Universidad Internacional de La Rioja (UNIR)

ESIT

Máster universitario en Dirección e Ingeniería de Sitios Web

RAMA PROFESIONAL

Arquitectura de software para servicios IoT en la Universidad Nacional de Educación

Trabajo Fin de Máster
presentado por: Jones Ortiz, Carlos
Director/a: Fernández Peña, Félix Oscar.

Ciudad: Guayaquil
Fecha: 23/07/2020

Resumen

En este trabajo de fin de máster se busca establecer conceptos y metodologías básicas para definir una arquitectura de software con una estructura eficiente, que permita una comunicación estable con tecnologías del internet de las cosas, y mantenga niveles de seguridad adecuados. Se estudian técnicas y tecnologías aplicadas a la captura, procesamiento y presentación de datos. Para la validación experimental de esta arquitectura se propone la implementación de un sistema para la detección y reconocimiento de matrículas de los vehículos que ingresan a la Universidad Nacional de Educación, con herramientas de software libre como Open CV y Python. Esta aplicación, como caso de estudio, permitió validar la utilidad de la arquitectura de software propuesta para la implementación de sistemas basados en tecnología del internet de las cosas.

Palabras Clave: Arquitectura de software, Internet de las cosas, Open CV, detección, identificación.

Abstract

This master's thesis work seeks to stablish basic methodologies and concepts in order to define a software architecture with an efficient structure that allows stable communication with internet of things technologies, and maintains appropriate security levels. Techniques and technologies applied to the capture, processing and data presentation are studied. For the experimental validation of this architecture, we propose the implementation of a system for the detection and recognition of license plates of vehicles entering to the Universidad Nacional del Ecuador with free software tools such as Open CV and Python. This application, as a case study, allowed to validate the utility of the software architecture proposed for the implementation of system based on internet of things technologies.

Keywords: Software architecture, Internet of Things, Open CV, Detection, Identification.

Índice de Contenido:

Índice de figuras:

Índice de tablas:

1. Introducción

La evolución tecnológica que se evidencia permite que sea posible la automatización o digitalización de actividades cotidianas de los seres humanos, herramientas como la comunicación inalámbrica apoyado en técnicas como el Internet de las Cosas (IoT, del inglés *Internet of Things*) proporcionan un conjunto de alternativas eficientes y fáciles de integrar para solventar problemas como el antes mencionado.

Los sistemas de información web, también se han visto mejorados con el pasar del tiempo y la evolución de las Tecnologías de la Información y Comunicación (TIC), las arquitecturas de software han sido piezas fundamentales en el desarrollo de sistemas integrales y capaces de interactuar con otros sistemas.

La interoperabilidad e interconexión de sistemas y plataformas ha sido fundamental para el desarrollo y proliferación de ecosistemas inteligentes, la automatización de hogares e instituciones educativas es tema cada vez más común en proyectos de investigación. Los problemas como el tráfico y la movilidad pueden ser solventados mediante soluciones IoT que integren otros servicios como la computación en la nube o BigData para el tratamiento de grandes cantidades de información (Bruneo, et al., 2019). Como especifican los autores en su artículo, una de las principales motivaciones para este tipo de proyectos es alcanzar ecosistemas homogéneos donde múltiples aplicaciones y plataformas puedan interactuar en función de un objetivo común.

Uno de los principales desafíos para la adopción a gran escala del IoT es alcanzar la heterogeneidad. Verdouw, Sudmaeker, Tekinerdogan, Conzon, & Montanaro (2019) propusieron un marco de arquitectura para modelar sistemas basado en el Internet de las Cosas desde diversos puntos de vistas arquitectónicos; los autores validan este marco de trabajo en un estudio de caso enfocado al ámbito de la agricultura y alimentación.

El Internet de las Cosas maneja un amplio espectro de aplicación y acción; es por ello que la definición de estructuras y arquitecturas toma relevancia en el sentido de proporcionar guías y preceptos definidos para interconexión de dispositivos de control, percepción o procesamiento. A diferencia del internet tradicional, el Internet de las Cosas integra tanto elementos físicos como lógicos, característica que hace que el diseño e implementación de este tipo de sistemas se enfrenten a grandes desafíos (Guang, Tonghai, Meng, Xinyu, & Wenfei, 2020). Una arquitectura de software es la responsable de definir métodos para la segmentación eficiente de un sistema y cómo sus componentes son identificados y se comunican entre sí.

En la Universidad Nacional de Educación existen varios procesos que incluyen y necesitan del funcionamiento conjunto de diversos dispositivos; sin embargo, no existe una arquitectura de software que sirva como referencia para la integración de dichos elementos de una manera eficiente

Este trabajo final de máster tiene como tarea fundamental desarrollar una arquitectura web basada en tecnología IoT que permita la gestión e integración de tareas y procesos que se ejecutan en la Universidad Nacional de Educación.

En el capítulo 2, se aborda la temática relacionada a los servicios y arquitectura de software basadas en el Internet de las Cosas, modelos de referencia y sus principales casos de uso; además de la utilización de herramientas libres como Python y Open CV.

En el capítulo 3 se plantean los objetivos general y específicos, además de la metodología utilizada para alcanzar los mismos; detallando cada uno los pasos y tareas realizadas a lo largo de la investigación.

En el capítulo 4 se identifican las necesidades y los requisitos para la implementación y se especifica la arquitectura con cada una de sus capas. Se propone, como caso de estudio, el desarrollo de un sistema que detecte e identifique las matrículas de los vehículos que intentan ingresar al parqueadero de la universidad que, informe de manera automática al personal encargado si el auto en cuestión tiene o no el acceso correspondiente.

En el capítulo 5 se realiza la validación de la arquitectura propuesta mediante la implementación del sistema de reconocimiento de matrículas, realizando diferentes pruebas para contrastar los resultados posteriores a la automatización.

En el capítulo 6 se realiza la descripción de los resultados, la discusión y conclusiones de la investigación realizada.

2. Contexto y estado del arte

2.1. Servicio web

La evolución tecnológica actual ha provocado que los entornos web, ya no sean simples páginas estáticas encargadas de mostrar información; hoy en día, en la web, se puede encontrar verdaderos sistemas informáticos que permiten la gestión de diferentes tareas.

Este crecimiento, también ha generado la necesidad de que exista una interacción y comunicación entre diferentes sistemas, sin importar el lenguaje en el que han sido desarrollados o el sistema operativo sobre el que operen.

En este sentido, los servicios web han aparecido como alternativa viable que, a través de estándares como XML, HTTP, SOAP, entre otros, han permitió la interacción eficaz, y segura entre sistemas. La W3C considera que un servicio web es una aplicación de software, donde mediante una URI, su interface y enlaces pueden ser definidos, descritos y descubiertos como artefactos XML (Bernardis, Bernardis, Berón, Riesco, & Pereira, 2018). Estos servicios web pueden soportar una interacción directa con otros sistemas o software, mediante protocoles de internet.

El desarrollo y proliferación del uso de Web Services ha sido posible gracias a la evolución de tecnologías, lenguajes y estándares que permiten su definición, quizá, uno de los estándares más conocidos en el Web Services Definition Language (WSDL), cuyas especificaciones son un dialecto XML, que define reglas claras en cada componente del servicio web.

La utilización de servicios web ha sufrido un aumento considerable, convirtiéndose en una de las implementaciones más utilizadas hoy en día, es posible que uno de los aspectos más importantes en el desarrollo de sistema web, es la elección de las herramientas adecuadas, ya que es probable que esto influya en los requisitos finales (Sayago, Flores, & Recalde, 2019).

Los servicios web han ganado espacio en el mercado debido a los beneficios que presentan a la hora del desarrollo de aplicaciones distribuidas, son componentes de software capaces de ejecutarse independientemente del lenguaje o plataforma que lo provee. (León, Boixader, & Luque, 2014).

Definen un modelo de interacción sistema a sistema, la W3C, aporta información relevante sobre la estructura de una arquitectura de software donde se observa que el modelo orientado a servicios usa metadatos, que son una propiedad clave en estas

arquitecturas, se utilizan para documentar aspectos del servicio como detalles de la interfaz y el enlace de transporte (W3C, 2020).

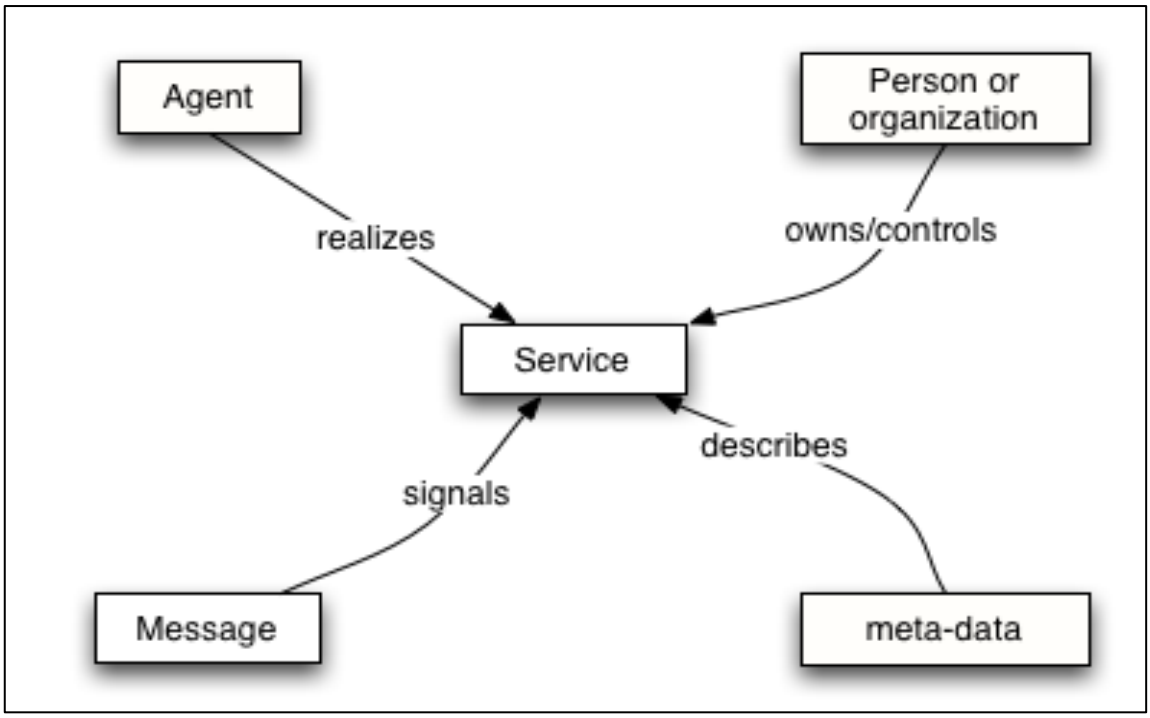


Figura 1. Modelo básico de arquitectura orientada a servicios. (W3C, 2020)

Como se evidencia en la Figura 2, en el modelo en cuestión, un agente realiza o proporciona el servicio y otro lo consume, los mismos están mediados por los mensajes que se intercambian.

Hoy en día, los servicios web se basan en dos tecnologías principales como son SOAP y REST, cada uno con sus características y ventajas según las necesidades que se tengan. Para esta propuesta, se hace uso del modelo REST ya que presenta una estructura ligera, sencilla, con la seguridad y rendimiento adecuado. La posibilidad de enviar datos en formato JSON y su compatibilidad con las herramientas existentes en la Universidad Nacional de Educación vuelven a este modelo la solución ideal para la tarea de compartir información.

2.2. Arquitectura de software

Una arquitectura de software puede tener diferentes connotaciones, comúnmente hacen referencia a la jerarquización de la información en un entorno web; sin embargo, también pueden especificar formatos de comunicación entre sistemas o servicios en la web.

Generalmente una arquitectura se divide en capas para segmentar de manera adecuada sus componentes, generalmente adoptan un patrón modelo vista-controlador-comunicación (Mendoza, Ordóñez, Ordóñez, & Jurado, 2017). Estas capas permiten que los componentes de hardware y software interactúen; presenta un nivel de abstracción que da sentido tanto a las funciones lógicas como a las tareas que ejecutan los elementos de hardware.

Las arquitecturas de software que sirven de referencia para la integración de servicio IoT deben cumplir una serie de principios como permitir la escalabilidad, el soporte de nuevos estándares, servir de modelo para el desarrollo de arquitecturas más específicas (Kurebwa & Mushiri, 2019). IoT es una tecnología con un amplio espectro, por la arquitectura en la que se basa el servicio no debe ser rígida, por el contrario, su estructura debe ser dinámica para que pueda ser aplicada a diferentes ámbitos.

Al no existir un modelo de referencia generalizado y claramente definido, diferentes investigaciones ya artículos científicos se han enfocado en proponer arquitecturas que sirvan de referencia para integración de diversos servicios. Hadj, Ghozzi, & Chaari (2019), proponen “Una nueva arquitectura para Internet de las Cosas y Big Data”, manifiestan que el Big Data e Internet de las Cosas se consideran los principales paradigmas al definir una arquitectura de integración de servicios evidenciando un valor agregado al combinar Data WareHouse y Data Lake.

Los sistemas de reconocimiento óptico de caracteres (OCR) también son un campo donde el desarrollo e integración de arquitecturas IoT ha tomado relevancia. Kiran, Ganta, Krishna & Praveen (2020) proponen un método para el reconocimiento de matrículas de vehículos mediante técnicas de procesamiento de imágenes. La arquitectura que proponen se basa en la extracción, procesamiento y detección de la imagen.

Una arquitectura de software ya sea simple o compleja debe describir de manera clara los componentes del sistema y sus interacciones (Verdouw, Sudmaeker, Tekinerdogan, Conzon, & Montanaro, 2019). Los autores afirman que estas arquitecturas no deben ser representadas en diagramas, más bien deben ser expuestas en vistas que representen los elementos desde una perspectiva de negocio o implementación.

La constante evolución del internet ha agilizado el acceso e intercambio de información, con tecnologías flexibles, de libre acceso y con la capacidad de integrar recursos y servicios. En la actualidad, el internet tiene como objetivo el propiciar la colaboración y el trabajo comunitario.

Existen diversos estilos arquitectónicos que se enfocan en la comunicación e interacción de diferentes componentes. Es importante que la arquitectura propuesta no solamente se

enfoque en la comunicación entre el servicio y el cliente; es necesario que la misma abarque todos los aspectos del proceso de percepción y procesamiento de información.

2.2.1. Arquitectura Lambda

Es una de los principales estándares que hace referencia a protector IoT, la misma consta de cuatro componentes principales:

- Captura de Datos
- Capa de procesamiento
- Almacenamiento
- Capa de consulta

Esta arquitectura equilibra la latencia y el rendimiento, lo que la vuelve efectiva en cuanto a la tolerancia a errores al procesar por lotes y proporcionar vistas precisas de los datos.

Si bien esta arquitectura ha sido pensada y mayormente utilizada en análisis de Big Data, su objetivo principal es el tratamiento masivo de datos, lo que no escapa de la meta propuesta en la investigación por lo que su estructura y capas, puede ser plenamente adaptable.

Lambda, engloba tecnologías y dispositivos que obtienen datos en bruto como sensores o en este caso una cámara IP; en cuanto al procesamiento, esta arquitectura lleva a cabo servicios preparación, adecuación, tratamiento e integración de datos para comunicar información adecuada a la siguiente capa.

En la capa de almacenamiento se pueden integrar bases de datos, por su parte en el nivel de aplicación se desarrollarían los servicios completos en función d la información detectada y formalizada.

En la actualidad, en sistemas de alto nivel se pueden distinguir dos tipos de procesamiento de datos, en modo batch o en modo stream (tiempo semi real). El procesamiento batch se encarga del tratamiento de datos en tiempos limitados por espacios que pueden ir de 15 minutos a 3 horas, o incluso diario.

Por su parte, el modo stream, tiene la capacidad de procesar datos casi en simultáneo con el momento en que estos se producen.

Esta diferencia es fundamental para la propuesta planteada, ya que en la misma es necesario que la información obtenida en la captura de datos sea validada al instante y así determinar si el usuario tiene acceso al parqueadero.

El objetivo de plantear una arquitectura basada en Lambda, es implementar un sistema capaz de combinar las dos técnicas de procesamiento de datos, para esta manera alcanzar los mejores resultados posibles. El modelo batch brinda un mayor alcance y análisis a profundidad, por su parte el modo stream presenta información en línea para una rápida toma de decisiones.

2.2.2. Arquitecturas de referencia para el internet de las cosas (IoT ARM)

Es un modelo arquitectónico de referencia pensado por un grupo de investigadores y científicos especialistas en el área, su principal objetivo es proporcionar fundamentos arquitectónicos enfocados al Internet de las Cosas, de manera que se logre una interacción sencilla entre diferentes tecnologías y sistemas.

Esta metodología, para el establecimiento de una arquitectura, propone la realización de diseño de un sistema partiendo de diferentes actividades que finalizan con la definición de vistas. Se identifican cuatro actividades principales:

- Crear una Vista de Entidades Físicas
- Crear una Vista de Contexto de Internet de las Cosas
- Proceso de requerimientos
- Derivar vistas (Funcionales, Información, Operación)

En el proceso de requerimientos se analizan las oportunidades y amenazas del sistema, definiendo los requisitos originales del sistema ya sean funcionales o no en base a las necesidades arquitectónicas y elección de tácticas o patrones.

En la derivación de vistas se integran otros componentes como una vista funcional, de información, operación o despliegue.

Existen otros modelos de referencia y metodologías para establecer arquitecturas funcionales basada en IoT, entre las más conocidas están:

Tabla 1. Diseños y modelos de referencia para arquitecturas IoT

Diseños y modelos de referencia para arquitecturas IoT			
Modelo	**Comercial**	**Genérico**	**Capas**
ITU		X	4
IoTWF	X		7
Modelo IEEE		X	3
Intel IoT	X		6
IoT Simple	X		5
Arquitectura de referencia IoT IBM	X		3

Elaboración propia del autor

- **Modelo ITU:**

Presenta un diseño de cuatro capas, se enfoca de manera eficiente en la gestión y seguridad; según (ITU, 2020) entre sus principales características están:

- Gestión de dispositivos.
- La posibilidad de encendido y apagado de forma remota.
- Gestión de tráfico en la red.
- Gestión de la topología de la red.

Integra en su capa de aplicación niveles de autenticación, autorización y privacidad; su capa de dispositivo realiza el control de acceso y protección de datos.

- **Modelo IoTWF:**

El IoTWF (Internet of Things World Forum) está conformado por empresas como Cisco, Rockwell Automatio o IBM.

Suárez, Ávila, & Páez (2019) afirman que este modelo se organiza en 7 niveles o capas funcionales:

- Controladores: Son todos los dispositivos físicos encargados de la generación de datos.
- Capa de conexión: Se encarga de la transmisión segura de los datos desde la capa 1 hasta la capa de procesamiento.

- Computación de borde: Se enfoca en reducir y convertir el flujo de datos en información que puede ser almacenada o tratada en los niveles superiores.
- Acumulación de datos: Es el nivel donde se almacena la información que puede ser utilizada por aplicaciones.
- Abstracción de datos: Su función es ordenar los datos provenientes de diferentes fuentes.
- Aplicación: Esta capa interpreta los datos mediante aplicaciones software, permitiendo así el control y monitoreo de la información.
- Colaboración y procesos: Es la capa controladora de la interacción que se pueda dar entre el usuario y el sistema IoT.

- **Modelo IEEE:**

Es un modelo genérico, si bien no es un estándar aprobado se presenta como un marco de referencia valido que puede tener variaciones sobre sus capas, permitiendo mejorar sus características para una implementación propia de un sistema IoT.

IEEE (2020) indica que sus capas son:

- Aplicación
- Red y Comunicación de datos
- Capa de censado

- **Intel IoT:**

Intel propone un modelo de referencia que simplifica la conectividad y seguridad en el Internet de las Cosas.

Reddy & Shareef (2018) en su publicación detallan que este modelo de referencia actúa sobre seis capas principales que son:

- Capa de comunicación y conectividad: Utiliza diferentes protocolos para la comunicación de diversos dispositivos en una red PAN, LAN o WAN.
- Capa de análisis de datos: Utiliza Edge Computing para el manejo de datos.
- Capa de gestión: Actúa sobre los dispositivos controlando sus acciones y operaciones.
- Capa de control: Integra control de acceso a dispositivos y diferentes políticas de seguridad.
- Capa de seguridad: Ofrece protección robusta en cada nivel de la arquitectura.

- Capa de aplicación: Integra una interfaz para el monitoreo de datos.

- **IoT Simple:**

Es uno de los modelos de referencia más utilizados; propone una arquitectura de cinco capas que se relacionan:

- Capa Sensor / Actuador: Son todos los dispositivos encargados de censado y captura de datos.
- Capa Gateways: Da soporte a dispositivos IoT que no tienen conexiones TCP IP.
- Capa de Red: Se encarga del control de la red empresarial.
- Capa de Análisis: Controla y administra los datos capturados.
- Capa Big Data: Encargada del almacenamiento y procesamiento de la información, se puede decir que es el Data Center del sistema.

- **Arquitectura de referencia IoT de IMB:**

Es un modelo propuesto por la empresa IBM, (Jeon & Suh, 2018) afirman que se basa en tres capas que pueden integrar otros niveles como la seguridad de datos:

- Capa Devices (Dispositivos)
- Cada Edge
- Capa cloud (Nube)

En su capa Edge Computing, se produce un control de latencia y filtrado de datos que se dirigen a la capa Cloud, mejorando el rendimiento de la computación en la Nube.

2.3. Protocolos de comunicación SOAP y tecnologías REST

Estas tecnologías no pueden ser comparadas directamente, ya que SOAP es un protocolo, mientras que REST, es más un estilo arquitectónico.

Muchos desarrolladores se han enfrascado en esta discusión, preguntándose si a la hora de implementar un API para sus sistemas, cuál de estas es más eficiente; sin embargo, no es posible decir si una es superior a otra, ya que su utilización depende netamente de las necesidades del usuario.

