
**UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA**

**FACULTAD DE SISTEMAS Y TELECOMUNICACIONES**

**CARRERA DE TELECOMUNICACIONES**

# TRABAJO DE INTEGRACIÓN CURRICULAR

Previo a la obtención del título de:

**INGENIERO EN TELECOMUNICACIONES**

Desarrollo de un sistema de estacionamiento automatizado para vehículos utilizando técnicas de visión por computadora

**AUTOR:**

Guale Villao Heidy Julissa

**DOCENTE TUTOR:**

Ing. Manuel Montaño Blacio. MSc.

**LA LIBERTAD - ECUADOR**

**Año 2025**

## APROBACIÓN DEL DOCENTE TUTOR

En mi calidad de docente tutor del trabajo de Integración Curricular denominado: **"Desarrollo de un sistema de estacionamiento automatizado para vehículos utilizando técnicas de visión por computadora"**, elaborado por **Heidy Julissa Guale Villao** estudiantes de la Carrera de Telecomunicaciones, Facultad de Sistemas y Telecomunicaciones de la Universidad Estatal Península de Santa Elena, previo a la obtención del título de Ingeniería en Telecomunicaciones, me permito declara que, tras supervisar el desarrollo y estructura final del trabajo, este cumple y se ajusta a los estándares académicos. En consecuencia, lo considero apto en todos sus aspectos y listo para ser evaluado por el docente especialista.

Atentamente,

Ing. Manuel Montaño Blacio. MSc.

**DOCENTE TUTOR**

## DECLARACIÓN DE DOCENTE ESPECIALISTA

En mi calidad de docente especialista del trabajo de Integración Curricular **"Desarrollo de un sistema de estacionamiento automatizado para vehículos utilizando técnicas de visión por computadora"**, elaborado por **Heidy Julissa Guale Villao** estudiantes de la Carrera de Telecomunicaciones, Facultad de Sistemas y Telecomunicaciones de la Universidad Estatal Península de Santa Elena, previo a la obtención del título de Ingeniería en Telecomunicaciones, me permito declara que, tras supervisar el desarrollo y estructura final del trabajo, este cumple y se ajusta a los estándares académicos. En consecuencia, lo considero apto en todos sus aspectos y listo para la sustentación del trabajo. Atentamente Ing. Fernando Vinicio Chamba Macas, Mgt. DOCENTE ESPECIALISTA.

Atentamente,

Ing. Fernando Chamba Macas, Mgt

**DOCENTE ESPECIALISTA**

## DECLARACIÓN AUTORÍA DE LAS ESTUDIANTES

El presente trabajo de Integración Curricular con el título **"Desarrollo de un sistema de estacionamiento automatizado para vehículos utilizando técnicas de visión por computadora"**, declaro que la concepción análisis y resultados son originales a la actividad educativa en el área de Telecomunicaciones.

Atentamente

Guale Villao Heidy

Heidy Julissa Guale Villao

**C.I. 2400057994**

## DECLARACIÓN DE RESPONSABILIDAD

Quienes suscribe, Heidy Julissa Guale Villao con C.I. 2400057994, estudiante de la carrera de Telecomunicaciones, declaro que el trabajo de titulación denominada **"Desarrollo de un sistema de estacionamiento automatizado para vehículos utilizando técnicas de visión por computadora"**pertenece y es exclusiva responsabilidad de la autora y al patrimonio intelectual de la Universidad Estatal Península de Santa Elena.
Atentamente

Atentamente

Guale Villao Heidy

Heidy Julissa Guale Villao

**C.I. 2400057994**

## TRIBUNAL DE SUSTENTACIÓN

Ing. Ronald Rovira Jurado, PhD.

**DIRECTOR DE LA CARRERA**

Ing. Manuel Montaño Blacio, MSc.

**DOCENTE TUTOR**

Ing. Fernando Chamba Macas, Mgt.

