del suelo[1], y se procedió a filmar los vehículos que entraban al estacionamiento o circulaban por la calle interna Libres Pensadores de la universidad. De este proceso de medición se obtuvieron dos videos de una duración aproximada de 40min.

Teniendo en cuenta que el algoritmo de OCR tarda 1200ms en procesar un frame es necesario realizar un filtrado, ya que los 40min de filmación equivalen en 72000 imágenes, y la mayoría no poseen vehículos o la patente no está enfocada. El proceso de filtrado consistió en un recorte manual de las partes del video donde no había ningún vehículo, lo que terminó dejando 7min de filmación. Si bien este número es bajo, llevó a obtener un total de 35 vehículos diferentes en distintos frames,

[1]Se consideran los 60cm del tripode y los 10cm de la calzada, debido a que no era conveniente colocar la cámara sobre la calle.

Caracteres	Perfectas	1 error	2 errores	3 errores	4 errores	5 errores	$\sum$
6	60	78	25	10	3	0	176
7	282	185	49	5	1	1	523
$\sum$	342	263	74	15	4	1	699

Tabla 6.3: Resumen de las patentes reconocidas.

suficientes para el análisis. Posteriormente se convirtió el video en frames, utilizando Open CV, para luego procesar las 11000 fotografías y eliminar en las que no era posible detectar la forma de una patente por un ser humano. Luego se obtuvieron 3600 fotografías, que se procesaron por el algoritmo de OCR y se descartaron todas las imágenes con porcentaje de confianza inferior al 50 %. La confianza de la red sobre una patente estimada se define como el promedio de las probabilidades para cada carácter.

Las imágenes se clasificaron en una carpeta con el valor predicho por el algoritmo. Finalmente se contabilizaron los errores generando el resumen de Tab. 6.3.

6.1.3. Análisis de resultados

En base a las pruebas de interior, se logra apreciar que el rendimiento del algoritmo de OCR resulta satisfactorio en un entorno ideal controlado, es decir en un escenario de vehículo detenido, corta distancia y un angulo entre $-60°$ y $60°$. Por otro lado, en la prueba de exterior en la cual el escenario presenta condiciones extremas y no favorables, como velocidad no nula, distancia variable (corta, media y larga distancia) y ángulos fuera de rango, los resultados obtenidos cuentan con una baja eficiencia.

Realizando un análisis más exhaustivo de la prueba de exterior es posible obtener que la eficiencia del sistema ronda el 49 %. En la Tab. 6.3 se observa que para patentes de seis caracteres la eficiencia ronda el 34 % esto se puede deber a que el dataset de entrenamiento contaba con una mayor tasa de patentes de siete caracteres o a que el estado de las patentes de seis caracteres al ser de mayor antigüedad se encuentran en peores condiciones lo que dificulta su estimación. Por otro lado cuando se trata de patentes de siete caracteres estas poseen una eficiencia cercana al 54 %, valor que

se considera aceptable para esta prueba. Se observa que para ambos casos la mayor tasa de error se da cuando existe un único carácter erróneo independientemente de su posición. Finalmente, es importante recordar que estos resultados se obtuvieron para un escenario poco favorable como los ejemplificados en la Fig. 6.4. Si se extrapolan a un escenario menos hostil (vehículo cercano a la cámara y velocidad baja), los errores en la Tab. 6.3 disminuirán considerablemente.

Figura 6.4: Ejemplificación de las imágenes obtenidas durante la prueba de exterior.

En base a estas pruebas se define que los limites para las condiciones de trabajo son:

1. Distancia entre 50cm y 2m.

Figura 6.5: Vistas del SL mini y servidor junto con el vehículo de durante las pruebas de integración.

2. Ángulo entre $-60°$ y $60°$.

3. Velocidad de circulación paso de hombre.

Los valores de distancia se toman teniendo en cuenta que el largo de un auto ronda los 4m [30], es por ello que se considera una distancia máxima de detección de 2m.