Es importante tener en cuenta para el protocolo de comunicación que en cuanto a aplicaciones de IoT, se basan en el Internet, por lo que es importante considerar soluciones adecuadas. Las tecnologías englobadas en los Web Services SOAP XML, en relación a IoT, son capaces de soportar aspectos relevantes par aun sistema colaborativo (García, Prieto, &

Fisteus, 2006). REST, se presenta como una alternativa viable, y ha venido ganando espacio en este ámbito. Permite una comunicación en internet de manera sencilla, ligera y natural (Molina, 2016).

Los servicios REST presentan una arquitectura de capas pensada en la escalabilidad; los clientes no son capaces de distinguir entre si le están devolviendo una respuesta o si está realizando una petición de manera directa al servidor; un ejemplo de esto, es el funcionamiento de un balanceador, que se encarga de redireccionar las peticiones.

Las API REST están basadas en el protocolo HTTP, usan códigos de respuesta HTTP para funciones que a través de una URI permite la estructuración de los recursos disponibles.

Para un sistema colaborativo, que requiera de la comunicación entre nodos, periféricos o sensores, es importante contar con protocolos ligeros, donde REST se presenta como la principal alternativa. Una de las principales ventajas de utilizar REST es la posibilidad de implementar una arquitectura cliente servidor en diferentes lenguajes, es posible enviar la información en distintos formatos y a nivel de rendimiento es mejor que los servicios SOAP gracias a si ligereza ya mencionada.

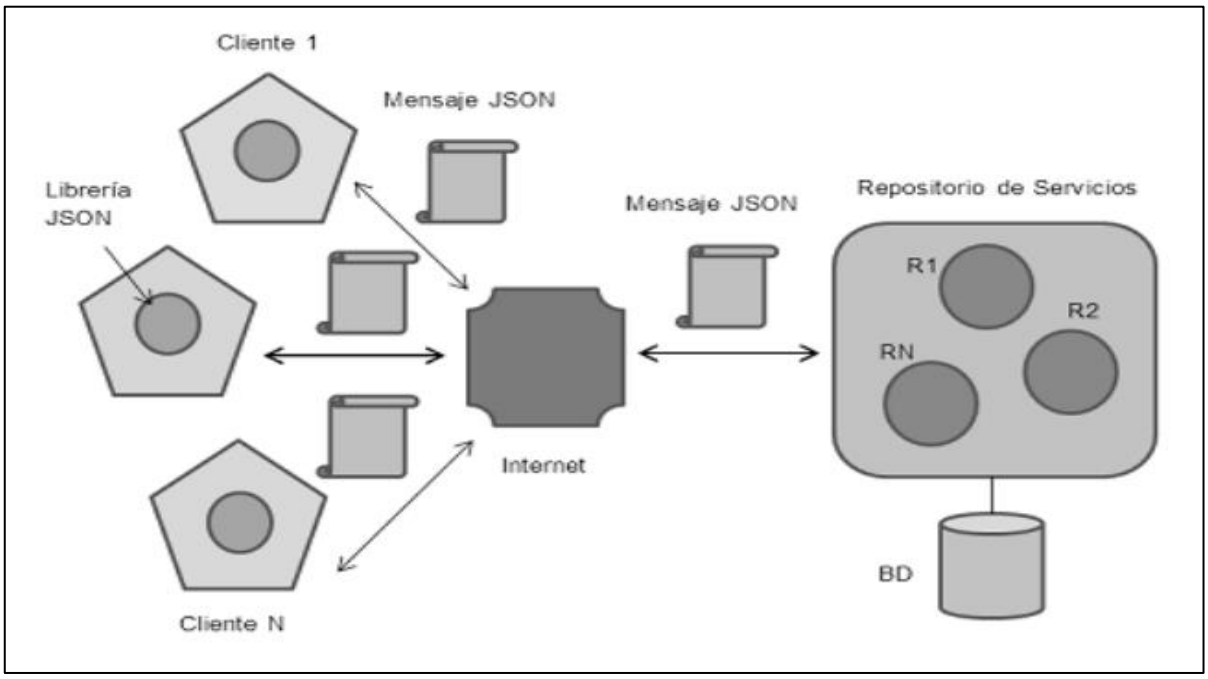


Figura 2. Esquema de consumos de servicios REST – JSON. (Campo, Chanchí, & Arciniegas, 2013, p. 6)

Como se observa en la Figura 2, el protocolo REST se plantea como una arquitectura cliente/servidor, donde el servicio es un recurso, mismo que se identifica y accede mediante

una URL. REST utiliza cuatro métodos para la comunicación, GET, PUT, DELETE y POST, para el intercambio de información, se hace uso de diferentes lenguajes como XML o JSON, siendo este último uno de los más populares en la actualidad.

JSON es un formato basado en texto, es ligero y su sintaxis se deriva de JavaScript (Chanchí, Arciniegas, & Campo, 2016), su principal característica es que al ser un formato independiente de cualquier lenguaje.

Tabla 2 Comparación entre REST y SOAP

REST	SOAP
Es un estilo arquitectónico	Es un protocolo
Puede utilizar el SOAP de servicios web, ya que al ser un concepto puede utilizar cualquier protocolo como HTTP o SOAP	No puede utilizar REST ya que es un protocolo
Bajo consumo de recursos	Aunque generalmente es fácil de usar, su complejidad puede significar n alto consumo de recursos
El cliente no necesita información de enrutamiento a partir de la URI inicial.	Los clientes necesitan saber las operaciones y su semántica antes del uso
Generalmente es fácil de construir y adoptar	Se necesitan puertos dedicados para diferentes tipos de notificaciones.

Elaboración propia del autor

Se puede llegar a la conclusión de que REST permite la creación de servicios muchos más simples y ligeros, de un uso intuitivo y sencillo y utilizable en cualquier navegador web. Los servicios que lo usan para compartir información no necesitan ser compatibles, es decir, que el emisor puede ser una aplicación basada en Java, mientras que el receptor estar desarrollado en Python.

2.4. Internet de las cosas

Puede ser definido como la interconexión entre personas u objetos que son capaces del intercambio de datos mediante una red sin la interacción directa de humano a humano o humano – ordenador.

Zito (2018), menciona que el término Internet de las Cosas, denominado IoT por sus siglas en inglés, se ha integrado en la sociedad de una manera intensa, sobre todo en los medios de comunicación. La evolución de los dispositivos electrónicos, la capacidad de

analizar grandes cantidades de datos mediante Big Data, han contribuido de manera activa a la proliferación del IoT.

El paso del tiempo ha transformado el concepto de IoT, en el 2005, e un informe publicado por la Unión Internacional de Telecomunicaciones (UIT) se hace referencia al internet de las cosas como una promesa de un mundo de dispositivos verdaderamente interconectados (Alvear, Rosero, Peluffo, & Pijal, 2017).

Es importante considerar que el IoT hace referencia a la evolución del internet, proporcionando avances sustentables en la capacidad de capturar, analizar y compartir datos que generen información. En síntesis, el Internet de las Cosas se refiere a la utilización de diferentes dispositivos para la detección de información, como elementos RFID para la identificación de radiofrecuencia, sensores, sistemas de posicionamiento global y otros.

Una de las principales características de las tecnologías IoT es la interconectividad, la misma permite el acceso a una infraestructura de información y comunicación. También proporciona servicios relacionados con objetos, cambios dinámicos o escalabilidad, que hace referencia al número de dispositivos IoT interconectados.

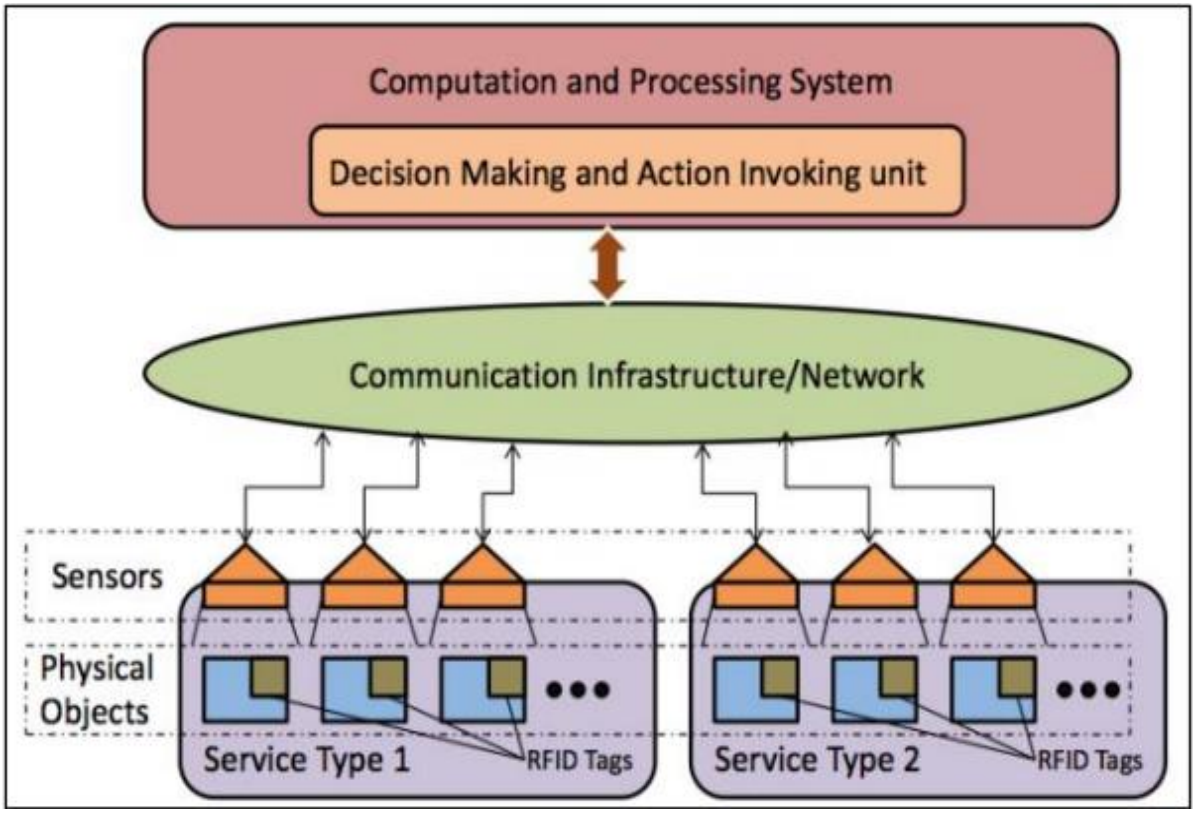


Figura 3. Modelo básico de un sistema IoT. (Cobos, 2016, p. 3)

Como se muestra en la Figura 3, una arquitectura básica de un sistema IoT, pueda estar compuesta de tres capas, aplicación, red y percepción, sin embargo, desde un punto de vista técnico, y en relación al gran desarrollo que ha tenido, se pueden considerar capas para el modelo de gestión y de negocio.

La comunicación de un sistema IoT se basa popularmente en tecnologías inalámbricas; una de las premisas fundamentales es la privacidad y confiabilidad de la información que se trasmite, esto implica autenticidad de los datos y el aseguramiento de la calidad de las mediciones que realizan los dispositivos.

El control de los dispositivos y las comunicaciones debe ser coordinada en función de la utilización efectiva de los datos recopilados, por lo que es necesario una total armonía y cooperatividad en las aplicaciones, procesos, dispositivos y servicios integrados al sistema IoT.

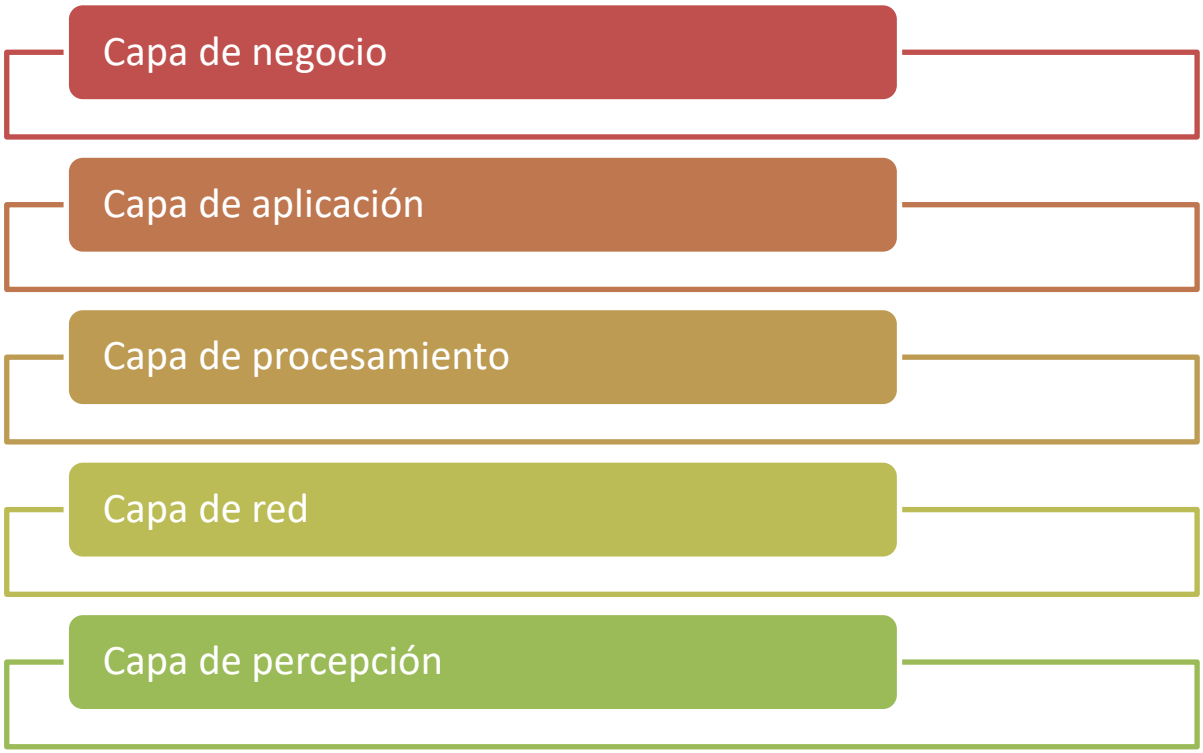


Figura 4. Modelo básico de un sistema IoT, basado en 5 capas. (Autor, 2020)

Como se muestra en la Figura 3, una arquitectura básica de un sistema IoT, pueda estar compuesta de tres capas, aplicación, red y percepción, sin embargo, desde un punto de vista técnico, y en relación al gran desarrollo que ha tenido, se pueden considerar capas para el modelo de gestión y de negocio.

La capa de percepción es la encargada de la captación de datos, puede utilizar diferentes sensores, actuadores o dispositivos, como el caso de esta propuesta, una cámara.

La capa de red tiene como objetivo principal el envío y recepción de datos, mediante protocolos basados en 3G, 4G, Wireless, Bluetooth, ZigBee u otras tecnologías (Cobos, 2016). Si se integra una capa de procesamiento, la misma interactúa a partir de la información obtenida, se transmite a través de la red hacia dispositivos o sistemas inteligentes que procesan los datos y ejecutan acciones.

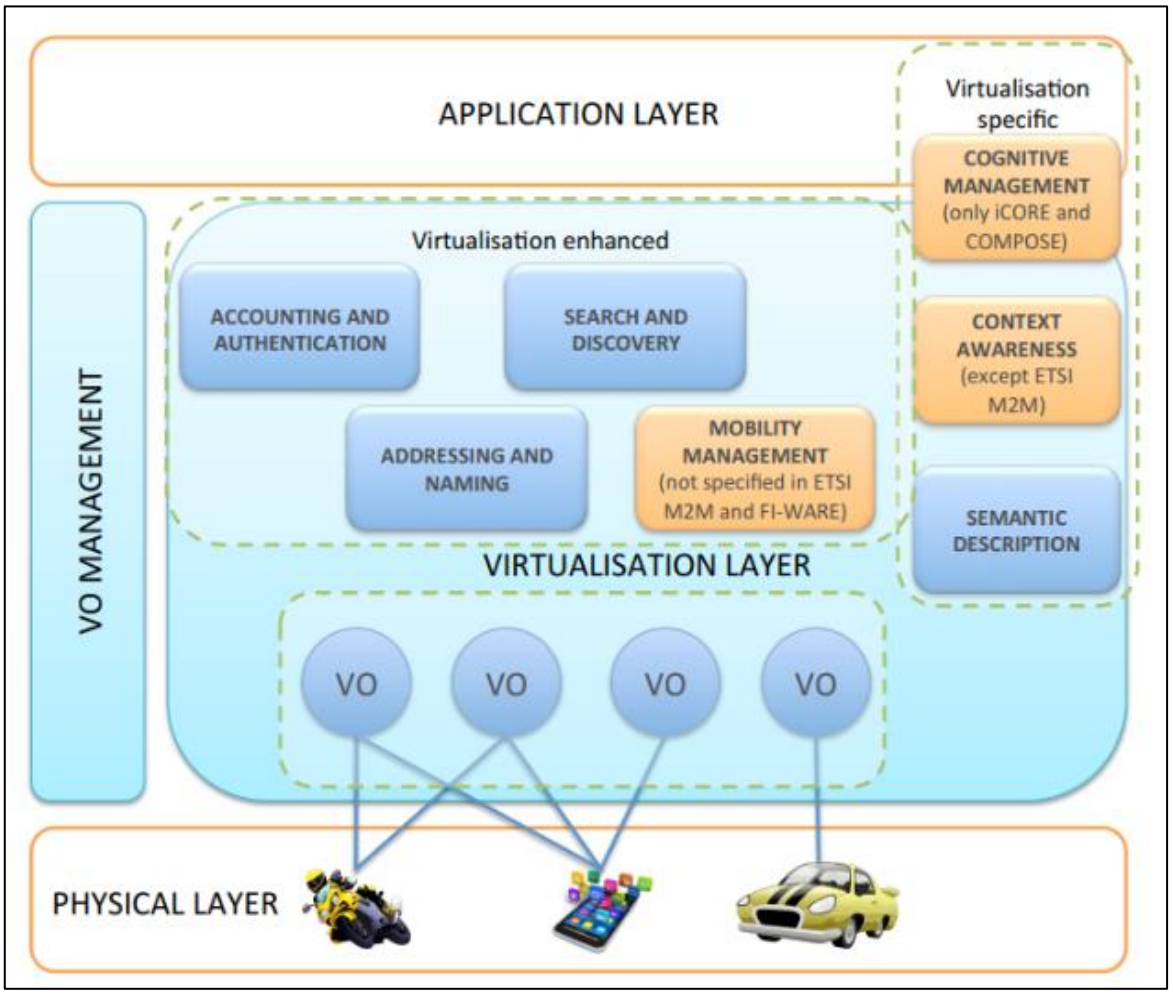


Figura 5. Virtualización de un sistema IoT (Din, et al., 2019, p. 7625).

La arquitectura de un sistema IoT puede abarcar diferentes escenarios en función de las necesidades del entorno, en la Figura 5, se evidencia como los autores llaman capa física a la capa de percepción, y sustituyen las capas de procesamiento y de red, por una de virtualización. La capa de virtualización incluye elementos virtuales para interconectar sus servicios y objetos reales.

Si bien no existe aún una estandarización de arquitecturas para tecnologías IoT, se han realizado aportes que evidencian una tendencia hacia situaciones específicas como el acceso al medio y dispositivos o la conectividad de estos al internet.

La IEEE (Institute of Electrical and Electronics Engineers) destacan protocolos como el 802.15 y 802.15.4 que propicia la comunicación mediante bajas tasas de transmisión para la integración de dispositivos de recursos limitados.

El protocolo CoAP o Constrained Application Protocol (Protocolo de capa de aplicación), se especializa en la transferencia de datos por la web permitiendo a una variedad d dispositivos conectarse y compartir información.

La W3C (World Wide Web Consortium), mediante el protocol SOAP (Simple Object Access Protocol) proporciona estándares de cómo los dispositivos u objetos pueden comunicarse a través de datos en formato XML.

En función de los modelos referenciados por los autores y los estándares propuestos por las diferentes entidades mencionadas se puede definir un conjunto de protocolos que pueden ser integrados a una arquitectura soportada en IoT:

Tabla 3. Tecnologías para una arquitectura soportada en IoT

Tecnologías para una arquitectura soportada en IoT	
Servicios web	SOAP REST
Capa de aplicación o procesamiento	HTPP
Capa de transporte	TCP
Captura de datos	802.15.4

Elaboración propia del autor

Actualmente, el Internet de las Cosas tiene aplicaciones en áreas como la salud, la construcción, agricultura de precisión, educación, control de tráfico vehicular, entre otros.

2.4.1. Visión artificial

La visión artificial o visión por computador es un conjunto de tecnologías que forman parte del área de la inteligencia artificial. Puede ser definida como la ciencia de programar un ordenador para el procesamiento de imágenes y videos (Alvear, Rosero, Peluffo, & Pijal, 2017).

Es la capacidad que tiene un ordenador de observar el mundo que lo rodea, deduciendo su estructura y propiedades a partir de un conjunto de imágenes bidimensionales. Son sistemas autónomos que pueden cumplir funciones que van desde una simple detección de objetos hasta la interpretación de diversas escenas. Esta disciplina se encuentra en pleno desarrollo, y sus aplicaciones han ido en aumento sobre todo en campos como la robótica, inspección automática, análisis de imágenes médicas, o navegación de vehículos.

Está compuesta por un conjunto de procesos encargados del análisis de imágenes, como son la captación, memorización, procesamiento e interpretación de datos (Aperador, Bautista, & Mejía, 2013). Algunas de las principales aplicaciones de esta ciencia son controles de calidad de productos, control vehicular, reconocimiento facial, entre otros.