**DOCENTE ESPECIALISTA**

Ing. Luis Amaya Fariño, Mgt.

**DOCENTE DE ÁREA**

Ing. Corina Gonzabay De La A, Mgt.

**SECRETARIA**

## AGRADECIMIENTO

A Dios por brindarme de sabiduría, fortaleciéndome en momentos difíciles y por no dejarme caer durante este proceso, su guía me ha permitido mantenerme firme hasta alcanzar esta meta. A mis padres, hermanos y sobrinos, por su amor, comprensión y palabras de aliento en cada etapa de esta travesía académica, su apoyo incondicional ha sido un pilar esencial en mi vida.

A mí misma por no rendirme a lo largo de este desafiante camino, por preservar y confiar en mis capacidades incluso en los momentos de dudas.

A mis amigos Sebastián, Andy, Maeba, Carmen y Daniela, gracias por compartir su sincera amistad, momentos de alegría, de desafíos y superar juntos cada reto que se presentó en el camino.

Al ingeniero Montaño, por su orientación, paciencia y compromiso durante el desarrollo de este proyecto, su acompañamiento fue fundamental para la culminación.

A mi pareja, por su compañía, motivación constante y por creer en mí incluso en los momentos de mayor incertidumbre.

*Heidy Julissa Guale Villao*

## DEDICATORIA

A mis queridos padres que con su sabiduría me han educado, comprendido y ayudado en esta etapa. Especialmente a mi papá que es mi gran amigo, sus consejos de vida, apoyo, esfuerzo y trabajo, han forzado mi carácter y humildad.

A mis queridos hermanos, Danny, Julio, Karen, Stephany y Ariana, gracias por la paciencia que me tienen y por esos ánimos durante todo este proceso. A mis queridos sobrinos Zhoemi y Elian que con su cariño y sus pequeños abrazos reconfortan mi alma. A mi pareja por su compañía, compresión en los días difíciles y estar presente a lo largo de este proceso.

A mis amigos del colegio, universidad, entorno laboral y a todas aquellas personas que, con su apoyo y confianza, han contribuido a mi crecimiento personal y profesional. Gracias por confiar en mí en cada una de las actividades que hemos compartido.

*Heidy Julissa Guale Villao*

Índice General

## Índice de Figura

## Índice de Tabla



## Índice de Anexo

## RESUMEN

El sistema de monitoreo de parqueo se implementó en los exteriores del laboratorio de redes de la Universidad Estatal Península de Santa Elena, mediante la instalación de una cámara que cubre toda el área de interés. Cada fotograma capturado se transmite y procesa en tiempo real utilizando Python y la biblioteca OpenCV. Para la detección de vehículos, se emplea el modelo YOLOv5s previamente entrenado, con una precisión del 99,8%.

La detección de ocupación se realiza aplicando máscaras binarias sobre regiones específicas, comparando la superposición entre las áreas delimitadas y los vehículos. Si existe coincidencia, el espacio se marca ocupado. Este enfoque obtuvo un 95,6% de exactitud durante las pruebas, empleando un umbral del 40% de intersección.

Los resultados se exponen a través de una API construida en Flask (Python) y se envían mediante el protocolo HTTP a Node-RED. Se crea un flujo de datos y se transmite la información por telemetría a la plataforma IoT ThingsBoard, donde se diseñan paneles interactivos para la visualización dinámica y pública del estado del parqueadero.



**Palabras claves:** Fotograma, YOLO, máscara binaria, API, Node-RED, HTTP, ThingsBoard, IoT.

## ABSTRACT

The parking monitoring system was implemented outside the network laboratory at the Santa Elena Peninsula State University by installing a camera that covers the entire area of interest. Each captured frame is streamed and processed in real time using Python and the OpenCV library. For vehicle detection, the pre-trained YOLOv5s model is used, with an accuracy of 99.8%.