6.2. Pruebas sobre el servidor

6.2.1. Integración de los sistemas SL y servidor

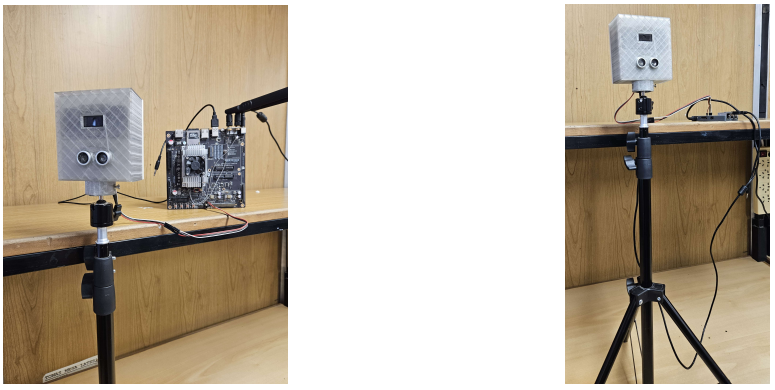
Se montó un sistema SL mini junto con una notebook como servidor, el sistema montado se observa en la Fig. 6.5, y se crearon registros de entrada, y cambio la configuración de un sistema SL en tiempo real. Con esta prueba se logró probar los cambios de configuración en tiempo real y que los registros se creen de forma correcta.

6.2.2. Creación de registros

Para probar el correcto de funcionamiento del servidor, se creó un usuario de admin y un cliente. Se crearon dos lugares junto con un script en Python 3 emulando el envío de datos de las barreras, generando en el servidor 5900 registros para los lugares, probando el funcionamiento del sistema de inicio a fin. Los resultados de estas pruebas pueden observarse Fig. 5.12, algunos de los registros creados pueden apreciarse en la Fig. 5.13.

6.3. Estado del prototipo

En la actualidad se cuentan con 2 prototipos funcionales, uno para la versión SL y otro para el modelo SL mini. El prototipo SL se encuentra montado sobre la placa Nvidia Jetson TX1 en su kit de desarrollo, la cual es capaz de correr el algoritmo de forma local. Mientras que la versión SL mini se ejecuta en la placa Raspberry Pi 3 b+, que como se hizo mención en anteriores capítulos no es capaz de correr el algoritmo de OCR por lo que esta tarea, se ejecuta en el servidor de prueba. El mismo se encuentra alojado en una notebook de la marca CX, la cual cuenta con un procesador Intel i5-1135G7, con 32 GB de ram, sin gpu dedicada, con lo que el tiempo de procesamiento por imagen es de 1,2s, como se mencionó en la Tab. 3.1. Para ambos modelos se cuenta con la misma cámara usb genérica y el sensor de distancia ultrasónico US-100. Ambos prototipos se pueden apreciar en la Fig. 6.6.



(a) Prototipo SL. (b) Prototipo SL mini.

Figura 6.6: Prototipos en su estado actual.

Capítulo 7

Conclusiones

En este capítulo se realizarán las conclusiones generales correspondientes al uso de redes neuronales convolucionales para la de detección de caracteres, más específicamente para detectar patentes argentinas, la elección de placas y sensores para el diseño del prototipo y el funcionamiento de la plataforma web. Además se detallarán las ventajas y desventajas encontradas en los algoritmos, para finalmente presentar posibles mejoras a futuro.

7.1. Conclusiones generales

Durante el proceso de investigación sobre OCR y sus utilidades, se observó que en nuestro país este tipo de tecnologías son poco explotadas, con aplicaciones escasas, limitándose al uso de herramientas creadas por terceros. Por lo que el diseño de este prototipo representa una innovación en lo que a control de acceso de estacionamientos respecta, permitiendo una automatización y control del acceso de los vehículos al recinto con una interacción nula por parte del usuario.

Se ha estudiado en profundidad el funcionamiento de redes neuronales, más específicamente las CNN y se observó que estas son las más favorables para la implementación en sistemas embebidos, por su bajo costo computacional en comparación con las LSTM, logrando reconocer las patentes de manera efectiva.