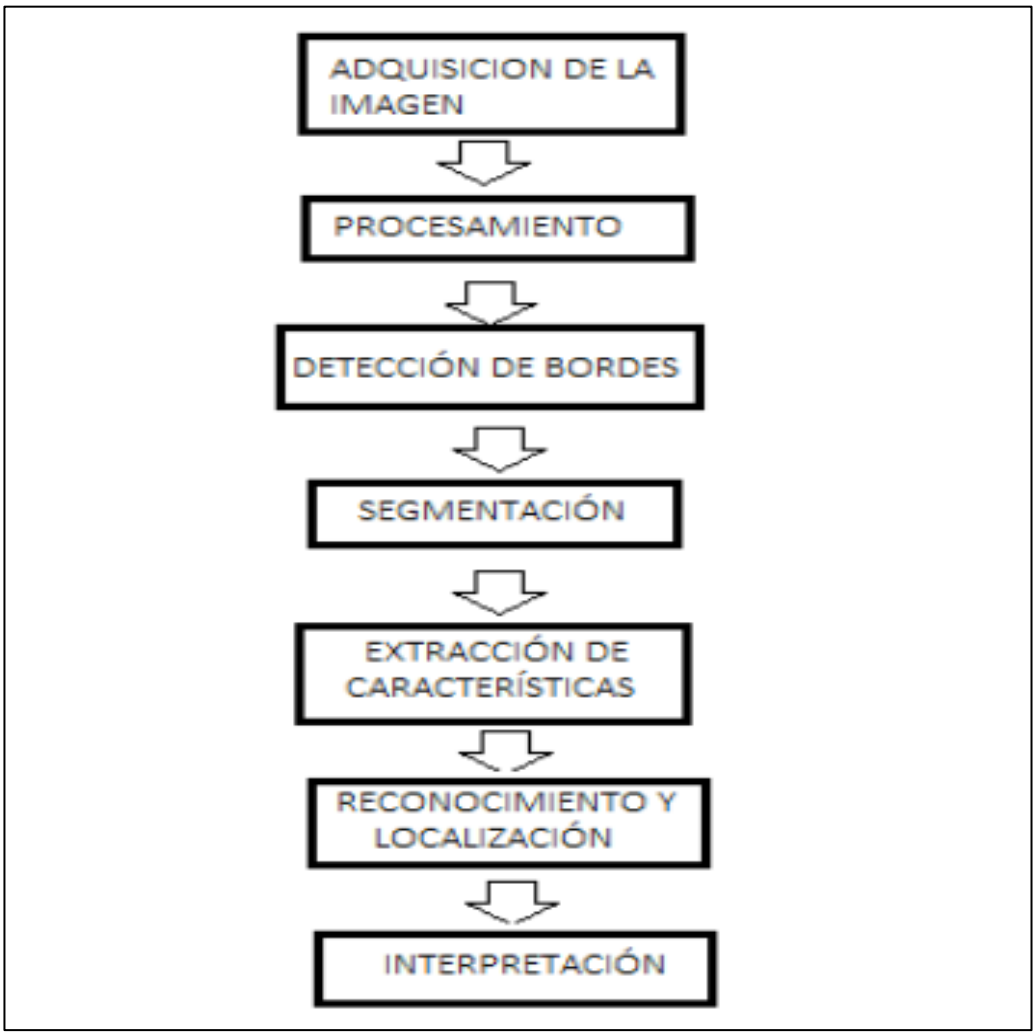


Figura 6. Funcionamiento de la visión artificial al determinar una imagen. (Alvear, Rosero, Peluffo, & Pijal, 2017, p. 248)

En la Figura 6 se muestra la manera en que es ejecutado un proceso de reconocimiento de imágenes por visión artificial, la adquisición de la imagen hace referencia a la captura y

digitalización de la imagen. El procesamiento y detección de bordes son fundamentales en este proceso, ya que se discrimina el fondo, concentrándose en los objetos de interés (Álvarez, 2014).

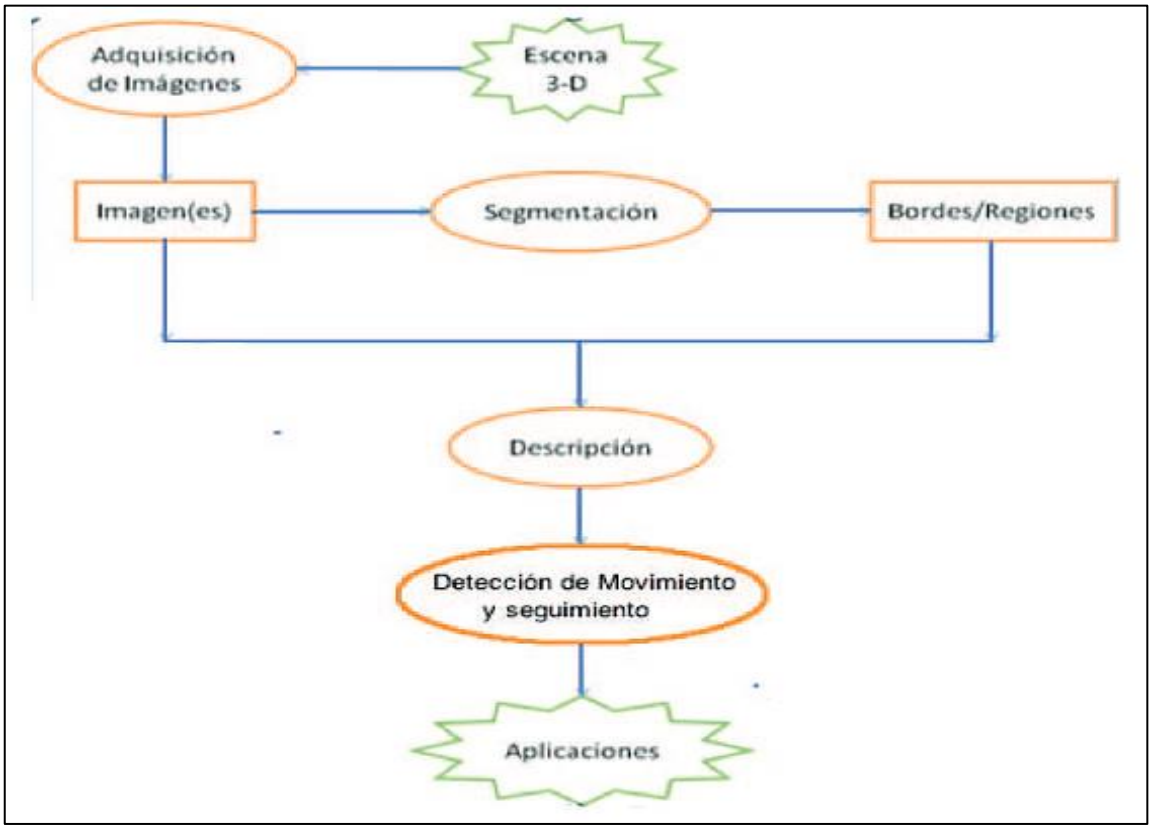


Figura 7. Esquema de un sistema de visión artificial. (Loaiza, Manzano, & Múnera, 2012, p. 89)

En la Figura 7 se observa otro esquema de visión artificial; en este, los autores integran en un mismo nivel las etapas de segmentación y bordes y regiones, lo que se tomaría como el procesamiento.

La segmentación de la imagen puede ser realizada en pixeles en función de sus valores, bien sea RGB (Red Green Blue), o HSV (Hue Saturation Value). Es necesario que la imagen sea tratada, con el objetivo de facilitar posteriores procesos (Alvear, Rosero, Peluffo, & Pijal, 2017). Al procesar, se disminuye el ruido de los dispositivos electrónicos, se mejora la calidad y se resaltan características de la imagen.

En cuanto al Internet de las Cosas, la aplicación de la visión artificial se ha extendido de manera exponencial, llegando a mejorar actividades de la vida diaria. En el área médica, ha sido posible la clasificación y visualización efectiva de imágenes (Sanabria S. & Archila D.,

2011). En este punto, la mejora de imágenes permite al médico una mejor interpretación de resultados.

Las “ciudades inteligentes”, donde el control vehicular aparece como uno de sus pilares fundamentales, también ha sido ampliamente abordado en el estudio de la visión artificial; la propuesta, se basa en la emulación de este tipo de proyectos, capturando y procesando matrículas de los vehículos, para así determinar los que pueden ingresar al parqueadero.

2.4.2. Sistemas automáticos de detección de matrículas

Para delimitar el alcance de la propuesta a desarrollar, es importante analizar el significado, funcionamiento y aplicaciones de estos sistemas.

Los sistemas de reconocimiento automático de una matrícula son conocidos como ANPR (Automatic Number Plate Recognition) o ALPR (Automatic License Plate Recognition). Se basan en la visión artificial para identificar vehículos mediante su número de matrícula. Son un caso particular de la visión artificial, están pensados para la lectura del contenido de una matrícula a partir de la captura de imágenes por una cámara.

Este es un método de vigilancia en masa, para leer las matrículas de los vehículos se basa en el reconocimiento óptico de caracteres. Pueden ser utilizados para el almacenamiento de imágenes capturadas por una cámara; de igual manera el texto de la matrícula e incluso, la fotografía del conductor.

Existen diferentes alternativas de actuación para la detección de matrículas. Los más conocidos y utilizados son la detección de límites verticales y horizontales, y los basados en la binarización. En cuanto a la primera técnica, esta tiene la función de realizar sumas de las características verticales, es decir, las variaciones drásticas en la gradiente de la imagen, para ello, se fija un umbral.

Figura 8. Ejemplo de reconocimiento por detección de límites verticales. (Autor 2020).

En cuanto a los sistemas basados en binarización, se puede decir que es un método muy utilizado; consiste en binarizar imágenes partiendo de un umbral óptimo, conseguido en función del histograma de la imagen, representando la imagen por la mayoría de sus pixeles blancos.

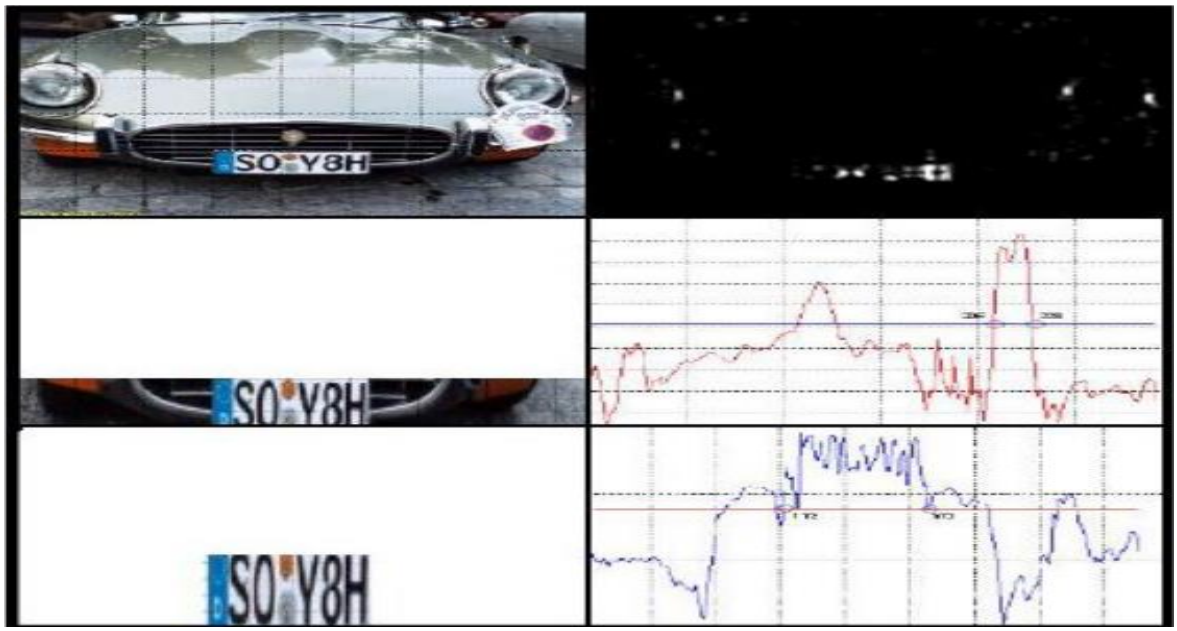


Figura 9. Reconocimiento por binarización. (Muñoz, 2014, p. 11)

El algoritmo general de sistemas de reconocimiento automático se basa en cuatro etapas (Lugo, Ton-Sieras, & García, 2016): 1) el mejoramiento, que inicia con la captura y detección de la imagen, 2) la restauración, que se enfoca en la alineación de la imagen, 3) el

procesamiento, donde se extraen lo caracteres necesarios y 4) la segmentación, donde se analizan dichos caracteres generando un ID de usuario.

Existen varios productos similares en el mercado; sin embargo, la mayoría están enfocados en el control de tráfico vehicular y generación de multas. Esta propuesta se diferencia al interactuar con un sistema establecido para determinar si un vehículo cuenta o no con acceso al parqueadero.

El sistema de reconocimiento a desarrollar se ejecutará en un ordenador convencional, y se enlazará con el resto de aplicaciones y la base de datos donde reposa la información de los usuarios del parqueadero de la Universidad Nacional de Educación **(ver Anexo 1)**; se trata de una base de datos PostgreSQL que cuenta con tablas donde se registra la solicitud de parqueo por cada usuario, además de si el mismo ha realizado el pago para que dicha reserva se haga efectiva.

Se utiliza una serie de técnicas para la detección y tratamiento de la imagen y extraer los datos alfanuméricos necesarios para la validación de acceso; cada uno de estos procesos se especifican dentro de la arquitectura que se plantea.

Los sistemas ANPR incluyen procesos o algoritmos para la detección y locación de una matrícula aislando el elemento a capturar, su orientación y tamaño (Betancor, 2008). La normalización y segmentación también son aspectos fundamentales para el ajuste de brillo, contraste y delimitar los caracteres.

Finalmente, el análisis sintáctico y geométrico, permite comprobar los caracteres encontrados y sus posiciones.

Figura 10. Proceso de reconocimiento de imagen. (Autor 2020).

Para establecer la comunicación, los sistemas de reconocimiento automático manejan diferentes alternativas como la comunicación serial, redes de área local o comunicación por bus de campo.

Por su parte Campos (2014), menciona que estos sistemas pueden encontrar adversidades como una pobre resolución de imagen y baja iluminación o desenfoque; es por ello que la ubicación y calibración es importante de acuerdo a la complejidad del prototipo y la utilización que requiera dársele. Un punto a favor a tener en cuenta es que, al estar ingresando al parqueadero, los autos deben detenerse, lo que da el tiempo necesario para que la cámara capture la imagen de forma adecuada.

Los sistemas de reconocimiento automático de matrículas se basan normalmente en redes neuronales; generalmente se encargan de la clasificación se los caracteres encontrados en una matrícula a partir de los datos capturados en el proceso de segmentación.

A gran escala, estos modelos computacionales tienen una alta capacidad de aprendizaje; su desempeño ha provocado que en la actualidad sean muy utilizadas en proyectos de visión artificial.

Por su parte las redes neuronales convolucionales (CNN), hacen referencia a campos de recepción que emulan el comportamiento de las neuronas en la corteza visual primaria de un cerebro biológico. Consisten en múltiples capas de filtros convolucionales de una o más

dimensiones. Después de cada una de sus capas añade una función para realizar un mapeo casual.

2.5. Python

Al desarrollar proyectos basados en Internet de las Cosas, es importante dominar o conocer lenguajes de programación o plataformas que enriquezcan y faciliten alcanzar los objetivos planteados.

Python, se presenta como uno de los lenguajes de programación más populares y utilizados, puede ser instalado en el sistema operativo de dispositivos IoT como Raspbian o Raspberry Pi, permitiendo la escritura de código que puede incluir la interacción con alarmas, sensores o el control de motores.

Cuenta con un índice de paquetes que alberga diversos módulos desarrollados pro terceros, ya sea en estos o en su biblioteca estándar es posible encontrar infinitas posibilidades para áreas como:

- Acceso y control de bases de datos
- GUIs de escritorio
- Desarrollo web
- Educación
- Desarrollo de juegos

Se ha escogido este lenguaje como lenguaje base de la arquitectura de software a proponer debido a su sintaxis intuitiva y clara, además de la variedad de librerías especializadas con las que cuenta como es el caso de NumPy, eficaz en el proceso de operaciones con matrices y vectores, aspecto importante al momento del tratamiento de imágenes.

2.6. OpenCV

Sus siglas significan Open Source Computer Vision Library, por ende, se puede concluir que OpenCV es una librería que facilita el procesamiento de imágenes. En principio fue pensada para aplicaciones de visión artificial en tiempo real. Es multiplataforma y cuenta con versiones para GNU/Linux, Windows o Mac OS.

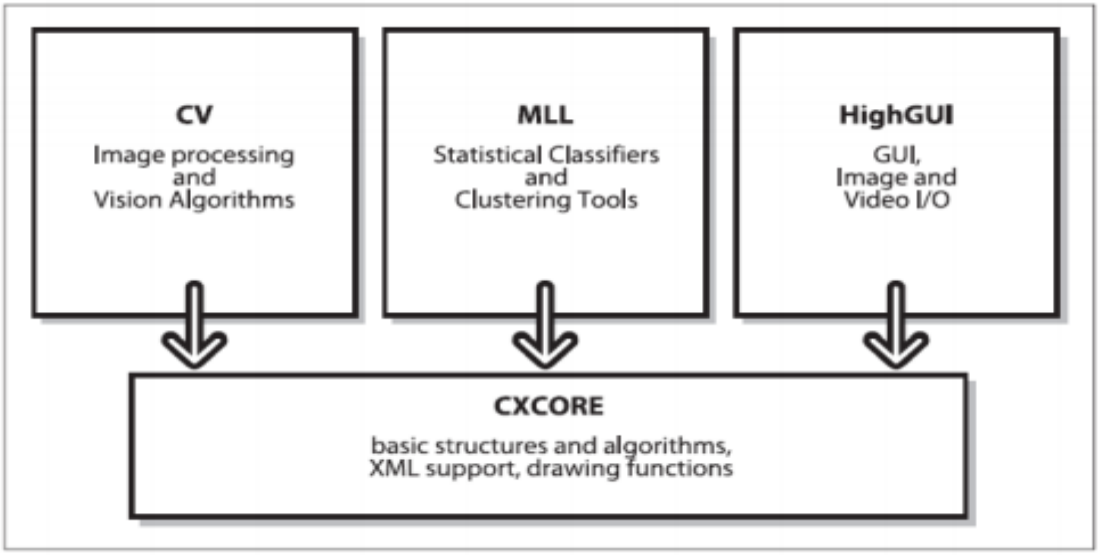


Figura 11. Estructura básica de OpenCV. (Campos, 2014)

La librería ofrece un entorno de desarrollo sencillo y eficiente; ha sido utilizado en diversos sistemas, desde seguridad con detección de movimiento hasta aplicaciones de manejo de procesos y tiene cuatro módulos principales: CV, repositorio de sus funciones básicas, Cvaux que contiene funciones auxiliares, Cxcore que hace referencia a estructura de datos y Highgui, que implementa funciones para el manejo de GUI.

El módulo de OpenCV para el lenguaje Python proporciona paquetes para el desarrollo de programas que interactúan con archivos de video; es posible importar el módulo completo o solo paquetes internos como “cv”. Para la propuesta se usan los paquetes de CV que contienen controles de movimiento y funciones para el seguimiento de objetos y extracción de fondos.

Las áreas de aplicación de esta librería incluyen características 2D y 3D, estimación de pose de cámara, la posibilidad de reconocimiento de gestos y facial, robótica móvil, segmentación e incluso realidad aumentada.Es multiplataforma, ofrece soporte para varios sistemas operativos y arquitecturas de hardware como celulares o Raspberry Pi.

OpenCV ofrece una estructura de visión por ordenador sencilla para el usuario, permitiendo crear aplicaciones sofisticadas y potentes (Campos, 2014). Contiene alrededor de 500 funcionalidades en relación a diferentes áreas, y es muy utilizada en gestión de imágenes médicas, robótica, seguridad, entre otras.

Esta librería ha sido utilizada en varios proyectos conocidos, como la visión del vehículo no tripulado Stanley, de la Universidad de Stanford, diseño que en el año 2005 ganaría el

Gran desafío DARPA, ha sido clave en el programa Swistrack, que es una herramienta de seguimiento distribuido.

Otro de los motivos para adopción de este recurso para el desarrollo práctico del prototipo, es que existen diferentes prototipos de libre acceso en la nube, por lo que resulta más sencillo la adaptación de diferentes soluciones, en una que satisfaga las necesidades expuestas en la problemática.

2.7. Algoritmo K-NN

Para facilitar el reconocimiento de caracteres, existen diferentes técnicas y metodologías, entre las más populares están las redes neuronales (ANN) y los algoritmos del K-vecinos más cercanos (KNN).

Para esta investigación se hace uso de la técnica K-NN, es un algoritmo de aprendizaje supervisado, es decir, que cumple la meta de clasificar instancias a partir de un juego de datos iniciales.

A diferencia de otros algoritmos, K-NN, no genera un modelo a partir del aprendizaje con información de entrenamiento, sino, en este el aprendizaje se realiza en simultáneo con las pruebas y test que se realicen con los datos.

Esta metodología sencilla, ideal en la introducción al Machine Learning; se presenta como parte fundamental de la arquitectura propuesta, en la medida en que sirve para “entrenar” la biblioteca OpenCV en cuanto a la lectura de bordes.

La biblioteca, cuenta con un algoritmo llamado **KNearest_create,** muy utilizado en la visión por computador.