Occupancy detection is performed by applying binary masks to specific regions, comparing the overlap between the delimited areas and the vehicles. If there is a match, the space is marked occupied. This approach achieved 95.6% accuracy during testing, using a 40% confidence threshold.

The results are presented through an API built in Flask (Python) and sent via HTTP to Node-RED. A data stream is created, and the information is transmitted via telemetry to the IoT platform ThingsBoard, where interactive dashboards are designed for dynamic and public visualization of parking status.



**Keywords:** Frame, YOLO, binary mask, API, Node-RED, HTTP, ThingsBoard,

# INTRODUCCIÓN

La congestión vehicular se ha consolidado como un problema alarmante a escala global, con un crecimiento exponencial que impacta directamente la vida en nuestras ciudades. Dentro de este complejo escenario, el estacionamiento emerge como un contribuyente significativo y persistente. La dificultad para encontrar un lugar donde dejar el automóvil no es un inconveniente; es un factor que agrava los atascos y aumenta los tiempos de desplazamiento [1].

Dentro del Ecuador, especialmente en las pequeñas ciudades en crecimiento persiste una problemática y es el tráfico vehicular, las cantidades de vehículos que circulan por sus calles a diario buscan plazas de estacionamiento, siendo escasas desde horas de la mañana[2] . Los usuarios se sienten afectados al no conseguir una plaza disponible, esto representa atrasos al ingresar a sus trabajos, estudios o alguna actividad.

Frente a esta problemática, el desarrollo e implementación de soluciones tecnológicas basadas en técnicas de visión por computador, con algoritmos de detección de objetos y segmentación de imágenes, han demostrado ser herramientas prometedoras para mejorar la gestión del estacionamiento. Sistemas inteligentes capaces de identificar en tiempo real la disponibilidad de plazas mediante cámaras y redes neuronales entrenadas, permiten informar a los usuarios sobre espacios libres de manera precisa y oportuna. Este sistema, reduce significativamente el tiempo dedicado a la búsqueda de parqueo y disminuye la circulación innecesaria de vehículos.  Además, al integrar el sistema con páginas web o aplicaciones móviles, facilita la toma de decisiones para los usuarios, fomentando una movilidad más eficiente, ordenada y sostenible.

# CAPÍTULO I

## 1.1. Antecedentes

Durante el estudio de los proyectos que han trabajado en estacionamientos automatizados, se han encontrado algunas con los mismos propósitos, este trabajo [3] se encuentra en el ámbito del internet de las cosas (IoT) para el estacionamiento inteligente de vehículos en ciudadelas universitarias. Propone un algoritmo de detección vehicular que utiliza un magnetómetro, donde se integra un sistema de comunicación basado en tecnología LoRaWAN, los sensores del sistema detectan la presencia de vehículos y establecen una intercomunicación, para posteriormente transmitir la información a una plataforma en la nube, haciéndola accesible a los usuarios finales.

En el siguiente proyecto [1], se utiliza visión artificial como mecanismo para el parqueo, este consiste en presentar un software de procesamiento de imágenes con Python y OpenCV. La detección de vehículos se realiza a través de una cámara de video, que envía los datos a una unidad de control y a un punto de acceso inalámbrico, estos componentes se interconectan para transmitir la información a una plataforma IoT y posteriormente los datos se presentan al usuario final.

Este estudio [4], se fundamenta en la red YOLO v3, aplicada específicamente para identificar espacios y vehículos en estacionamientos. Al implementarlo, se introduce una estructura residual que permite extraer características profundas de los espacios de estacionamiento, utilizando cuatro mapas de características a diferentes escalas para la detección de objetos. En comparación con el siguiente estudio [5] , propone una solución innovadora basada en visión artificial, consiste en combina el uso de YOLO v3 para la detección de objetos, una Raspberry Pi como plataforma de hardware y TensorFlow 2 para la clasificación de los objetos detectados.