Se logró crear desde cero un prototipo funcional para resolver la tarea propuesta utilizando protocolos habituales en la industria. La elección de sensor fue una tarea

compleja debido a la situación actual de los regímenes aduaneros de nuestro país, lo que dificultó el acceso a algunos de ellos.

Uno de los enfoques más importantes de este trabajo fue el diseño modular, buscando que la interdependencia de las partes del sistema sea lo más baja posible, es decir, permitir cambiar partes del sistema disminuyendo los errores al integrar estos nuevos elementos. Es posible cambiar el servicio de detección de patentes por uno mejor, sin necesidad de cambiar el código del back-end o del front-end, respetando solo que la comunicación entre el back-end y este servicio sea por HTTP y devuelva el JSON esperado. A su vez este diseño permite cambiar el sensor de distancia o la cámara sin tener un impacto considerable en el resto del sistema. En síntesis este tipo de diseño permite una mayor flexibilidad a la hora de integrar nuevas tecnologías.

En general las tareas de diseño y desarrollo son complejas, sobre todo a la hora de brindar soluciones innovadoras. Sin embargo se logró desarrollar un prototipo funcional para permitir el acceso de vehículos a un estacionamiento sin interacción de los usuarios.

7.2. Posibles mejoras

Finalmente se presentan posibles mejoras para el prototipo presentado a lo largo de este trabajo:

- Implementar un nuevo algoritmo para la detección de patentes en otro tipo de vehículos (motos, camiones).
- Mejorar la eficiencia del algoritmo de OCR.
- Desarrollar e implementar de algoritmo de reconocimiento de patentes extranjeras.
- Reducir el costo de ensamble de los sistemas SL.
- Sustituir la cámara por una que permita mejores fotografías en ambientes externos.
- Integrar una pasarela de pagos para efectuar el cobro del estacionamiento.

- Desarrollar una versión con display para mejorar la experiencia de los usuarios.

Bibliografía

[1] Asociación de Fábricas Argentinas de Componentes. "Flota circulante en Argentina 2021". es. En: (mar. de 2021).

[2] Adriano Calalesina. *La imposible misión de buscar estacionamiento en el microcentro neuquino*. URL: https://www.lmneuquen.com/la-imposible-mision-buscar-estacionamiento-el-microcentro-neuquino-n923626 (visitado 19-05-2023).

[3] 20minutos. *La falta de aparcamiento irrumpe entre los principales problemas de los madrileños*. es. Section: Madrid. Feb. de 2018. URL: https://www.20minutos.es/noticia/3269456/0/falta-aparcamiento-irrumpe-principales-problemas-madrilenos/ (visitado 19-05-2023).

[4] Hossam El-Din Ibrahim. "Car Parking Problem in Urban Areas, Causes and Solutions". en. En: *SSRN Electronic Journal* (2017). ISSN: 1556-5068. DOI: 10.2139/ssrn.3163473. URL: https://www.ssrn.com/abstract=3163473 (visitado 19-05-2023).

[5] Ana Inés Basco et al. *Industria 4.0: Fabricando el Futuro*. es. Google-Books-ID: geiGDwAAQBAJ. Inter-American Development Bank, jul. de 2018.

[6] Daniel Rivera Arroyave et al. *SmartParkUdeA: Sistema IoT para el estacionamiento inteligente de vehículos en ciudad universitaria*. Feb. de 2021.

[7] Agustín Formoso, Agustín Mazzilli y Rafael Sotelo. "ParkIt - Plataforma inteligente de estacionamiento público". es. En: *Memoria Investigaciones en Ingeniería* 12 (nov. de 2014). Number: 12, págs. 85-94. ISSN: 2301-1106. URL: http://revistas.um.edu.uy/index.php/ingenieria/article/view/351 (visitado 19-05-2023).

[8] Ankandrew. *Reconocedor de Texto(OCR) para Patentes vehiculares de Argentina*. original-date: 2020-08-07T00:46:17Z. Mayo de 2023. URL: https://github.com/ankandrew/cnn-ocr-lp (visitado 25-06-2023).

[9] IBM. *¿Qué es Deep Learning?* es-es. URL: https://www.ibm.com/es-es/topics/deep-learning (visitado 25-06-2023).