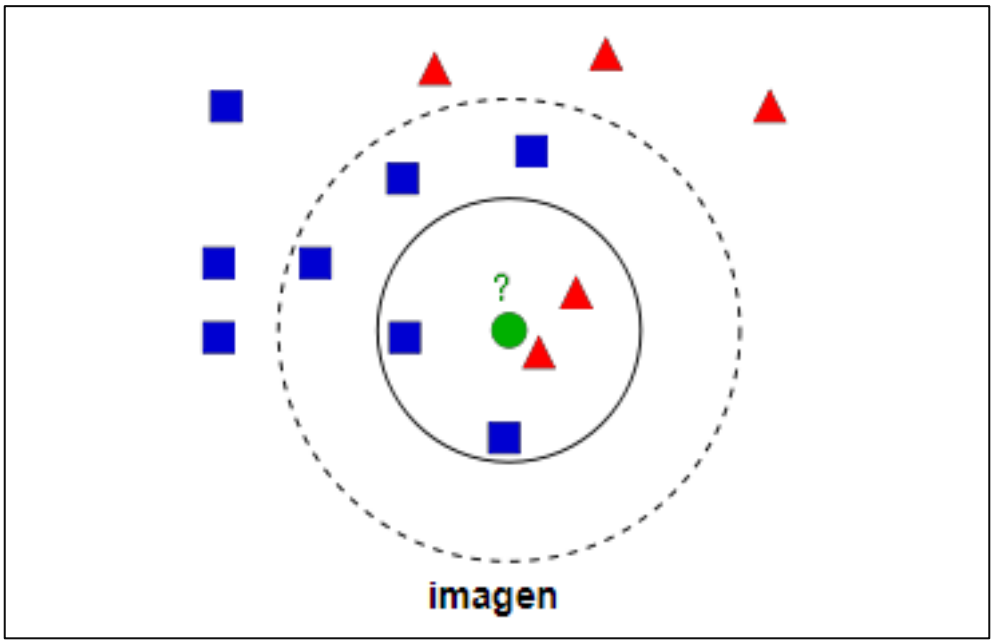


Figura 12. Funcionamiento básico del algoritmo KNearest_create. tomado de: *https://docs.opencv.org/3.4/d5/d26/tutorial_py_knn_understanding.html*

En la imagen se muestra un modelo de cómo este algoritmo identifica su “vecino” más cercano, calculando la distancia entre el ítem a clasificar y los demás datos, modelo que ha sido ampliamente aplicado en sistemas de recomendación, búsqueda semántica o detección de anomalías.

González, Santos, Campos, & Morell (2016) especifican que este algoritmo es uno de los de mejor desempeño en cuanto a Machine Learning. Constituye un modelo básico y entendible aplicable a problemas de predicción basado en los objetos del conjunto de datos para el entrenamiento. Estas características lo hacen ideal para la comparación de caracteres.

3. Objetivos concretos y metodología de trabajo

3.1. Objetivo general

Proponer una arquitectura de software para la integración de servicios IoT en la gestión de la Universidad Nacional de Educación.

3.2. Objetivos específicos

1. Analizar las principales tecnologías y tendencias en cuanto a la captura y procesamiento de imágenes basadas en IoT.
2. Definir la arquitectura de software que permita utilizar los datos censados por un sistema IoT en la gestión de procesos de la Universidad Nacional de Educación.
3. Validar la arquitectura propuesta a través de la implementación de un sistema de control de acceso al parqueadero de la Universidad Nacional de Educación.

3.3. Metodología

Para el cumplimiento de los objetivos planteados es necesario considerar una serie de pasos a seguir dentro de un proceso sistemático:

1. Indagar y adquirir fundamentos conceptuales acerca de las principales arquitecturas, protocolos de comunicación y algoritmos de detección y procesamiento de imágenes, para generar el conocimiento necesario para la finalización del proyecto propuesto.
2. Establecer ajustes para el preprocesamiento de imágenes, técnicas de segmentación y binarización de la imagen y detección de caracteres.
3. Definir los servicios web encargados de la validación de permisos una vez obtenido los datos.
4. Integrar en la arquitectura el sistema de detección y el servicio web de validación de datos.
5. Evaluar la arquitectura propuesta mediante la implementación de un prototipo para la detección de matrículas para el ingreso al parqueadero de la Universidad Nacional de Educación.

El tipo de investigación realizada es experimental, ya que es importante entender los diferentes escenarios que se puedan presentar en cuanto a las ventajas y limitaciones de las tecnologías escogidas para el desarrollo basadas en datos cuantitativos.

Se hace uso de esta metodología para plantear la problemática desde un punto de vista práctico. Se implementa en un estudio de caso concreto como es el reconocimiento de matrículas para validar el acceso al parqueadero de la Universidad Nacional de Educación.

Se miden variables concretas como la eficiencia de la arquitectura en función de tiempos de respuesta y exactitud.

Finalizado el desarrollo del sistema de reconocimiento y la evaluación de la arquitectura como tal, se plantea la aplicación de una encuesta para conocer la percepción de varios involucrados.

La encuesta diseñada cuenta con seis preguntas cerradas con alternativas en función de escala de Likert que permitirá conocer la percepción y nivel de satisfacción en relación al prototipo implementado. Se ha concebido que la encuesta se aplique en el campus de nivelación de la universidad, tres de las preguntas están dirigidas a los guardias de seguridad que controlan el acceso al parqueadero y las otras tres al personal administrativo, docente y alumnos que hacen uso del parqueadero.

La Universidad cuenta con cinco guardias de seguridad a los que se le aplicó la encuesta; por su parte la población actual de usuarios del parqueadero asciende a trescientas personas, por lo que se hace necesario la aplicación de una fórmula de muestreo.

$$n = \frac{N\infty^2 \,.\, z^2}{(N-1)e^2 + \infty^2 . z^2}$$

Donde:

N: Tamaño total de la población = 300

Z: Nivel de confianza 90% = 1,64

∞**=** 0.5

e = Precisión (Error máximo admisible en términos de proporción) = 0.5

$$n = \frac{300(0{,}5^2)\ x\ (1{,}64^2)}{(300-1)\ (0.05^2) + \ 0.25\ x\ \ 1{,}64^2}$$

$$n = \frac{75\ x\ \ 2{,}69}{0{,}75 + 0{,}67}$$

$$n = \frac{201}{1{,}42}$$

$$n = \ 140{,}44$$

La fórmula aplicada da un total de 140 personas como tamaño de muestra para la aplicación de la encuesta.

4. Desarrollo de la arquitectura

4.1. Identificación de requisitos

Para definir la arquitectura, se utilizan las metodologías que permiten establecer modelos de negocios con diseños soportados en IoT, integrando funcionalidades de una arquitectura Lambda y la Arquitectura de Referencia para IoT (IoT ARM), en función de sus capas de captura de datos, procesamiento y presentación.

Se implementa una capa para la captura y extracción de los datos a censar o evaluar; la misma permite la integración de diferentes elementos de percepción como cámaras web, sensores y demás dispositivos. La capa de procesamiento tiene la capacidad de analizar y dar tratamiento a los datos en bruto emitidos por la capa de censado, aplicando técnicas de minería de datos y la implementación de motores de base de datos.

En la capa de presentación se integran interfaces o dispositivos que permitan la visualización de la información resultante del procesamiento realizado, facilitando así la comunicación e interacción de las diferentes plataformas y herramientas de la Universidad Nacional de Educación.

Desde una perspectiva general, la arquitectura comprende tareas como el análisis y el tratamiento de datos, hasta la implementación de una interfaz de administración con el objetivo de mejorar la gestión de procesos en la Universidad Nacional de Educación; es por ello que la misma debe permitir:

- Una eficiente conectividad y comunicación.
- Gestión multidispositivo.
- Recolección, análisis y actualización de datos.
- Escalabilidad y flexibilidad.
- Alta tasa de disponibilidad.
- Integración y seguridad.

4.2. Descripción de la arquitectura propuesta

Se describe la arquitectura web soportada en IoT representada desde un modelo funcional y un modelo de implementación. En el estilo funcional se detallan los componentes de la arquitectura propuesta, mientras que la vista de implementación hace referencia a los dispositivos de hardware y el software que la integran.

- **Modelo funcional:**

Basados en los paradigmas de la arquitectura Lambda se presenta un modelo arquitectónico de cuatro capas:

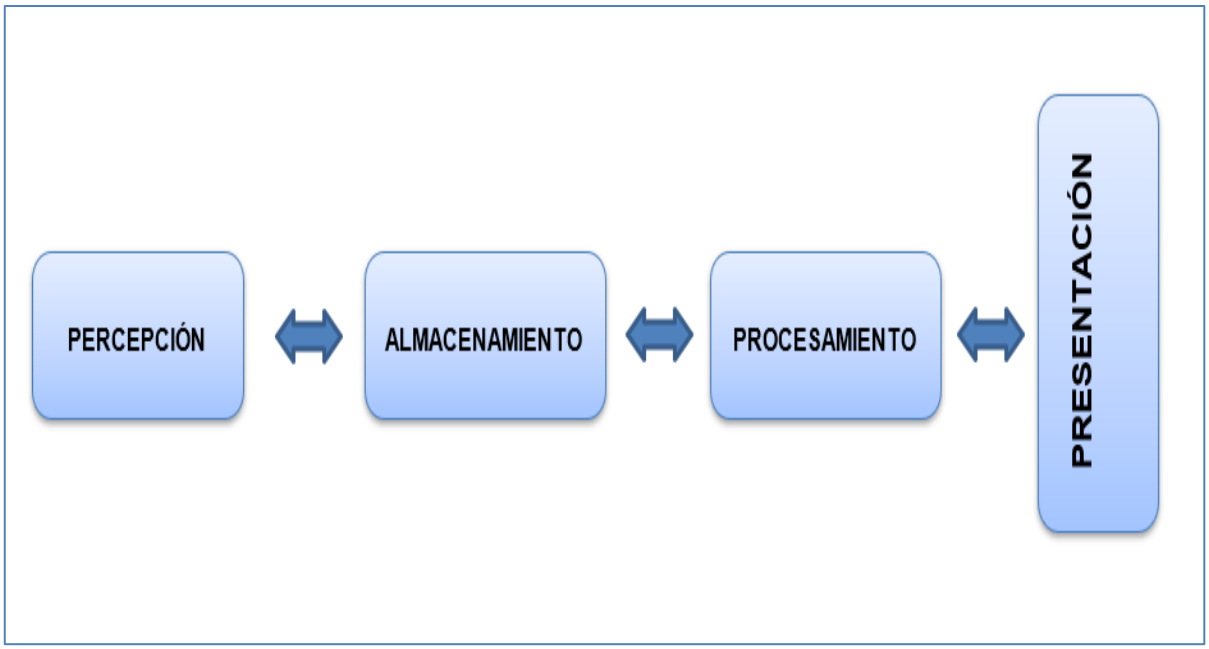


Figura 13. Modelo funcional de la arquitectura propuesta. Elaboración propia del autor

En la Figura 13 se muestra la interacción de las capas que contiene la arquitectura propuesta; la capa de percepción es la encargada de la captura de datos; en esta capa se pueden integrar diferentes periféricos o dispositivos como sensores, en el caso de requerir controlar variables ambientales, dispositivos biométricos para el control de acceso, o cámaras para reconocimiento óptico de caracteres.

Los datos capturados en la capa de percepción se envían a la capa de almacenamiento donde se registra la información capturada; posteriormente la capa de procesamiento, mediante API REST, consulta y procesan estos datos. En el caso de estudio propuesto, se detalla un diagrama de la secuencia del tratamiento que se da a la imagen desde que es capturada y se integran las opciones de binarización y segmentación de la imagen. Es de destacar que el procesamiento concreto a realizar dependerá de cada caso concreto en que se utilice la arquitectura propuesta.

La capa de presentación hace referencia a los dispositivos o medios que permiten la visualización de los resultados de la verificación de la información capturada. También es posible integrar dispositivos actuadores, que realizarán acciones a partir de la información procesada.

- **Modelo de implementación:**

El modelo de implementación integra cada uno de los componentes tanto de hardware como de software necesarios para el desarrollo y despliegue de la arquitectura web soportada en IoT que se propone.

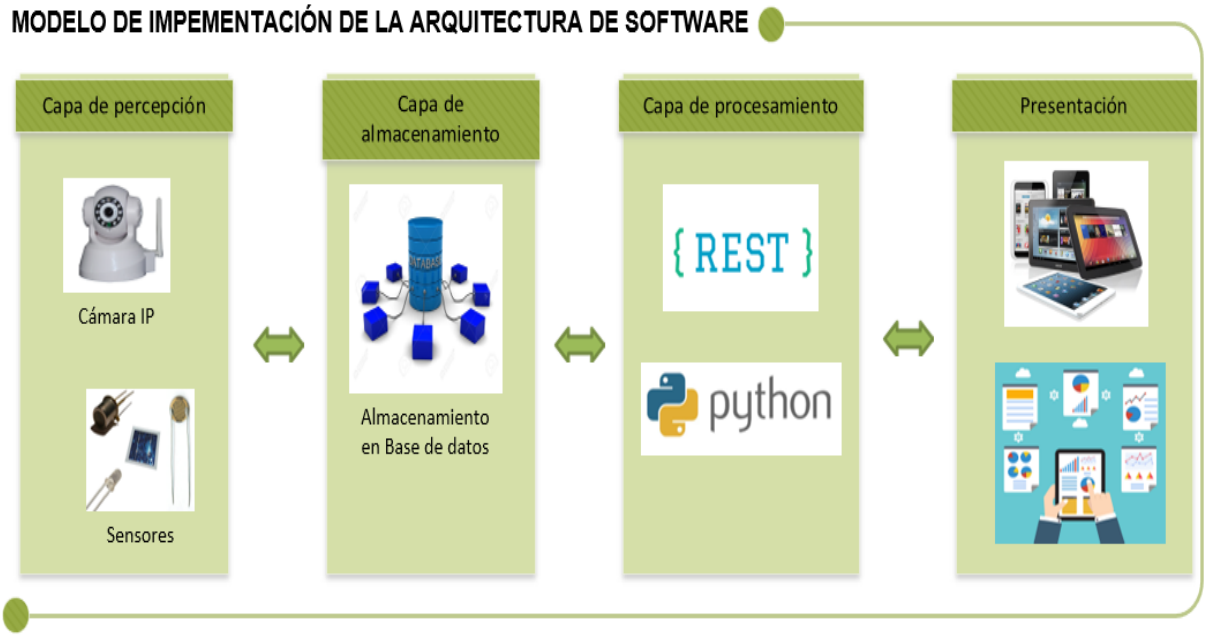


Figura 14. Modelo funcional de implementación. Elaboración propia del autor

En esta figura se detallan los elementos y procesos lógicos que intervienen en la captura y procesamiento de imágenes en el sistema; la capa de percepción es la encargada de la extracción de los datos en tiempo real. La capa de almacenamiento aloja la información para que sea accedida de manera eficiente mediante una API-REST para su procesamiento. La capa de procesamiento es la encargada de dar tratamiento a los datos en función de los requerimientos que se presenten; mientras que en la capa de presentación se realiza la interacción con los usuarios.

Una de las principales premisas de la arquitectura propuesta es incrementar la velocidad de respuesta y eficiencia en la gestión de procesos; es por ello que en la capa de procesamiento se incluyen tecnologías ligeras y potentes como REST y las funcionalidades del lenguaje de programación Python que cuenta con librerías eficientes para el reconocimiento óptico de caracteres y para el procesamiento de datos provenientes de sensores de variables ambientales o dispositivos biométricos que pueden ser requeridos en la gestión de datos provenientes de dispositivos IoT.

4.2.1. Capa de percepción

La arquitectura de referencia ha sido diseñada para gestionar diferentes procesos donde intervienen variedad de dispositivos que debe ser considerados; estos deben tener un tipo de comunicación, directa o indirecta, que lo enlaza a internet. HTTP y REST son muy conocidos y existen muchas librerías que lo soportan, al ser HTTP un protocolo simple basado en texto diferentes dispositivos pequeños son capaces de interactuar por ejemplo con recursos GET o POST; mientras que otros más complejos pueden utilizar librerías con un cliente completo de HTTP.

Esta capa es la responsable de recoger las propiedades físicas de los objetos (localización, temperatura, tamaño, entre otros). La librería Open CV cuenta con diferentes métodos para la lectura de imágenes, reconocimiento de caracteres, medición de temperatura ambiental o corporal. Esto hace que la arquitectura de referencia pueda ser aplicada en diferentes ámbitos y procesos requeridos por la Universidad.

4.2.2. Capa de almacenamiento

En toda arquitectura es necesario contar con un gestor de almacenamiento; este aspecto es también sugerido por el modelo Lambda. La base de datos que se integre debe permitir el manejo generalizado de datos de transacciones, eventos o procesos que se gestionan con la arquitectura, teniendo en cuenta una política de trazabilidad adecuada desde el punto de vista de la seguridad. No obstante, es importante destacar que el modelo de bases de datos necesariamente tendrá en cuenta cada contexto de aplicación específico.

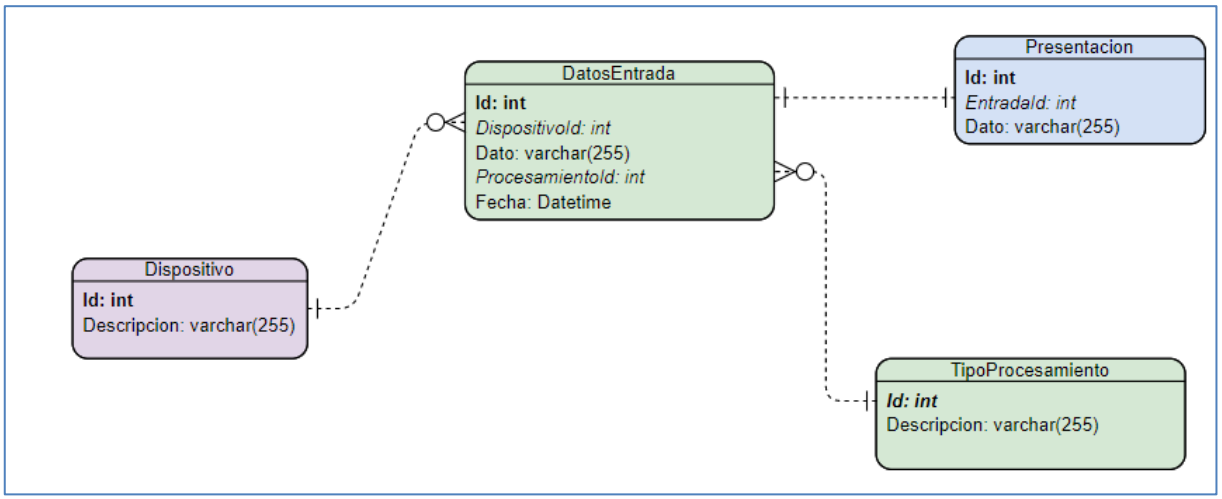


Figura 15. Diseño de base de datos. Elaboración propia del autor

En la Figura 15 se especifica el diseño del modelo relacional que independientemente del tipo de dispositivo permitirá el registro de las interacciones y operaciones del sistema. Los diferentes valores y atributos dependerán del dispositivo fuente, sin embargo, la interacción de la información deberá seguir este modelo de referencia.

La persistencia en base de datos facilitará el análisis, la consulta e interacción de la información. En el Anexo 1 se detalla el diseño de un modelo de base de datos genérico para la interacción con el caso de estudio propuesto.

4.2.3. Capa de procesamiento

La capa de procesamiento actúa a partir de la información obtenida en el bloque de percepción que expone una API-REST que permite a esta capa solicitar los datos, esta es transmitida hacia el sistema o dispositivo que procesa la misma determinando la secuencia de acciones a tomar dependiendo del resultado esperado.

Figura 16 Capa de procesamiento

La capa de procesamiento integra aplicaciones y tecnologías que permiten el tratamiento de los datos capturados, como se muestra en la figura 16, diferentes lenguajes y herramientas pueden formar parte de este nivel en la arquitectura propuesta para el análisis y evaluación de la información en tiempo real, minimizando los tiempos de acción en la ejecución de procesos.

4.2.4. Capa de presentación

Esta capa pone en uso los datos creados y procesados para su análisis, monitoreo y generación de reportes. Está compuesta por las interfaces o aplicaciones que permiten el acceso a la información de interés, así mismo, puede integrar un conjunto de dispositivos actuadores que realicen tareas a partir de los datos generados.

4.2.5. Diagrama de actividades

Este diagrama recopila los procesos generales que la arquitectura de software propone en cada una de sus capas:

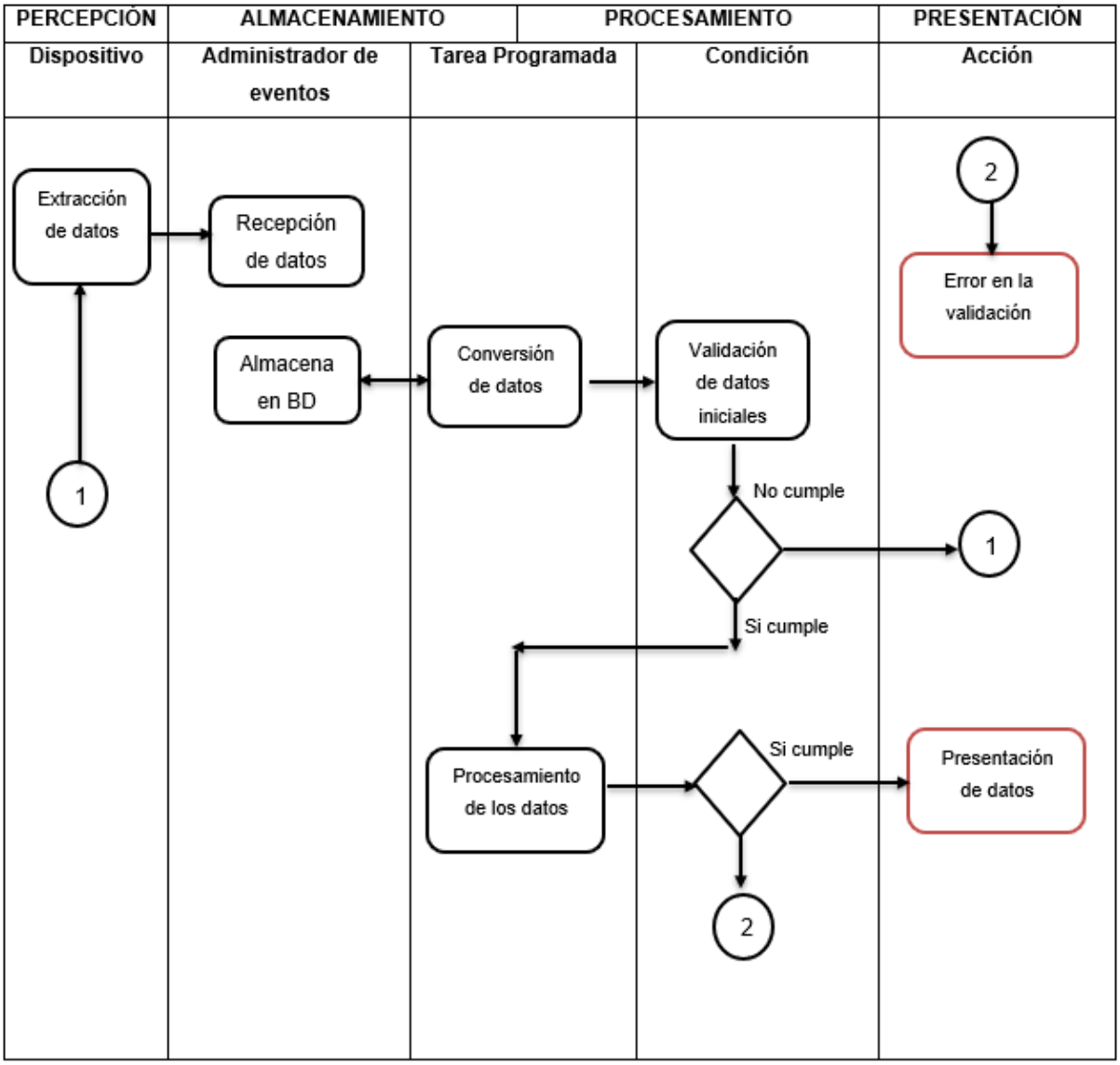


Figura 17. Diagrama de actividad del funcionamiento de la arquitectura propuesta

Para la arquitectura propuesta el diagrama de actividad que se expone en la figura 17; representa la secuencia de procesos que se toman en cuenta, se describen cada uno de los pasos y el flujo desde la extracción de información hasta la presentación de resultados.

Con bordes rojos se muestran los mensajes que se visualizan por interfaz, los demás gráficos hacen referencia a procesos lógicos.

4.2.6. Diagrama de estado

Un diagrama de estado, representa el conjunto de estados por los que pasa un sistema en función de los eventos que lo ejecuten.

Se controla el flujo de un estado mediante los eventos internos y externos el mismo es descrito desde el punto de vista del usuario. En la figura 18 se presenta el diagrama de estado del funcionamiento de la arquitectura para el procesamiento de los datos en el ámbito del caso de estudio que se utilizó para validarla. El comportamiento que se representa es genérico y se personalizará adecuando los procesos que varían según el contexto de aplicación.

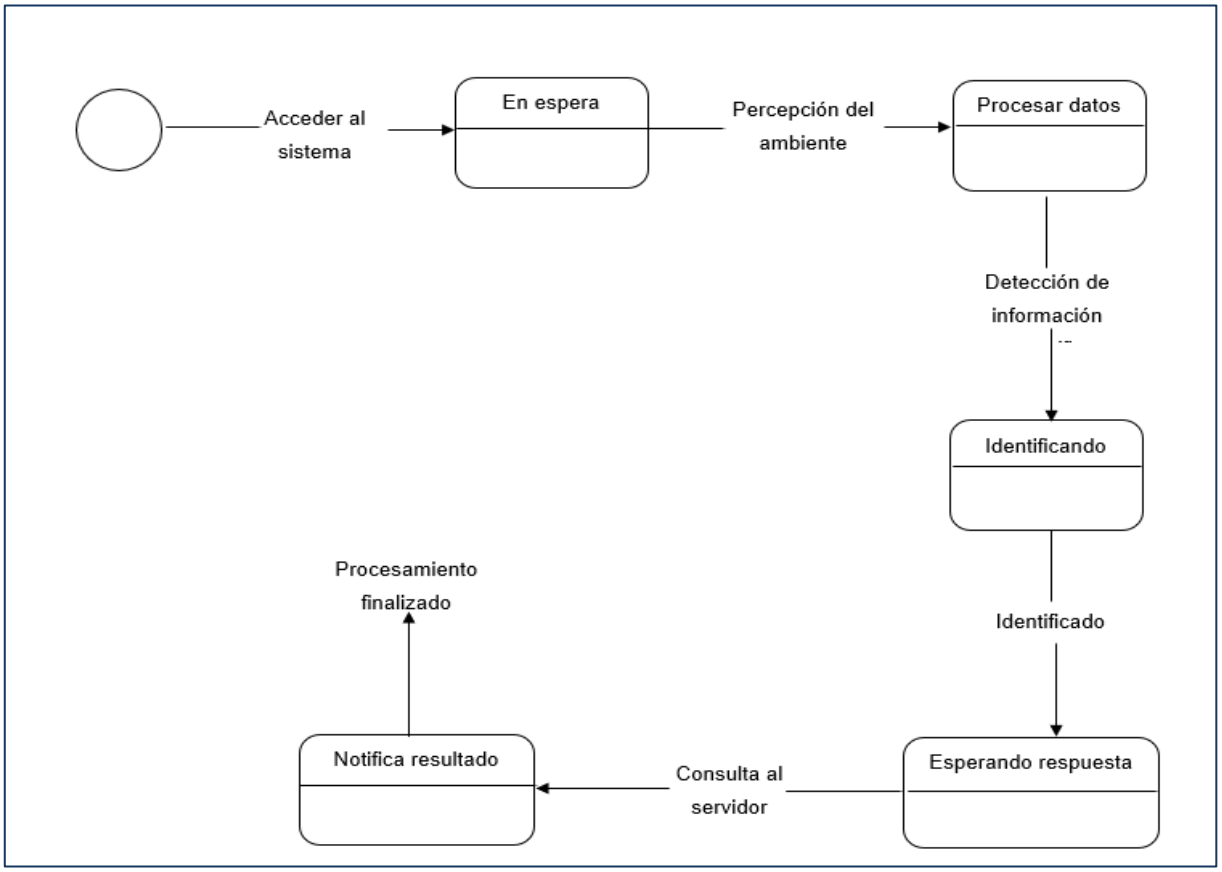


Figura 18. Diagrama de estado del funcionamiento de la arquitectura propuesta

El estado en espera hace referencia al tiempo que le toma a la aplicación el detectar un objeto válido que permita pasar al estado de procesar información, una vez que ha paso el reconocimiento, pasa al proceso de identificación donde interviene el servicio web; al final de todo el proceso, el sistema vuelve a un estado en espera.

4.3. Desarrollo de prototipo basado en la arquitectura propuesta

En este apartado se expone el desarrollo de una solución que permitió comprobar el adecuado funcionamiento de la arquitectura; se implementa un prototipo para la detección y validación de matrículas de vehículos que ingresan al parqueadero de la Universidad Nacional de Educación.

Las tecnologías utilizadas son:

- Python 3.7
- Django 2.8
- PostgreSQL 11
- Bootstrap 4
- Jquery
- Ajax

Librerías de Python que intervienen:

- Open CV 3
- Numpy 1.17
- Psycopg2 2.8 (Conector con base de datos)
- Requests 2.2 (Librería para la integración de Api’s REST)
- Pytesseract 0.3 (Utilizado para comparar el rendimiento del entrenamiento K-NN)

Se establecen las interacciones y rutas que la aplicación desarrollada ejecuta en función de las características plasmadas en la arquitectura propuesta como la escalabilidad y flexibilidad.

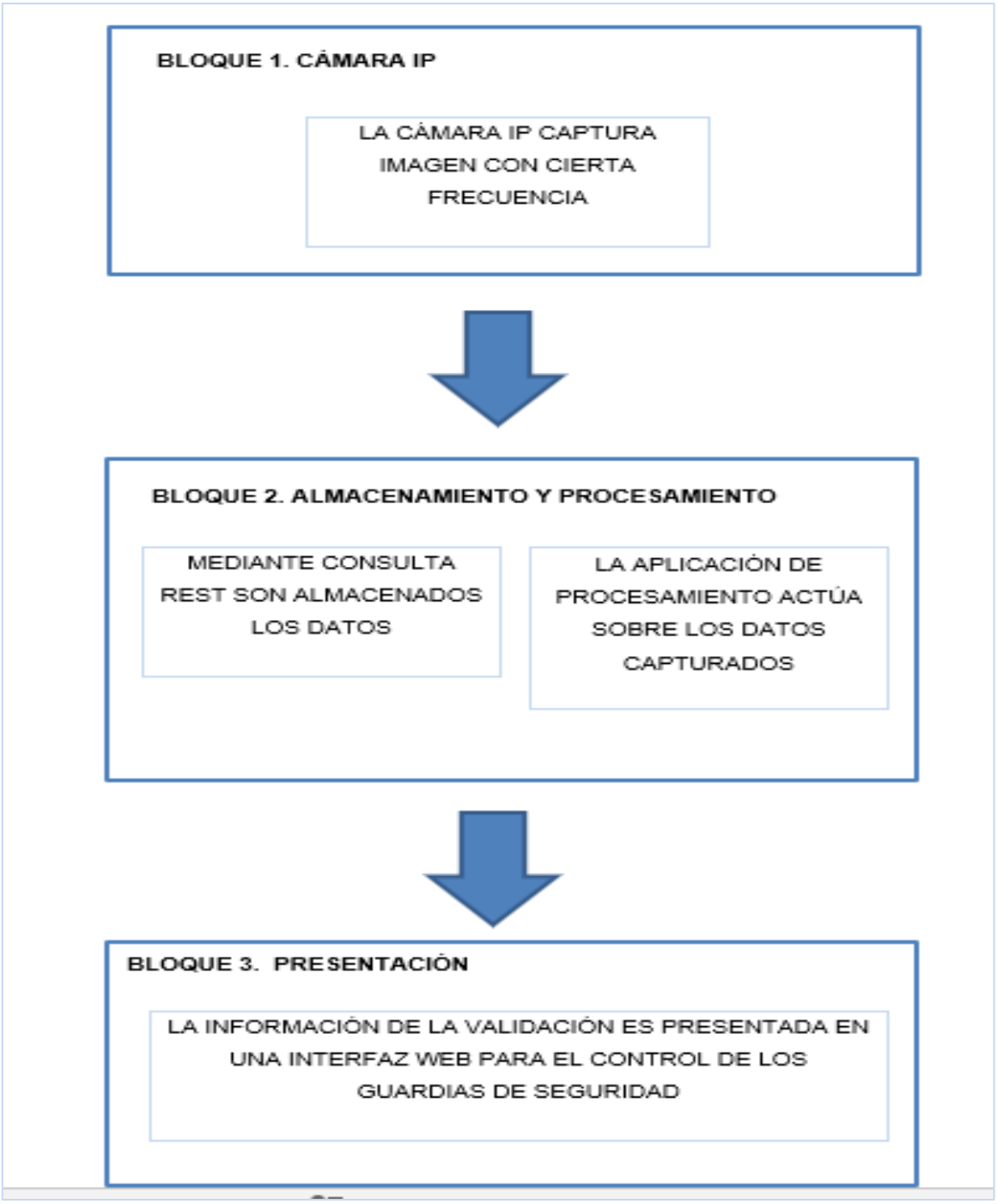


Figura 19. Diagrama de bloque del sistema

En la figura 20 se detalla un diagrama de bloque de la aplicación propuesta, donde cada bloque hace referencia a una capa de la arquitectura de software implementada. La cámara IP que actúa sobre la capa de percepción es la encargada de la extracción inicial de datos, misma que es almacenada en una base de datos para su análisis. El procesamiento inicia con el entrenamiento KNN para dar paso al tratamiento de la imagen capturada; encontrada la coincidencia, se valida si el vehículo cuenta con acceso al parqueadero, información que es mostrada en el bloque de presentación a través de una interfaz web.

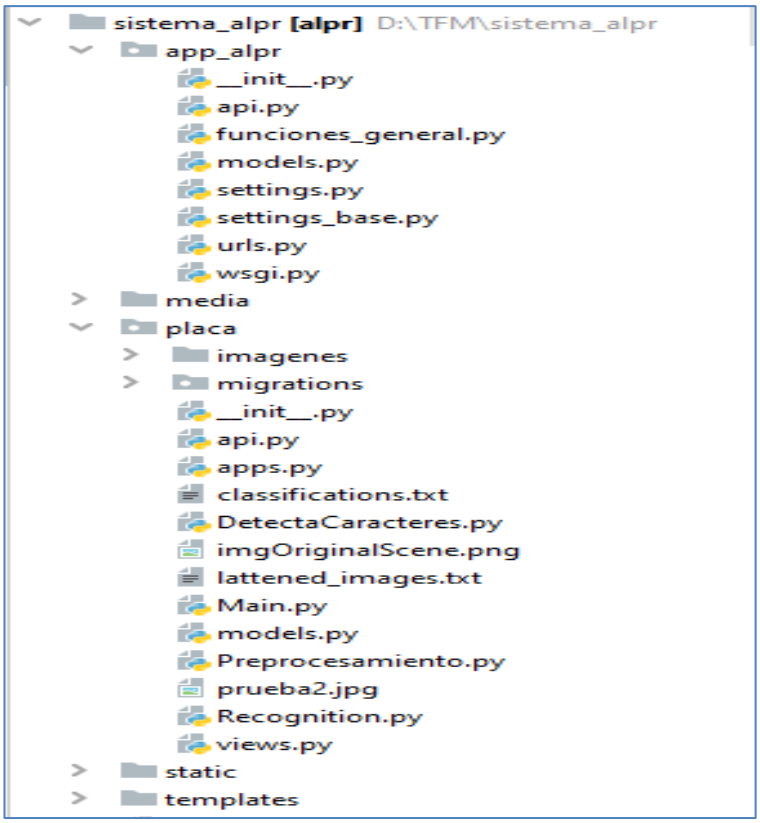


Figura 20. Estructura de carpetas de la aplicación

El sistema presenta una estructura sencilla, con una metodología MTV (Modelo Plantilla Vista), donde las vistas (ficheros .py) y las plantillas (documentos HTM) se encuentran debidamente separados.

- **Directorio app_alpr:** Contiene archivos de configuración como settings.py donde entre otras cosas encontramos el enrutamiento hacia archivos estáticos y ficheros multimedia, así como otras variables de configuración del sistema.
- **Directorio matrícula:** Es la aplicación en todo sentido, contiene los controladores principales que permiten que el sistema funciones.
    - **api.py:** Contiene las peticiones y respuestas REST a los servicios de verificación de usuarios y matrículas.
    - **DetectaCaracteres.py:** Contiene las funciones principales para la segmentación y detección preliminar de caracteres.
    - **Main.py:** Es el fichero de arranque del sistema, se encarga de la recepción de la imagen y de las llamadas a las funciones para su procesamiento.
    - **models.py:** Contiene las clases que sirven de apoyo en el procesamiento de las imágenes.

- **Preprocesamiento.py:** Es el tratamiento inicial de la imagen, conversión a escala de grises y ajuste de tamaño.
- **Recognition.py:** Es el fichero que almacena las funciones finales para la segmentación y el reconocimiento real de los caracteres encontrados.

- **Directorio media:** Contiene los archivos multimedia generados durante todo el proceso de detección de una matrícula.
- **Directorio static:** Contiene ficheros estáticos, es decir las hojas de estilo en cascada y los ficheros JS.
- **Directorio templates:** Contiene los documentos HTML que son las interfaces con las que interactúan los usuarios.

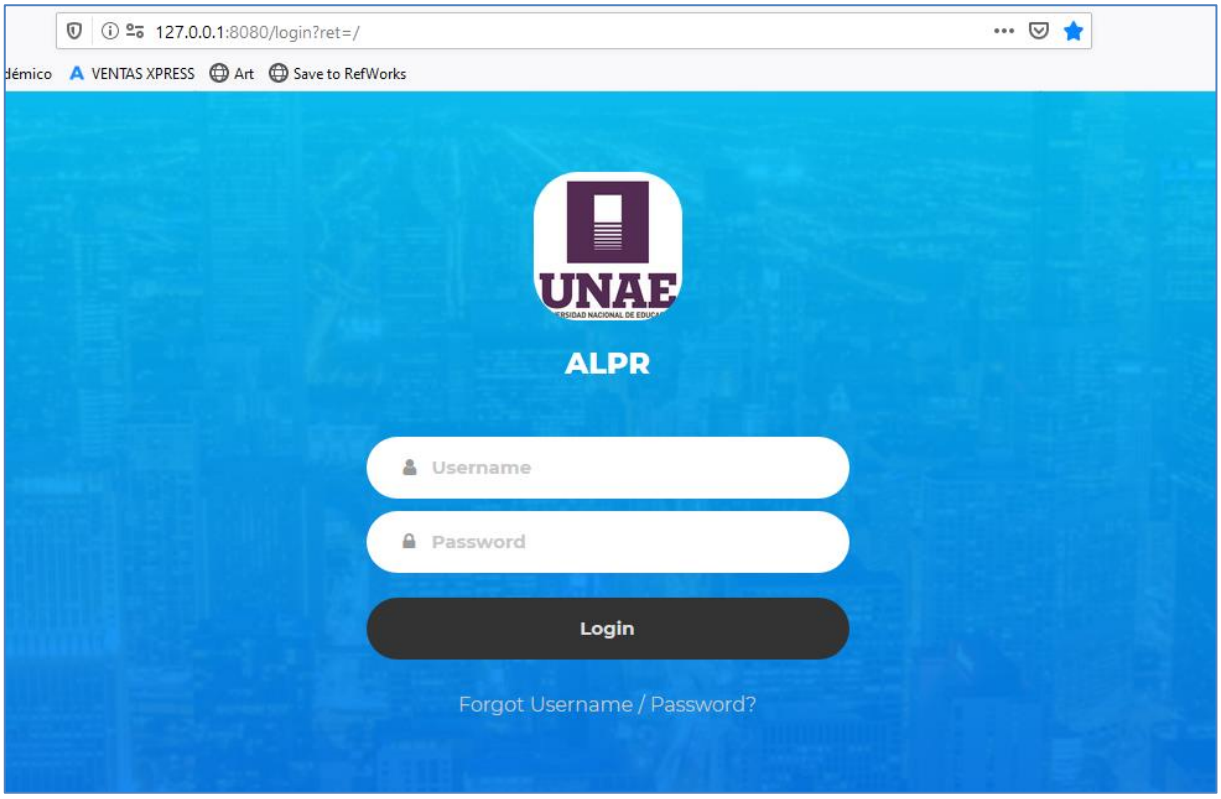


Figura 21. Pantalla de acceso al sistema

La figura 22 muestra la interfaz desde donde la aplicación caso de estudio, que implementa la arquitectura propuesta, comienza a interactuar. Se utiliza un servicio de validación de usuarios que reposa en la plataforma principal de la Universidad Nacional de Educación (Ver Anexo 2); si el usuario es correcto, retorna un token para confirmar el acceso.

Una vez iniciada sesión, se empieza a visualizar las constataciones de matrículas que el algoritmo de detección realiza.

La aplicación captura la imagen a través de la dirección IP de la cámara situada en la entrada al parqueadero (Ver Anexo 4). Para iniciar con el reconocimiento de caracteres es necesario realizar un entrenamiento sobre un conjunto de datos; se utiliza para ello la función KNearest propia de OpenCV (Ver Anexo 5).

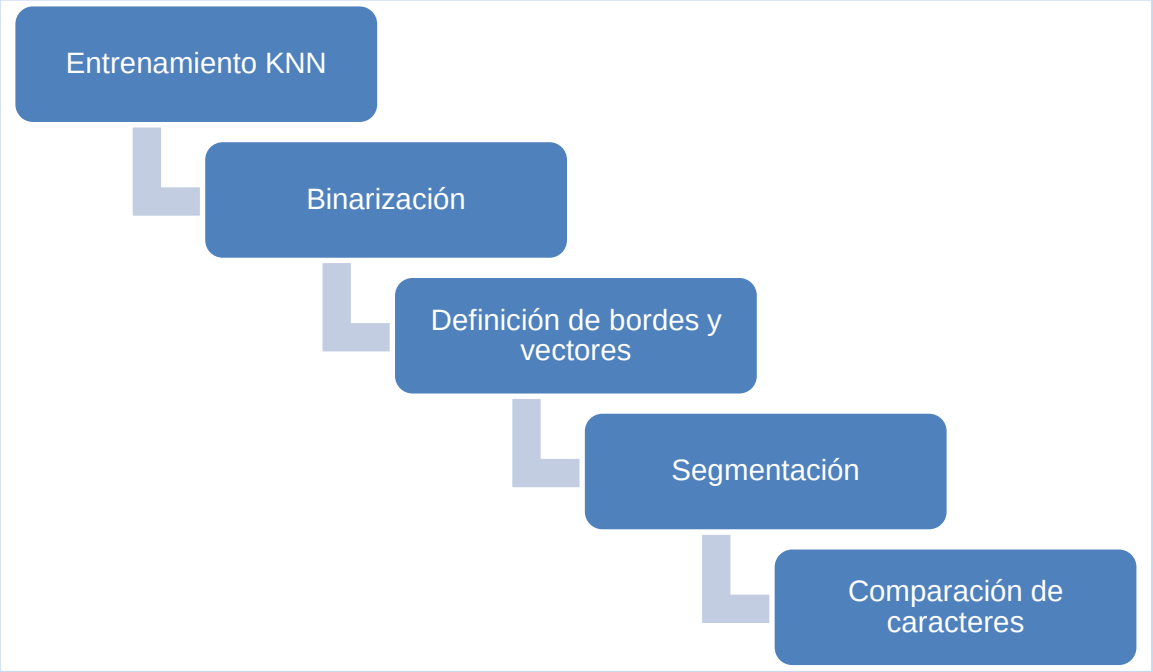


Figura 22 Secuencia de procesamiento

En la figura 23 se detalla la secuencia realizada en la capa de procesamiento; la misma inicia con el tratamiento de la imagen, se convierte a escala de grises y luego el threshold (umbralización) de contornos que son parte fundamental de la binarización, la información de la imagen procesada es contenida en dos objetos (Ver Anexo 3), permitiendo así que las funciones de tratamiento accedan a esta información de manera eficiente (Ver Anexo 6).