[10] Rubén Rodríguez Abril. *Redes Convolucionales*. es. Mar. de 2021. URL: https://lamaquinaoraculo.com/deep-learning/redes-neuronales-convolucionales/ (visitado 07-08-2023).

[11] Alexey Bochkovskiy, Chien-Yao Wang y Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv:2004.10934 [cs, eess]. Abr. de 2020. URL: http://arxiv.org/abs/2004.10934 (visitado 25-06-2023).

[12] Alexey. *Yolo v4, v3 and v2 for Windows and Linux*. original-date: 2016-12-02T11:14:00Z. Jun. de 2023. URL: https://github.com/AlexeyAB/darknet (visitado 25-06-2023).

[13] OpenCV. *OpenCV*. original-date: 2012-07-19T09:40:17Z. Jun. de 2023. URL: https://github.com/opencv/opencv (visitado 25-06-2023).

[14] CASADOMO. *Sistema de apertura automatizada de barrera que identifica vehículos*. es. Jul. de 2015. URL: https://www.casadomo.com/2015/07/01/sistema-de-apertura-automatizada-de-barrera-que-identifica-vehiculos (visitado 25-06-2023).

[15] Integralia. *Un sistema para controlar los accesos a recintos: Barreras*. es. Mayo de 2019. URL: https://www.aradock.es/nuevos-sistemas-de-control-de-accesos-barreras/ (visitado 25-06-2023).

[16] *Acceso sin contacto a barreras mediante RFID — SICK*. URL: https://www.sick.com/mx/es/sectores/seguridad-de-los-edificios/seguridad-en-los-exteriores/control-de-acceso/acceso-sin-contacto-a-barreras-mediante-rfid/c/p360565 (visitado 25-06-2023).

[17] *Documentación Raspberry Pi*. en. URL: https://www.raspberrypi.com/documentation/ (visitado 25-06-2023).

[18] NVIDIA. “Manual de usuario Jetson TX1”. en. En: ().

[19] *Documentación PySerial*. URL: https://pyserial.readthedocs.io/en/latest/pyserial.html (visitado 25-06-2023).

[20] Ian Craggs. *Documentación Paho MQTT*. en. URL: https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php (visitado 25-06-2023).

[21] Python Software Foundation. *Documentación Requests*. URL: https://requests.readthedocs.io/en/latest/ (visitado 25-06-2023).

[22] Google. *TensorFlow*. URL: https://www.tensorflow.org/?hl=es-419 (visitado 25-06-2023).

[23] Mati Presta. *Los 10 principales lenguajes de programación de backend*. es-ES. Feb. de 2021. URL: https://blog.back4app.com/es/lenguajes-de-programacion-de-backend/ (visitado 19-05-2023).

[24] Keepcoding. *¿Qué es el Kernel?* es. Section: Blog. Mar. de 2022. URL: https://keepcoding.io/blog/que-es-el-kernel/ (visitado 19-05-2023).

[25] *Kubernetes*. URL: https://kubernetes.io/es/ (visitado 12-07-2023).

[26] Docker Inc. *Docker Compose*. en. Mayo de 2023. URL: https://docs.docker.com/compose/ (visitado 05-05-2023).

[27] *Documentación de NestJS*. en. URL: https://docs.nestjs.com (visitado 05-05-2023).

[28] Michael B. Jones, John Bradley y Nat Sakimura. *JSON Web Token (JWT)*. Issue: 7519 Num Pages: 30 Series: Request for Comments Published: RFC 7519. Mayo de 2015. DOI: 10.17487/RFC7519. URL: https://www.rfc-editor.org/info/rfc7519.

[29] Kanon. *PowerShot SX150 IS*. URL: https://www.cla.canon.com/cla/es/support/consumer/cameras/powershot_cameras/powershot_sx150_is (visitado 24-07-2023).

[30] Oscar Duran. *¿Qué tan grande es el promedio de los autos? Descubre las medidas aquí — Actualizado agosto 2023*. es. Mar. de 2023. URL: https://hiferauto.es/autos/medidas-de-un-auto-promedio/ (visitado 07-08-2023).