La primera API especificada busca la imagen almacenada para dar inicio al procesamiento de la misma mediante la URL url(**'^main$'**, Main.main, name=**'Main'**).

```
imgOriginalScene = cv2.imread(MEDIA_ROOT + "\\" + "prueba2.jpg")

if imgOriginalScene is None:
    print("\nerror: no se ha leido ninguna imagen\n\n")
    return HttpResponse(json.dumps({'result': 'bad', 'estado': '400', 'mensaje': 'No se ha podido leer imagen'}),
                        content_type="application/json")

listOfPossiblePlates = Recognition.detectarPlaca(imgOriginalScene)          # detectar placa

listOfPossiblePlates = DetectaCaracteres.detectCharsInPlates(listOfPossiblePlates)
# detectar caracteres en placa
```

Figura 23 Api de consulta de imagen capturada

En la figura 24 se muestra la consulta realizada a la API, si la cámara no ha detectado imágenes devuelve una respuesta BAD con el mensaje correspondiente, caso contrario se inicia el tratamiento de la imagen obtenida.

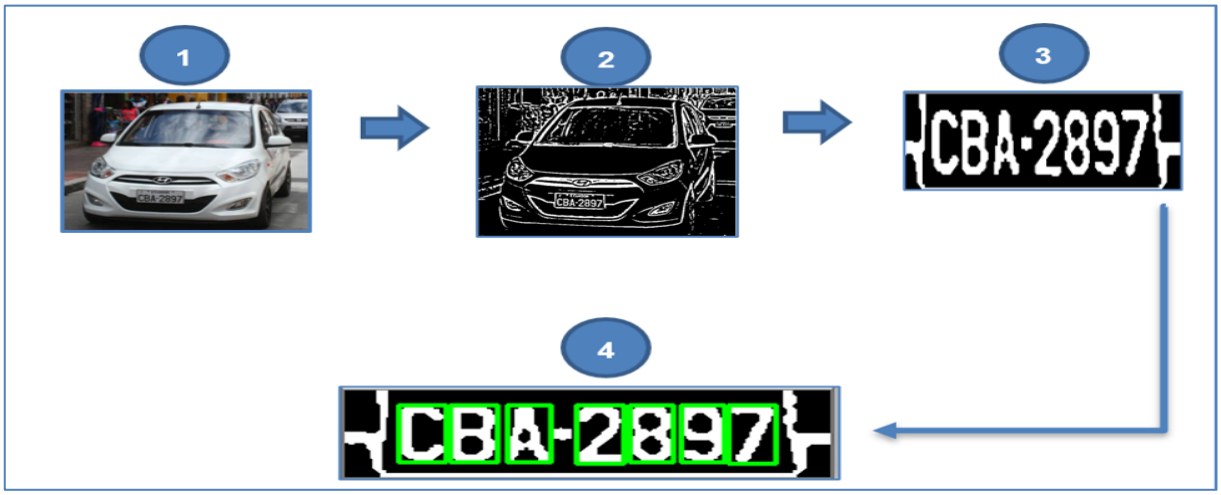


Figura 24. Funcionamiento de la capa de procesamiento. Elaboración propia del autor

En la figura 24 se evidencia el comportamiento de la capa de procesamiento una vez capturada y almacenada la imagen para el caso de estudio en particular en que se evaluó la arquitectura; es la encargada de dar tratamiento a los datos para obtener la información a evaluar. La detección involucra convertir a escala de grises la imagen capturada generando una distribución uniforme en los niveles grises; se aplican también operaciones morfológicas.

El procesamiento de la imagen inicia con el entrenamiento K-NN, para ello se utiliza un documento con imágenes planas que puede ser descargado de internet. Luego se crea un objeto para iniciar el entrenamiento **cv2.ml.KNearest_create()**.

La figura 24 muestra también una visión general de lo que esta capa realiza, desde el momento en que se accede a la imagen capturada hasta la segmentación de los caracteres encontrados. La implementación de esta capa depende del escenario en que se utilizará la arquitectura propuesta, siempre respetando el uso de Python como lenguaje de programación.

Figura 25. Imagen a escala de grises

Open CV cuenta con la función **cvtColor** que es la encargada de ajustar los colores en la imagen original como se visualiza en la figura 25. Poner en gris la imagen, facilita la binarización y maximizar el umbral de contornos.

Para el siguiente proceso se realiza un desenfoque Gaussian, es un efecto de suavizado que permite eliminar el ruido de fondo, se mezclan los niveles de gris entre los pixeles contiguos, lo que proporciona una imagen con bordes suaves donde desaparecen detalles innecesarios, para ello se aplica la función GaussianBlur:

```
cv2.GaussianBlur(IMAGEN, (5, 5), 0)
```

Figura 26. Threshold de la imagen original

Los métodos del valor de umbral son un conjunto de técnicas y algoritmos que facilitan la segmentación de gráficos rasterizados, separa todos los objetos de interés de la imagen del resto de elementos como se evidencia en la figura 26. Open CV permite esto mediante su función **adaptiveThreshold.**

Esta técnica permite transformar la imagen a blanco y negro, se establece un valor base donde cada uno de los pixeles que lo superan se transforman en un color binario (binarización), y el resto en viceversa.

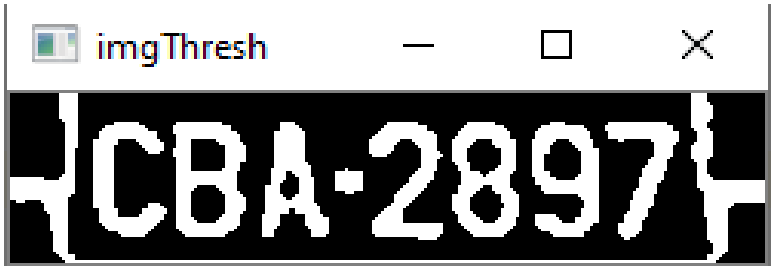


Figura 27. Threshold de la imagen agrandada

Para el caso de las matrículas de autos, como se evidencia en la figura 27, este proceso genera imágenes con contornos claramente definidos, donde se aprecian de mejor manera los caracteres mejorando así la tarea de segmentación

Los pasos siguientes a partir de esta imagen se enfocan en delimitar aún más el contorno de la posible matrícula, eliminando el fondo.

Figura 28. Delimitación de contornos

Los bordes con la curvatura que entrelazan a los conjuntos de pixeles adyacentes que constan de un mismo color, estos bordes actúan como una frontera que permite enfocar objetos de la imagen como se muestra en la figura 28.

Este paso es fundamental para detección, ya que permite reconocer eficientemente las formas en una imagen, este proceso es una de las etapas finales de la capa de procesamiento y se lleva a cabo utilizando las funcionalidades de la librería Numpy para delimitar con mayor precisión los contornos de la posible matrícula.

Para la segmentación de caracteres, es necesarios recortar y aumentar el tamaño del elemento encontrado:

La imagen en este estado está lista para la segmentación y reconocimiento real de caracteres. Posterior a esto se recorre cada uno de los caracteres encontrados para evaluar el más cercano mediante la función kNearest.findNearest.

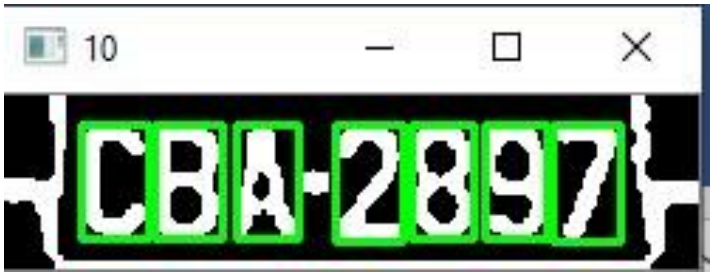


Figura 29. Segmentación y comparación de los caracteres.

La figura 29 muestra la manera en que la imagen queda finalmente segmentada. Al final se van agregando las coincidencias a una lista de caracteres para obtener la posible matrícula completa. Una vez recuperados los datos, se envían en una consulta REST a una Api en el sistema de la Universidad Nacional de Educación que contiene la información de pago del usuario, devolviendo un objeto Json con dicha información.

```
hdrs = {'User-Agent': 'Mozilla / 5.0 (X11 Linux x86_64) AppleWebKit / 537.36 (KHTML, like Gecko) Chrome'
                      ' / 52.0.2743.116 Safari / 537.36',
        'AUTHORIZATION': '[redacted]=='}
url = "https://sga.unae.edu.ec/check_access_parking/%s" % licPlate.strChars
solicitud = requests.get(url, headers=hdrs)
respuesta = solicitud.json()
return HttpResponse(json.dumps({'result': 'ok', 'mensaje': 'Placa encontrada %s ' % licPlate.strChars,
                                'respuesta': respuesta}), content_type="application/json")
```

Figura 30 Llamada a API de validación de acceso

Como se muestra en la Figura 30, al final del procesamiento este realiza una llamada a un servicio ubicado en otra de las plataformas de la Universidad, en que el que se solicita comprobar si la matrícula encontrada existe y cuenta con autorización para ingresar al parqueadero.

```
def datos_matricula(request, dato):
    try:
        import base64
        from django.http import HttpResponse
        from sga.models import SolicitudParqueoUnae
        if not 'HTTP_AUTHORIZATION' in request.META:
            return HttpResponse(json.dumps({'result': 'bad', 'estado': '500',
                                            'mensaje': u'No ha ingresado la cave de autorización'}), mimetype="application/json")
        keyaccess = request.META['HTTP_AUTHORIZATION']
        keyaccess = base64.decodestring(keyaccess)
        if keyaccess != KEY_ACCESS_API:
            return HttpResponse(json.dumps({'result': 'bad', 'estado': '500',
                                            'mensaje': u'KeyAccess Incorrecto'}), mimetype="application/json")
        if SolicitudParqueoUnae.objects.filter(vehiculosolicitud__placa=dato).exists():
            solicitud = SolicitudParqueoUnae.objects.filter(vehiculosolicitud__placa=dato)[0]
            return HttpResponse(json.dumps({'result': 'ok', 'estado': '200', 'usuario': u'%s' % solicitud.solicita,
                                            'modelo': u'%s' % solicitud.mi_vehiculo().modelo,
                                            'marca': u'%s' % solicitud.mi_vehiculo().marca,
                                            'tiene_acceso': u'%s' % solicitud.pagado_mes(datetime.now().month)}), mimetype="application/json")
        return HttpResponse(json.dumps({'result': 'ok', 'estado': '400',
                                        'mensaje': u'El vehículo %s no se encuentra registrado en el sistema' % dato}), mimetype="application/json")
    except Exception as ex:
        return bad_json(mensaje=ex)
```

Figura 31 API de validación de acceso

En la Figura 31 se evidencia la manera en que el servicio web recepta, valida y responde a la petición realizada, presentando los escenarios de que el vehículo existe, cuenta o no con acceso, o si el mismo no se encuentra registrado en el sistema.

5. Validación de la arquitectura propuesta

En primera instancia se comprueba el funcionamiento del API de validación de matrículas enviando una serie de parámetros.

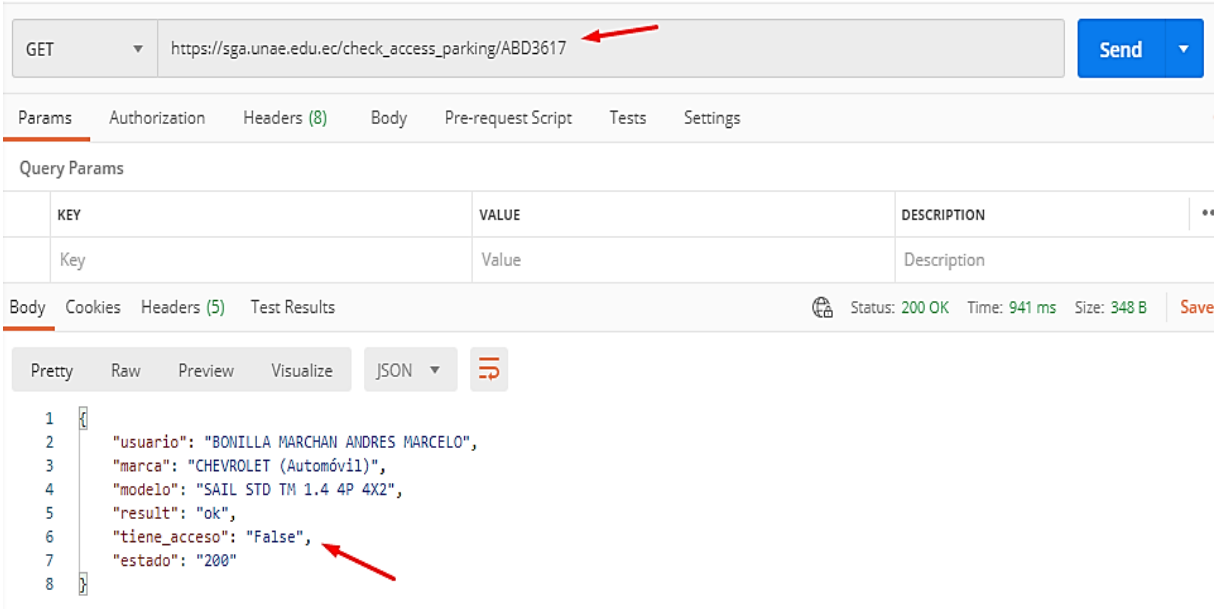


Figura 32 Comprobación de API en herramienta Postman

En la Figura 32 se muestra la respuesta obtenida en la prueba realizada a la API, se observa los datos que la misma devuelve como el usuario, marca y modelo de vehículo y si este cuenta con acceso al parqueadero.

Se realizaron diferentes pruebas (Ver Anexo 7) para comprobar el funcionamiento de la solución propuesta. La interfaz realizada efectúa llamadas asíncronas mediante Ajax para repetir este proceso de manera sistemática, de manera que el usuario vea la información en tiempo real.

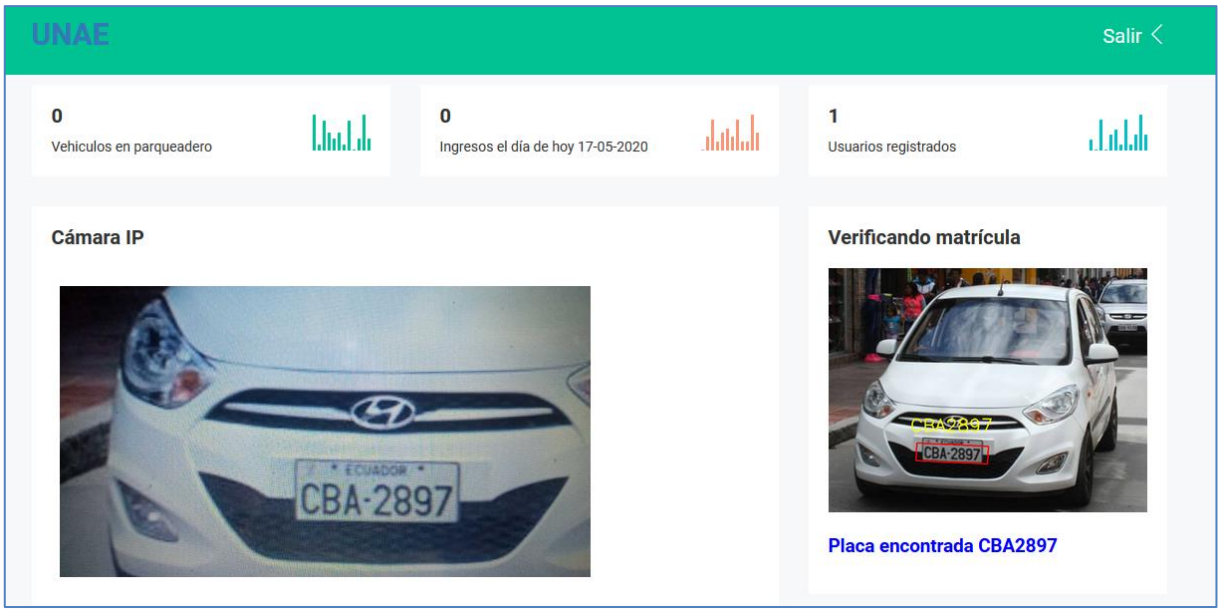


Figura 33. Prototipo de la interfaz principal del sistema

El objeto Json devuelto informa si el usuario ha realizado el pago correspondiente al mes en curso, lo que define si puede o no ingresar al parqueadero como se refleja en la Figura 33.

Para comprobar la veracidad del método escogido, se realizaron pruebas con cinco matrículas de vehículos, y se compara los datos obtenidos en relación a la metodología Tesseract.

Tabla 4. Resultados obtenidos

Resultados obtenidos con entrenamiento KNN y PyTesseract						
Matrícula	**Posibles coincidencias**		**Errores**		**Matrícula encontrada**	
	KNN	**PYTESS**	**KNN**	**PYTESS**	**KNN**	**PYTESS**
CBA-2897	8	7	1	2	CBA2897	CBA2897
UBA-4426	7	7	1	1	UBA4426	UBA4426
ABD-3617	10	11	2	4	ABD3617	ABD3617
GSU-6858	8	10	2	2	GSUG858	OSU6858
ABG-7529	7	7	1	2	ABG7529	ABO7529
UEI-1325	10	12	2	4	UEI1325	VEI325
PAA-9954	12	10	0	1	PAA9954	PAA9954

Elaboración propia del autor

En la Tabla 4 se especifican los resultados obtenidos de las pruebas realizadas con dos diferentes librerías. Si bien Pytesseract es una librería eficiente, el entrenamiento KNN presenta una mayor exactitud en el reconocimiento de caracteres, la principal diferencia encontrada se manifiesta al momento de trabajar con matrículas poco legibles, en este aspecto, uno de los errores más comunes se presenta en letras como la G donde Tesseract la detecta como una O.

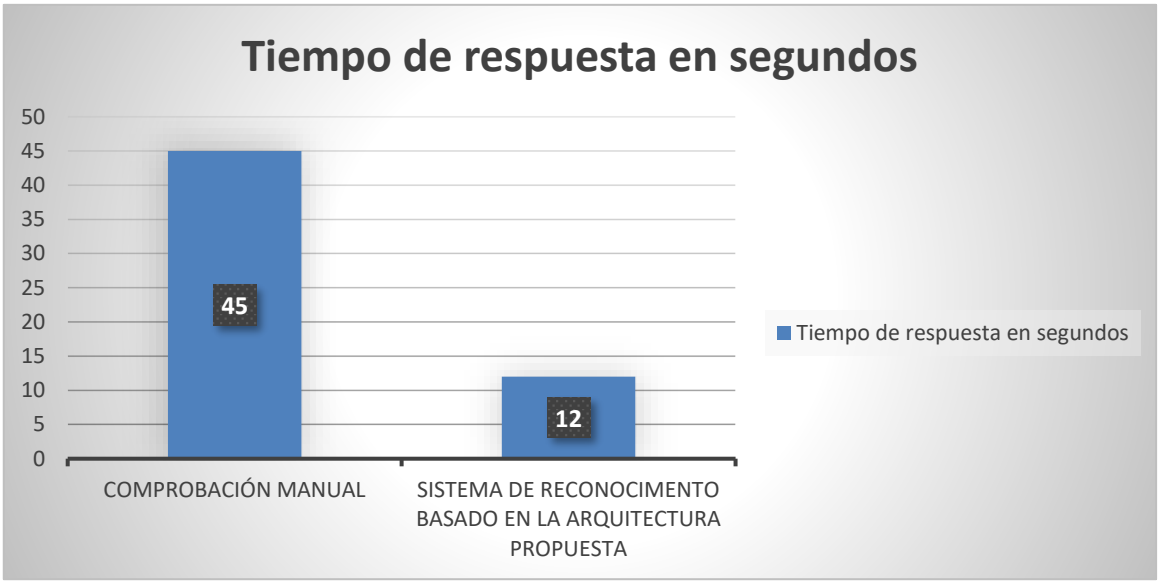


Figura 34 Comparación de los tiempos de respuesta

En la figura 34 se muestra una gráfica que denota cómo el proceso de verificación de matrículas se ve mejorado en función al tiempo; existe una diferencia de más de 30 segundos en la comprobación de un vehículo, que al exponer frente a los más de 200 que ingresan a diario se estima una rebaja sustancial en el tiempo de respuesta. Estos hechos son evidenciados en la encuesta aplicada a dos grupos específicos, arrojando los siguientes resultados.

Encuesta a Guardias de seguridad encargados del parqueadero:

1 ¿La interfaz gráfica presentada para el sistema le resulta amigable?

Tabla 5. Resultados obtenidos pregunta 1

Alternativa	Respuestas	Porcentaje
Si	5	100%
No	0	0%
Total	**5**	**100%**

Elaboración propia del autor

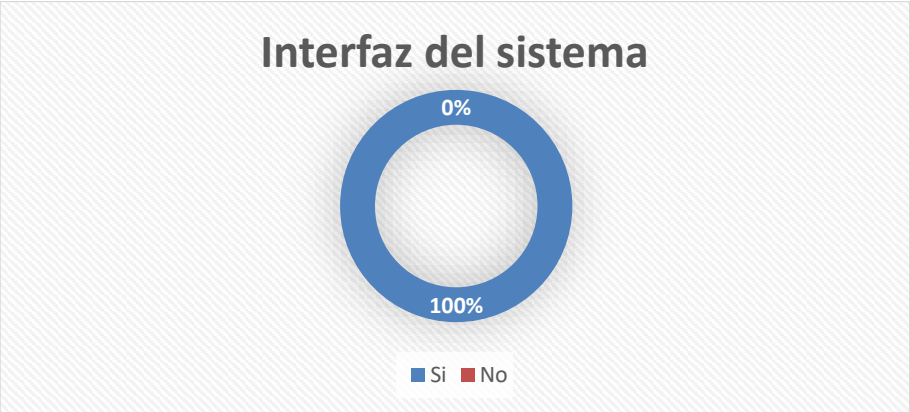


Figura 35. Gráfico pregunta 1 de la encuesta

Los cinco responsables encuestados coincidieron en afirmar que la interfaz propuesta para el monitoreo de los vehículos que ingresan al parqueadero presenta un diseño amigable y resulta intuitiva para ellos como refleja los resultados de la figura 35.

2 en una escala de 1 a 5, siendo 1 nada eficiente y 5 muy eficiente ¿Cuál es nivel de eficiencia del sistema detector de matrículas?

Tabla 6. Resultados obtenidos pregunta 2

Alternativa	Respuestas	Porcentaje
5	4	80%
4	1	20%
3	0	0%
2	0	0%
1	0	0%
Total	**5**	**100%**

Elaboración propia del autor

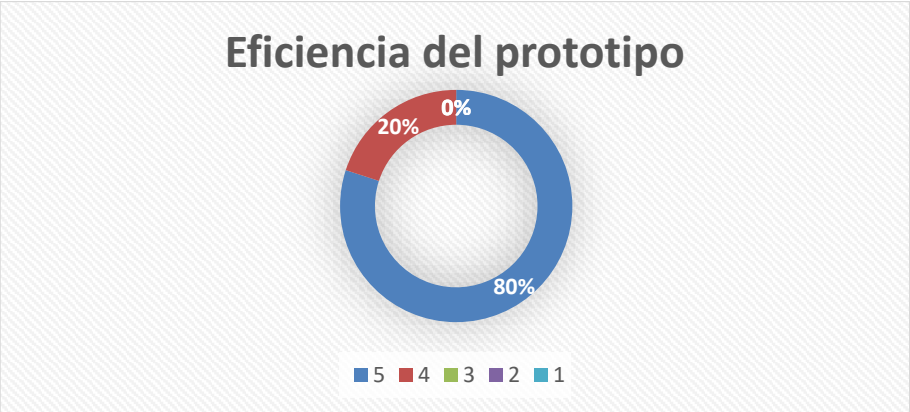


Figura 36. Gráfico pregunta 2 de la encuesta

Tanto en la tabla 6 como en la figura 36 se muestra que, en cuanto a la eficiencia del prototipo al momento de la detección y evaluación de la matrícula, el 80% de los encuestados coincidieron en un nivel de 5 al considerar que el prototipo cumple con las expectativas, el 20% restante optó por un nivel de 4.

3 ¿Cuál es el tiempo promedio que le tomaba la tarea de comprobar el acceso al parqueadero sin la integración del sistema?

Tabla 7. Resultados obtenidos pregunta 3

Alternativa	Respuestas	Porcentaje
Entre 5 y 15 segundos	0	0%
Entre 15 y 30 segundos	0	0%
Entre 30 y 45 segundos	1	20%
Más de 45 segundos	4	80%
Total	**5**	**100%**

Elaboración propia del autor

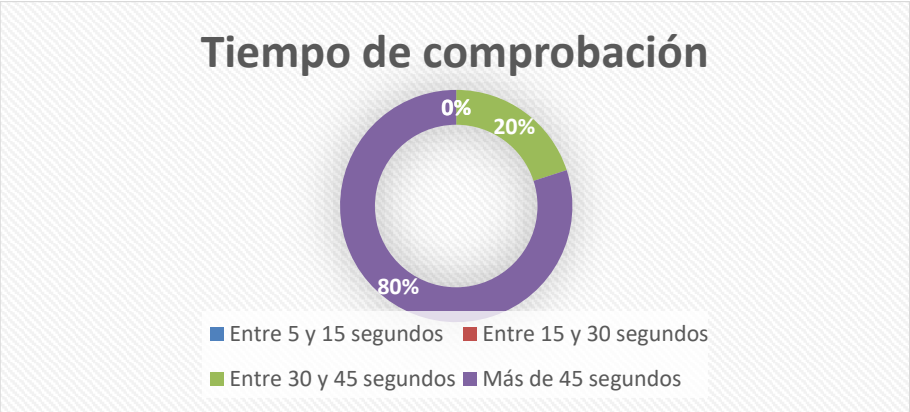


Figura 37. Gráfico pregunta 3 de la encuesta

En la figura 37 se evidencia que, para los encuestados, la implementación final del prototipo tendrá un alto impacto en la labor que estos realizan, ya que la prueba realizada demuestra que el tiempo promedio en que el sistema tarda en realizar la evaluación es 15 segundos, por su parte, los encuestados en su mayoría afirman que a ellos les tomaba más de 45 segundos.

Encuesta a usuarios del parqueadero:

4 ¿Cuántas veces al día ingresa al parqueadero de la Universidad Nacional de Educación?

Tabla 8. Resultados obtenidos pregunta 4

Alternativa	Respuestas	Porcentaje
1 vez	50	36%
2 veces	85	61%
3 veces	0	0%
4 veces	0	0%
5 o más	5	3%
Total	**140**	**100%**

Elaboración propia del autor

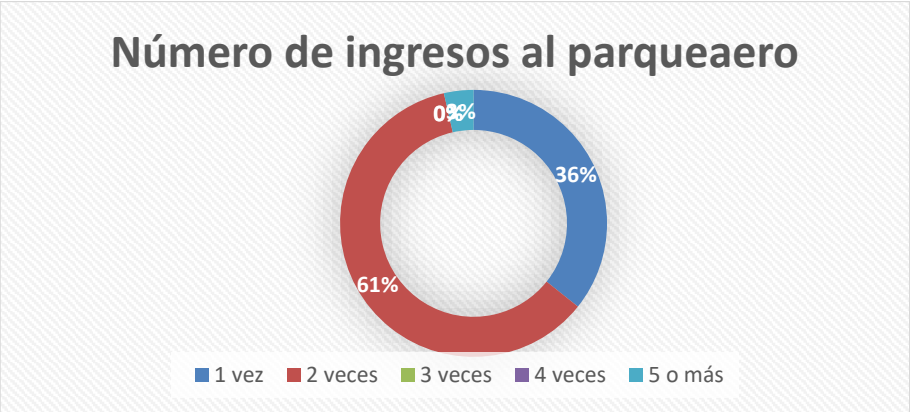


Figura 38. Gráfico pregunta 4 de la encuesta

Esta pregunta tiene la finalidad de determinar la cantidad de interacciones promedio que tendría el prototipo en el día, de las 140 personas evaluadas, 5 que corresponde al 3% indican que deben ingresar 5 o más veces en el día, por su parte un 36% afirma que lo realiza 1 sola vez al día, que es el factor común en alumnos, por su parte el 61% restante lo realiza dos veces al día.

Se puede determinar de estos resultados que la mayoría de las personas realizan su ingreso al parqueadero dos veces al día por lo que es necesario realizar mantenimientos periódicos al sistema propuesto.

5 ¿Ha tenido problemas alguna vez al momento de ingresar al parqueadero antes de la implementación del sistema?

Tabla 9. Resultados obtenidos pregunta 5

Alternativa	Respuestas	Porcentaje
Si	43	36%
No	97	61%
Total	**140**	**100%**

Elaboración propia del autor

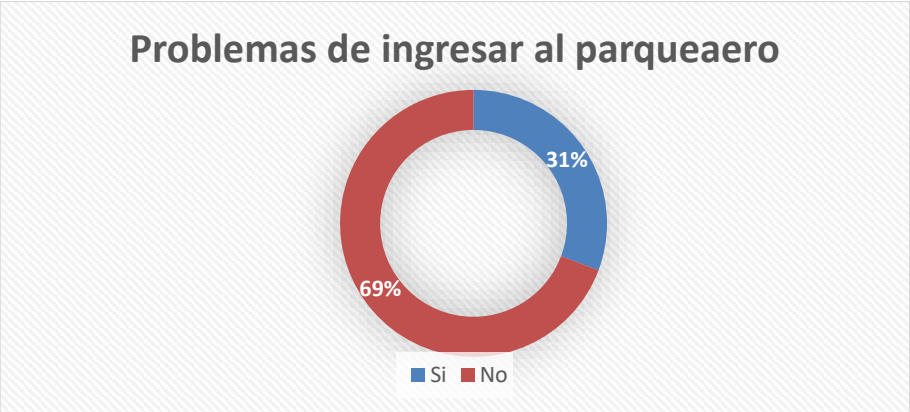


Figura 39. Gráfico pregunta 5 de la encuesta

EN la figura 39 se evidencia que el 69% de usuarios encuestados afirman no haber tenido mayores problemas al momento de ingresar al parqueadero, sin embargo, un alto porcentaje, 31% afirman haberlos tenido.

La mayoría de estos problemas eran originados por errores que cometían los guardias de seguridad, al revisar de manera manual los informes que el departamento de colecturía entregaba para determinar qué persona tenía acceso al parqueadero, dicho inconveniente es una de las principales motivaciones para la implementación del prototipo.

6 posterior a las pruebas realizadas con el prototipo ¿Considera usted que se ha agilizado el ingreso al parqueadero?

Tabla 10. Resultados obtenidos pregunta 6

Alternativa	Respuestas	Porcentaje
Si	124	89%
No	16	11%
Total	**140**	**100%**

Elaboración propia del autor

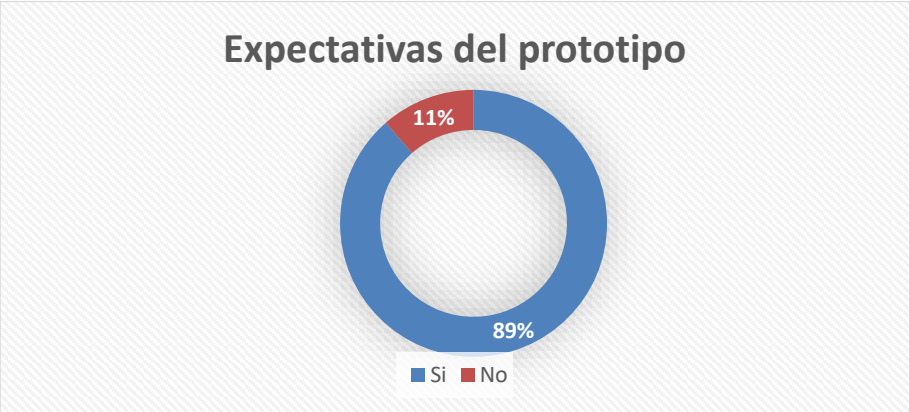


Figura 40. Gráfico pregunta 6 de la encuesta

De los 140 usuarios encuestados, 124 que corresponde al 89% se muestran conformes y afirman que, si observan mejoría en el proceso de ingreso al parqueadero, por su parte una minoría correspondiente al 11% considera que aún existen aspectos a mejorar como se evidencia en la figura 40.

En conclusión, las pruebas con el sistema fueron satisfactorias y demuestran la viabilidad del mismo al mejorar el proceso de ingreso al parqueado minimizando el tiempo que este tomaba, también, es un aporte para los guardias de seguridad encargados de dicha validación que ya no deben evaluar documentos físicos.

6. Conclusiones y trabajo futuro

6.1. Conclusiones

Finalizado el desarrollo y las pruebas del prototipo de detección y validación de matrículas, se obtienen las conclusiones que a continuación se detallan.

La arquitectura de software propuesta integra de manera eficiente diferentes dispositivos y plataformas, agilizando la gestión de procesos al reducir los tiempos de ejecución y aumentar el nivel de eficiencia y exactitud.

Las tecnologías como REST y Json presentan factores determinantes que las convirtieron en soluciones ideales para cumplir con el objetivo de la propuesta debido a que las plataformas de la Universidad Nacional de Educación se basan en Python, la utilización de REST no presenta ninguna objeción, la utilización de la librería requests de Python presenta los niveles de seguridad y efectividad óptimos para las peticiones y respuestas de información necesaria para el funcionamiento del prototipo.

Las capas integradas en la arquitectura propuesta garantizan un nivel de seguridad y eficiencia óptimos, la recolección de datos se realiza de manera coherente, la capa de procesamiento da el tratamiento necesario a las imágenes para maximizar el nivel de eficiencia al momento de la detección de caracteres, por su parte la capa cloud responde sin problemas cada una de las peticiones que realizan durante el proceso de validación.

La utilización de librerías como OpenCV y NumPY facilitan en gran medida el trabajo de recolección, clasificación y detección de caracteres en imágenes. Operaciones y procesos complejos como el desenfoque, detección de contornos necesarios para alcanzar el máximo rendimiento, se reducen a llamadas a funciones y clases dirigiendo los esfuerzos al diseño y optimización del prototipo.

Se constata la importancia de contar con un método de entrenamiento, el apartado teórico se exponen los métodos de redes neuronales convolucionales y el entrenamiento con algoritmos KNN, esta segunda opción se presentó como la indicada, ya que la librería base del proyecto (OpenCV) cuenta con funciones propias para realizar entrenamientos relacionados a la detección de caracteres.

La interfaz implementada en el caso de estudio utilizado para validar la arquitectura propuesta es sencilla y fácil de interpretar para los encargados, esto simplifica el proceso de validación agilizando por completo el ingreso al parqueadero por parte de los usuarios que,

según los resultados de la encuesta aplicada, en su gran mayoría consideran que el prototipo tiene un impacto positivo.

Las preguntas de la encuesta realizada tanto a usuarios como a guardias de seguridad reflejan las expectativas y el nivel de eficiencia que ha tenido el prototipo en las pruebas realizadas.

La arquitectura propuesta cumple con las expectativas, la misma cuenta con los niveles de seguridad y rendimiento esperados, es escalable y permite la fácil modificación o integración de elementos sin afectar su rendimiento; lo que permite que pueda ser implementada en otras áreas siempre que se tengan en cuenta las condiciones específicas dadas por el contexto de aplicación.

6.2. Líneas de trabajo futuro

Al finalizar las pruebas y validación del prototipo, se detallan diferentes vías de trabajo futuro que permitirían mejorar la arquitectura y por consecuencia el sistema detección de matrículas.

- **Extender el uso de la arquitectura propuesta**

Con la validación realizada a la arquitectura se demuestra la eficiencia de la misma, por lo que es posible su implementación en otros procesos de la universidad que requieran la integración de dispositivos inteligentes. Un ejemplo de esto, es el control de asistencia de empleados que se realiza mediante un dispositivo biométrico o la automatización de los sistemas de riego del campus donde intervendría el uso de sensores y actuadores.

- **Optimizar el sistema de captación**

Es posible ampliar el proceso de detección de imágenes, de momento, se utiliza una cámara IP sencilla de la que se toman los datos de manera directa, la calibración es realizada en el servidor que procesa las imágenes. Sería importante añadir un módulo previo al procesamiento de imágenes que se encargue de localizar dinámicamente las matrículas de los vehículos, maximizando así el rendimiento de la aplicación. Para esto se podría incluir la utilización de Arduino o Raspberry PI.

- **Minimizar la interacción del agente humano**

Arduino y Raspberry PI podrían ser utilizados también en este aspecto, de momento si bien el prototipo propuesto minimiza la interacción de los guardias de seguridad, esto se

puede mejorar con la implementación de un brazo controlado por una matrícula, que se levante de manera automática cuando el sistema detector valide la matrícula del auto que desea ingresar.

Este paso terminaría de automatizar por completo el proceso de ingreso de vehículos al parqueadero liberando a los guardias de seguridad de esta tarea.

7. Bibliografía

Álvarez, M. (2014). *Análisis, diseño e implementación de un sistema de control de ingreso de vehículos basado en visión artificial y reconocimiento de placas en el parqueadero de la Universidad Politécnica Salesiana - Sede Cuenca.* Cuenca: Universidad Politécnica Salesiana. Obtenido de https://dspace.ups.edu.ec/bitstream/123456789/7060/1/UPS-CT003790.pdf

Alvear, Rosero, Peluffo, & Pijal. (2017). Internet de las Cosas y Visión Artificial, Funcionamiento y Aplicaciones: Revisión de Literatura. *Enfoque UTE, 7*, 244-256. doi:10.29019/enfoqueute.v8n1.121

Aperador, W., Bautista, J., & Mejía, A. (2013). Determinación por Visión Artificial del Factor de Degradación en Aleaciones Biocompatibles. *Información Tecnológica, 24*(2), 109-120. doi:10.4067/S0718-07642013000200012

Bernardis, H., Bernardis, E., Berón, M., Riesco, D. E., & Pereira, M. J. (2018). Técnicas y herramientas para regular la seguridad en web services basados en WSDL. *XX Workshop de Investigadores en Ciencias de la Computaci´on*, 1051-1055. Obtenido de http://sedici.unlp.edu.ar/bitstream/handle/10915/68353/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y

Betancor, A. (2008). *Sistema De Reconocimiento de Matrículas Basado en Visión Artificial para Control de Acceso.* Cartagena: Universidad Politécnica de Cartagena. Obtenido de https://core.ac.uk/download/pdf/60416014.pdf

Bruneo, D., Distefano, S., Giacobbe, M., Longo, A., Longo, F., Merlino, G., . . . Tapas, N. (2019). An IoT service ecosystem for Smart Cities: #SmartMe project. *Internet of Things*, 12-33. doi:10.1016/j.iot.2018.11.004

Campo, W. Y., Chanchí, G. E., & Arciniegas, J. L. (2013). Arquitectura de Software para el Soporte de Comunidades Académicas Virtuales en Ambientes de Televisión Digital Interactiva. *Formación Universitaria , 6*(2), 3-14. doi:10.4067/S0718-50062013000200002

Campos, A. (2014). *Desarrollo de software de reconocimiento de matrículas de coche.* Gandia: Universidad Politécnica de Valencia. Obtenido de https://riunet.upv.es/bitstream/handle/10251/46232/Memoria.pdf?sequence=1

Chanchí, Arciniegas, & Campo. (2016). Construcción y evaluación de servicios interactivos en entornos de TVDi. *Revista: Ingeniería, 21*(1), 63 - 82. doi:10.14483/udistrital.jour.reving.2016.1.a05

Cobos. (2016). *Diseño e implementación de una arquitectura IoT basada en tecnologías Open Source.* Sevilla: Universidad de Sevilla. Retrieved from http://bibing.us.es/proyectos/abreproy/70884/fichero/TFM-Antonio+Cobos+Dominguez.pdf

Din, I. U., Guizani, M., Hassan, S., Kim, B.-S., Khan, M. K., Atiquzzaman, M., & Ahmed, S. H. (2019). The Internet of Things: A Review of Enabled Technologies and Future Challenges. *IEEE Access, 7*, 7606-7640. doi:http://dx.doi.org/10.1109/ACCESS.2018.2886601

García, O. C., Prieto, J. C., & Fisteus, J. A. (2006). Servicios web: Introducción y estado del arte. *Novática: Revista De La Asociación De Técnicos De Informática, 183*, 6-10. Obtenido de https://dialnet.unirioja.es/servlet/articulo?codigo=2206925

González, H., Santos, G., Campos, F., & Morell, C. (2016). Evaluación del algoritmo KNN-SP para problemas de predicción con salidas compuestas. *Revista Cubana de Ciencias Informáticas, 10*(3), 119 - 129. Obtenido de http://scielo.sld.cu/pdf/rcci/v10n3/rcci09316.pdf

Guang, C., Tonghai, J., Meng, W., Xinyu, T., & Wenfei, J. (2020). Modeling and reasoning of IoT architecture in semantic ontology dimension. *Computer Communications*, 580-594. doi:10.1016/j.comcom.2020.02.006

Hadj, M., Ghozzi, F., & Chaari, L. (2019). A New Architecture for Cognitive Internet of Things and Big Data. *Procedia Computer Science*, 534–543. Obtenido de 10.1016/j.procs.2019.09.208

IEEE. (2020). *IEEE SA STANDARDS ASSOCIATION*. Obtenido de P2413.2 - Standard for a Reference Architecture for Power Distribution IoT (PDIoT): https://standards.ieee.org/project/2413_2.html

ITU. (2020). *ITU Committed to connecting the world*. Obtenido de Internet of Things Global Standards Initiative: https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx

Jeon, B., & Suh, S. H. (2018). Design Considerations and Architecture for Cooperative Smart Factory: MAPE/BD Approach. *Procedia manufacturing*, 1094-1106. doi:10.1016/j.promfg.2018.07.146

Kiran, Ganta, Krishna, & Praveen. (2020). A Novel Method for Indian Vehicle Registration Number Plate Detection and Recognition using Image Processing Techniques. *Procedia Computer Sciencie*, 2623–2633. doi:10.1016/j.procs.2020.03.324

Kurebwa, J. G., & Mushiri, T. (2019). Internet of things architecture for a smart passenger-car robotic first aid system. *Procedia Manufacturing*, 27-34. doi:10.1016/j.promfg.2019.05.006

León, M. C., Boixader, F. R., & Luque, E. (2014). Arquitectura orientada a servicios: Un enfoque basado en proyectos. *Enseñanza Y Aprendizaje De Ingeniería De Computadores: Revista De Experiencias Docentes En Ingeniería De Computadores, 4*, 103 - 114. Obtenido de https://dialnet.unirioja.es/servlet/articulo?codigo=5396702

Loaiza, A., Manzano, D., & Múnera, L. (2012). Sistema de visión artificial para conteo de objetos en movimiento. *El Hombre y la Máquina*(40), 87-101. Retrieved from https://www.redalyc.org/pdf/478/47826850010.pdf

Lugo, J. d., Ton-Sieras, F., & García, Á. (2016). PlateR sistema de reconocimiento de placas y acceso vehicular automático. *Revista de Prototipos Tecnológicos, 2*(5), 1-7. Obtenido de http://www.ecorfan.org/spain/researchjournals/Prototipos_Tecnologicos/vol2num5/Revista_de_Prototipos_Tecnologicos_V2_N5_1.pdf

Mendoza, J., Ordóñez, H., Ordóñez, A., & Jurado, J. (2017). Architecture for embedded software in microcontrollers for Internet. *Procedia Computer Science*, 1092–1097. doi:10.1016/j.procs.2017.05.395

Molina, J. P. (2016). *Restful framework for collaborative internet of things based on iec 61850.* Leioa, BI, España: Universidad del País Vasco. Obtenido de https://dialnet.unirioja.es/servlet/tesis?codigo=212647

Muñoz, R. (2014). *Sistema de visión artificial para la detección y lectura de matrículas.* Valladolid: Universidad de Valladolid. Retrieved from https://uvadoc.uva.es/bitstream/handle/10324/11848/TFG-P-165.pdf;jsessionid=C25A6B52B0CC981CAE93C4A333EEA9F1?sequence=1

Reddy, S. R., & Shareef, Z. (2018). Intel Processor Architectural and Integrated Development Environment Exploration. *i-Manager's Journal on Embedded Systems, 6*(2), 10. doi:10.26634/jes.6.2.14757

Sanabria S., J. J., & Archila D., J. F. (2011). Detección y análisis de movimiento usando visión artificial. *Scientia Et Technica, 16*(49), 180-188. Obtenido de https://www.redalyc.org/pdf/849/84922625031.pdf

Sayago, J. P., Flores, E. L., & Recalde, A. (2019). Análisis Comparativo entre los Estándares Orientados a Servicios Web SOAP, REST y GRAPHQL. *Revista Antioqueña de las Ciencias Computacionales y la Ingeniería de Software (RACCIS)*, 10-22. doi:10.5281/zenodo.3592004

Suárez, A. C., Ávila, S. S., & Páez, M. Á. (2019). Mecanismos de seguridad en el internet de las cosas. *Revista vínculos, 16*(2). doi:10.14483/2322939X.15758

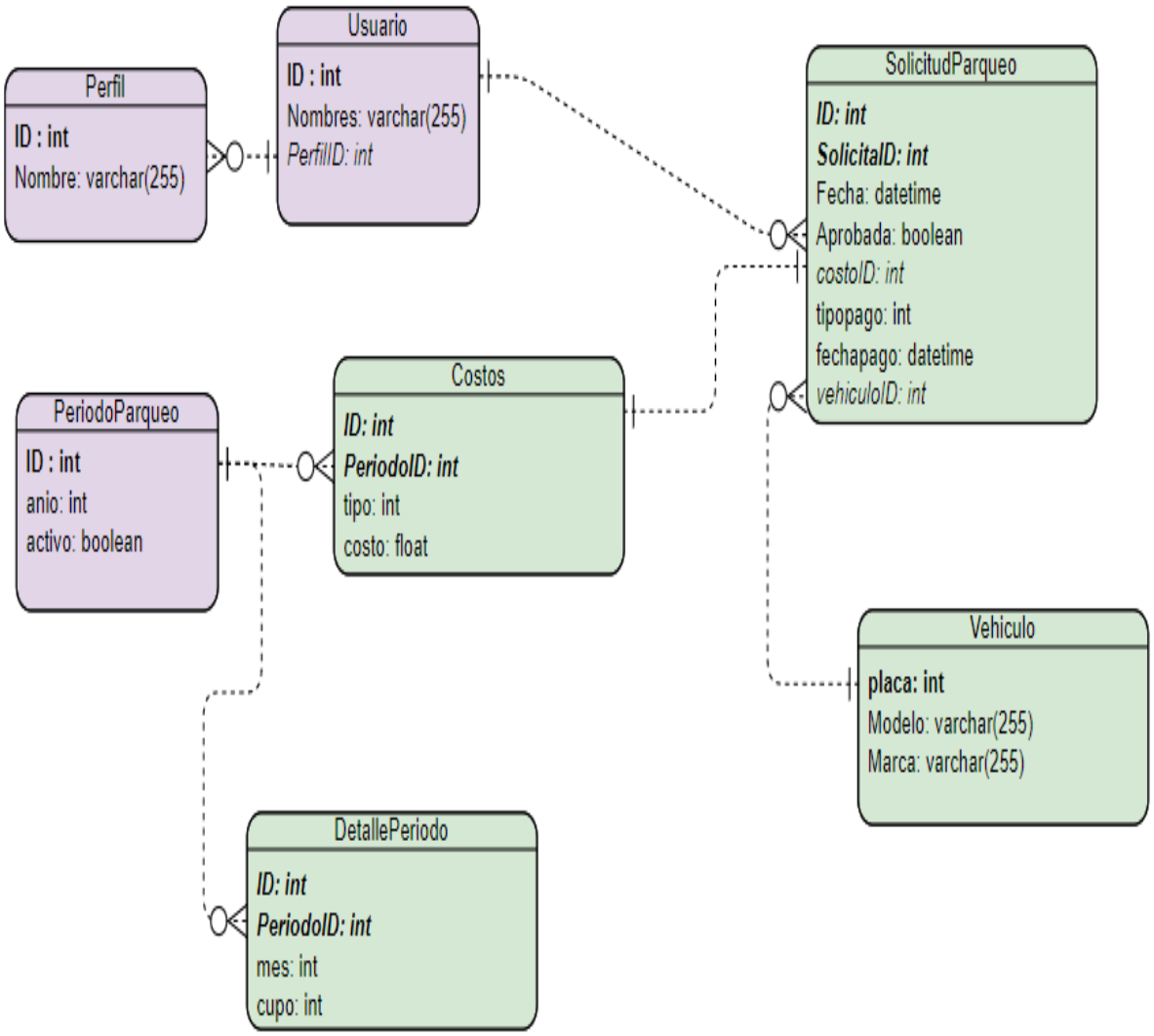
Verdouw, C., Sudmaeker, H., Tekinerdogan, B., Conzon, D., & Montanaro, T. (2019). Architecture framework of IoT-based food and farm systems: A multiple case. *Computers and Electronics in Agriculture*, 1-26. Obtenido de 10.1016/j.compag.2019.104939

W3C. (22 de 4 de 2020). *Web Services Architecture*. Obtenido de https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#concepts

Zito, M. (2018). La sustentabilidad de Internet de las Cosas. *Cuadernos del Centro de Estudios en Diseño y Comunicación. Ensayos*(70), 37-44. doi:https://doi.org/10.18682/cdc.vi70

Anexos

Anexo 1 Base de datos del sistema de solicitudes de parqueo de la UAE



Anexo 2 Servicio de autenticación centralizado

```
@transaction.commit_on_success
def autenticacion_sistema_alor_unae(request):
    from django.contrib.auth import authenticate, logout, login
    try:
        data = {}
        import base64
        parametro = request.META['HTTP_AUTHORIZATION']
        dividir = parametro.split('Basic ')
        credenciales = base64.decodestring(dividir[1])
        dividir = credenciales.split(':')
        usuario = dividir[0]
        contrasena = dividir[1]

        from Crypto.Cipher import AES
        import base64

        #encripta
        MasterSecretKey = 'aOkwICRqcSIELcHg'
        byte_array = base64.b64decode(contrasena)
        iv = byte_array[0:16]  # extract the 16-byte initialization vector
        messagebytes = byte_array[16:]  # encrypted message is the bit after the

        cipher = AES.new(MasterSecretKey.encode("UTF-8"), AES.MODE_CBC, iv)
        decrypted_padded = cipher.decrypt(messagebytes)
        plain_psw = decrypted_padded.strip().replace('','')
        plain_psw = str(str(plain_psw.strip())
        plain_psw = plain_psw.strip().replace('', '')
        a = ''.join(e for e in plain_psw.strip() if e.isalnum())

        from collections import OrderedDict
        user = authenticate(username=usuario, password=a)

        if user is not None:
            if not user.is_active:
                return HttpResponse(json.dumps('result': 'bad'
'estado': '400'), mimetype="application/json",status=status)
            else:
                if Persona.objects.filter(usuario=user).exists():
                    persona = Persona.objects.filter(usuario=user)[0]
                data= OrderedDict()
                data["id"] = user.username
                att = {}
                att = OrderedDict()
                att["attr1"] = user.username
                att["attr2"] = persona.nombres
                att["attr3"] = persona.apellido1
                att["attr4"] = persona.emailinst
                att["attr6"] = user.id
                data["attributes"] = att
                login(request, user)
                return ok_json(data)
        else:
            return HttpResponse(json.dumps('result': 'ok'
'estado': '200'), mimetype="application/json",status=status)
    except Exception as ex:
        return HttpResponse(json.dumps('result': 'bad'
'estado': '400'), mimetype="application/json",status=status)
```

Anexo 3 Clases que intervienen en el proceso de detección

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

    # constructor
###################################################################################
    def __init__(self, _contornos):
        self.contornos = _contornos

        self.boundingRect = cv2.boundingRect(self.contornos)

        [intX, intY, intW, intH] = self.boundingRect

        self.intBoundingRectX = intX
        self.intBoundingRectY = intY
        self.intBoundingRectWidth = intW
        self.intBoundingRectHeight = intH

        self.intBoundingRectArea = self.intBoundingRectWidth *
self.intBoundingRectHeight

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

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

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


###################################################################################
#################
class PosibleElemento:

    # constructor
###################################################################################
    def __init__(self):
        self.imgPlate = None
        self.imgGrayscale = None
        self.imgThresh = None

        self.rrLocationOfPlateInScene = None

        self.strChars = ""
    # end constructor
```

Anexo 4 Captación de imágenes desde cámara IP

```
cap  = cv2.VideoCapture("http://193.0.0.100:8080/videofeed")
ret, imgOriginalScene = cap.read()
#imgOriginalScene = cv2.imread(folder + "\\" + "prueba2.jpg")
if imgOriginalScene is None:
    print("\nerror: no se ha leido ninguna imagen\n\n")
    return HttpResponse(json.dumps({'result': 'bad', 'estado': '400', 'mensaje':
'No se ha podido leer imagen'}),
                        content_type="application/json")

listOfPossiblePlates = Recognition.detectarPlaca(imgOriginalScene)          #
detectar placa

listOfPossiblePlates = DetectaCaracteres.detectCharsInPlates(listOfPossiblePlates)
# detectar caracteres en placa
```

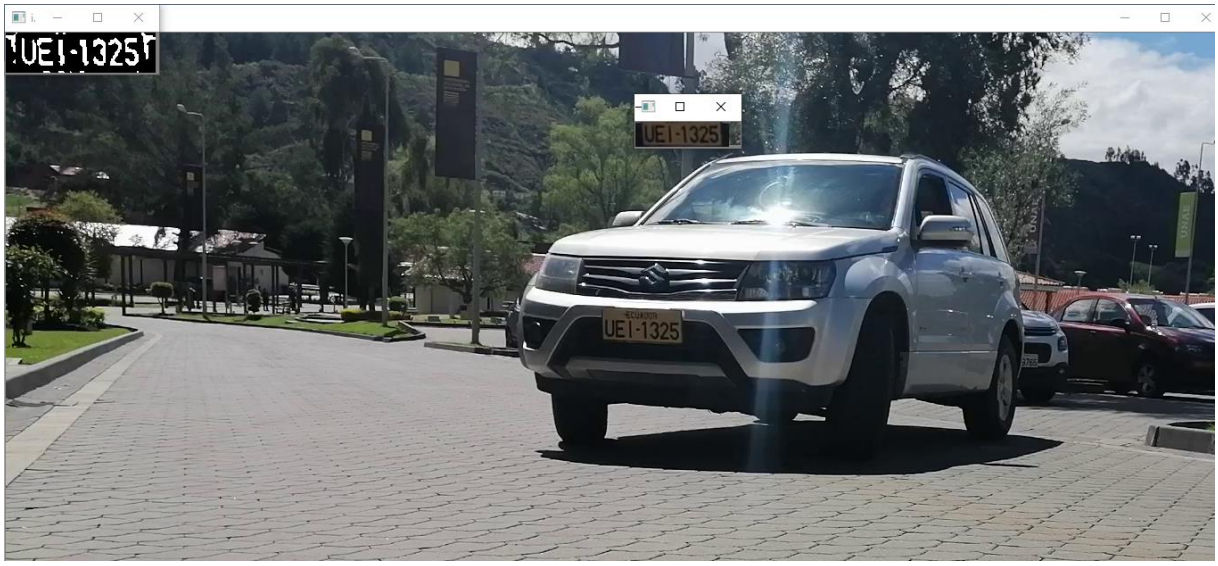
Anexo 5 Entrenamiento de detección KNN

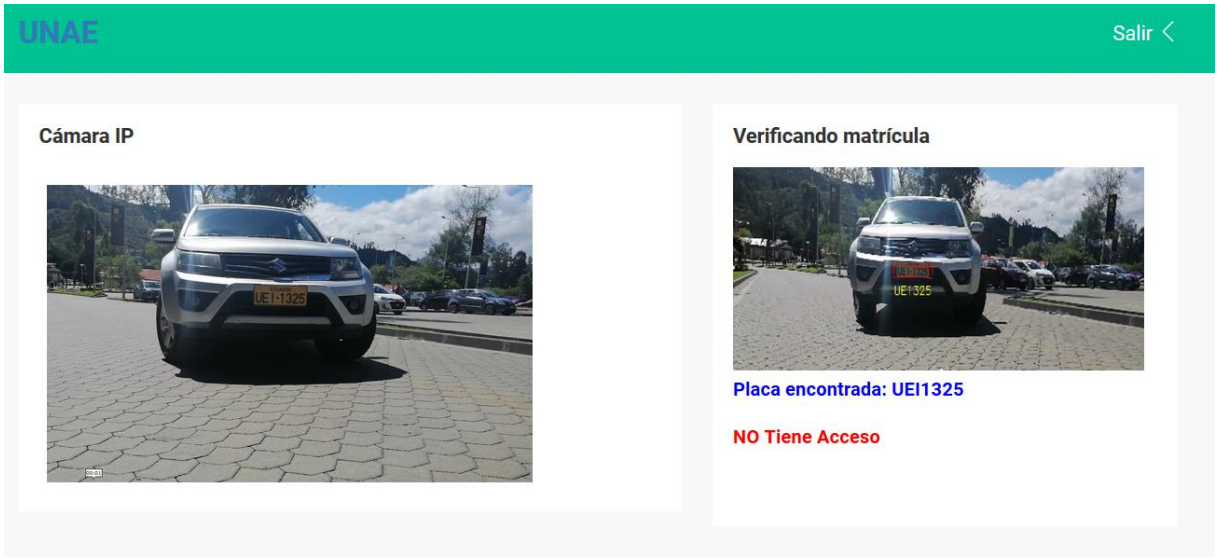
```
###################################################################################
def datosKNN():
    allContoursWithData = []                # listas vacias para llenar luego
    validContoursWithData = []

    try:
        npaClassifications =
np.loadtxt("D:\TFM\sistema_alpr\placa\classifications.txt", np.float32)
        # lee archivo de clasificaciones
    except:
        print("error, no se puede abrir clasificaciones.txt, salir del
programa\n")
        os.system("pause")
        return False
    # end try

    try:
        npaFlattenedImages =
np.loadtxt("D:\TFM\sistema_alpr\placa\lattened_images.txt", np.float32)
        # imágenes de entrenamiento
    except:
        print("error, no se puede abrir lattened_images.txt, al salir del
programa\n")
        os.system("pause")
        return False
    # end try

    npaClassifications = npaClassifications.reshape((npaClassifications.size, 1))
    # remodelar la matriz numpy a 1d, necesaria para pasar a la llamada para
entrenar

    kNearest.setDefaultK(1)

    kNearest.train(npaFlattenedImages, cv2.ml.ROW_SAMPLE, npaClassifications)
    # train KNN object

    return True
```

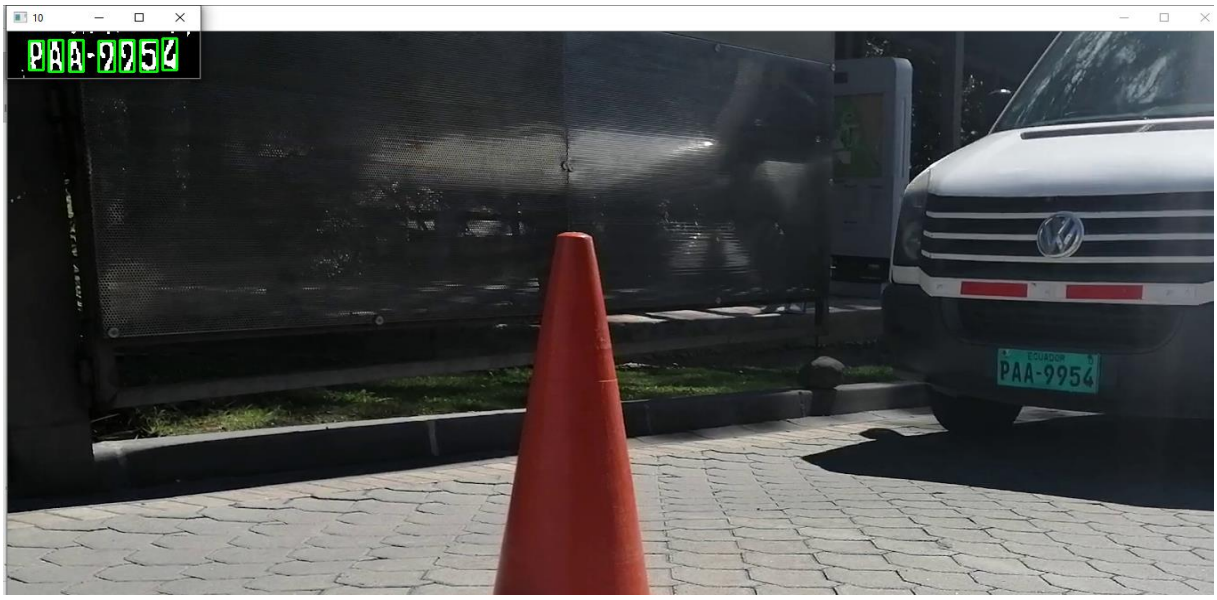
Anexo 6 Procesamiento de la imagen

```
###################################################################################
def procesar(imgOriginal):
    imgGrayscale = extraerValor(imgOriginal)

    imgMaxContrastGrayscale = maximizaContraste(imgGrayscale)

    height, width = imgGrayscale.shape

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

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

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

    return imgGrayscale, imgThresh


###################################################################################
def extraerValor(imgOriginal):
    h, w, canales = imgOriginal.shape

    imgHSV = np.zeros((h, w, 3), np.uint8)

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

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

    return imgValue


###################################################################################
def maximizaContraste(imgGrayscale):

    h, w = imgGrayscale.shape

    imgTopHat = np.zeros((h, w, 1), np.uint8)
    imgBlackHat = np.zeros((h, w, 1), np.uint8)

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

    imgTopHat = cv2.morphologyEx(imgGrayscale, cv2.MORPH_TOPHAT, Elemento)
    imgBlackHat = cv2.morphologyEx(imgGrayscale, cv2.MORPH_BLACKHAT, Elemento)

    imgGrayscalePlusTopHat = cv2.add(imgGrayscale, imgTopHat)
    imgGrayscalePlusTopHatMinusBlackHat = cv2.subtract(imgGrayscalePlusTopHat,
imgBlackHat)

    return imgGrayscalePlusTopHatMinusBlackHat
```

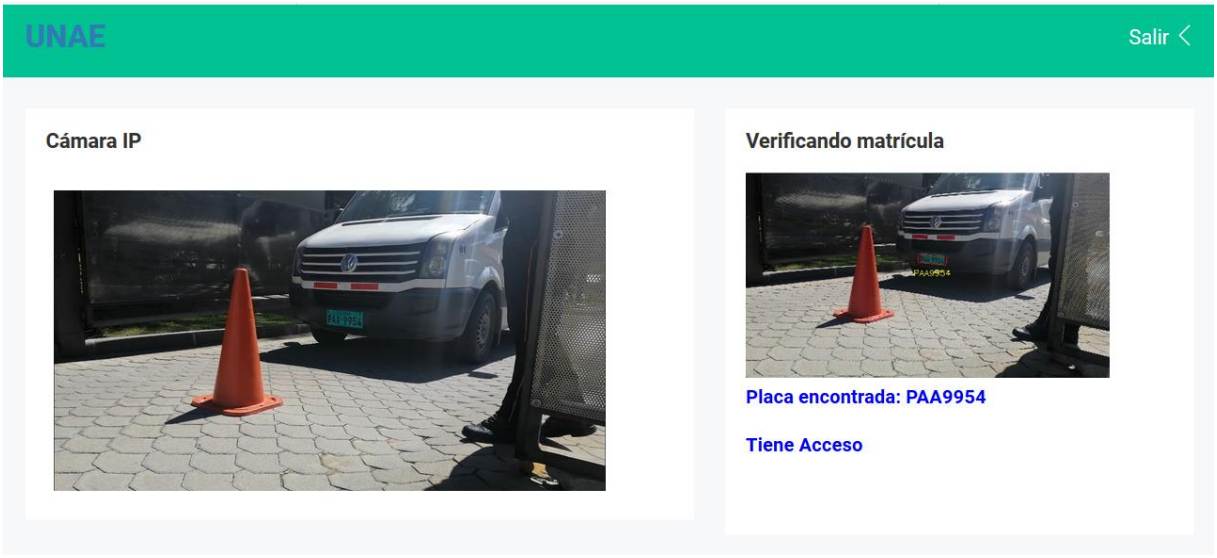
Anexo 7 Detalles de pruebas realizadas






10
PAA-9954
PAA-9954


PAA-9954
PAA9954


UNAE
Salir
Cámara IP
Verificando matrícula
Placa encontrada: PAA9954
Tiene Acceso