En el proyecto [6], se presenta el diseño de un sistema inteligente de parqueo y la implementación de un prototipo capaz de procesar las señales de video a través de cámaras IP, su funcionalidad consiste en utilizar el procesamiento digital de señales por medio de MATLAB. En este caso las técnicas fueron óptimas y permitieron la detección e información de los espacios disponibles en parqueaderos de forma automática. Los resultados se obtuvieron tras varios días de pruebas, comprobaron que el vehículo es captado por la cámara y se tarda 45 segundos en procesar otro vehículo.

### 1.2. Descripción del Proyecto

El proyecto se centra en la creación de un sistema de estacionamiento automatizado para diferentes tipos de vehículos empleando técnicas de visión por computadora, con algoritmos de reconocimiento de objetos para identificar y monitorizar las plazas en tiempo real.

La infraestructura principal incluirá una cámara estratégicamente ubicada que capturará el área delimitada. El programa captura cada fotograma de video, que sirve como entrada para el procesamiento y posteriormente se somete a un algoritmo de detección que permite obtener datos fiables sobre la precisión del sistema. Los datos recopilados se almacenan en plataformas, garantizando su disponibilidad y respaldo continuo. Esto facilitará el acceso remoto a la información sobre la ocupación del parqueadero. Para mejorar la interacción con los usuarios, se implementará una interfaz intuitiva que presentará de manera clara y visual la disponibilidad de las plazas.

## 1.3.Objetivos del Proyecto

### 1.3.1. Objetivo General:

Desarrollar un sistema de estacionamiento automatizado mediante técnicas computacionales para la detección de plazas ocupadas o disponibles en un parqueadero.

### 1.3.2. Objetivos específicos:

- Realizar el estado del arte de las técnicas de visión por computador que se adapten al contexto del escenario.
- Entrenar el algoritmo de visión por computador para identificar plazas ocupadas o disponible
- Desarrollar una interfaz de usuario para mostrar información de la disponibilidad de las plazas.
- Realizar pruebas del sistema en un entorno controlado para garantizar su precisión y eficiencia.

## 1.4. Justificación

La gestión efectiva de espacios cerrados, como los estacionamientos, requiere un control adecuado para que los ciudadanos puedan optimizar sus actividades al buscar plazas disponibles, esto permite un uso más eficiente del tiempo y los recursos. A pesar de los esfuerzos realizados en años recientes para abordar estos desafíos, los resultados han sido insatisfactorios y la ciudadanía continúa enfrentando obstáculos significativos.

En Ecuador, la creciente demanda de automóviles en el país, en el periodo 2014-2023 el parque automotor experimentó un creciente promedio anual de 7,49%; es así como del año 2022 al 2023 se incrementaron 185,06 mil vehículos [7]. El incremento de vehículos provoca un desequilibrio entre la oferta y demanda de parqueaderos, debido a eso las ciudades cuentan con un número limitado de espacios para estacionar.

En la provincia de Santa Elena, tanto los espacios de parqueo públicos como privados continúan empleando métodos tradicionales para gestionar el estacionamiento de vehículos. Estas prácticas, que han sido adoptadas por la ciudadanía, resultan insuficientes durante épocas festivas, cuando se producen grandes aglomeraciones. Al ser una zona turística, la provincia recibe un ingreso vehicular de aproximadamente 28.993 automóviles durante los feriados nacionales, según datos de la Comisión de Tránsito del Ecuador (CTE) [8].

Esta brecha tecnológica, puede ser abordada mediante la implementación de modelos y algoritmos de detección de objetos. Este enfoque implica la captura de imágenes a través de cámaras, seguido de su procesamiento para generar información relevante. Esto proporciona un impulso significativo en áreas cerradas para agilizar la búsqueda de espacios libres.

En el ámbito económico, el uso de herramientas computacionales ofrece un ahorro significativo comparado con otras tecnologías, como el uso de sensores de estacionamiento, que representan un costo variable según su complejidad y precisión.

En el área social, el proyecto servirá como una herramienta para reducir la congestión, permitiendo a los conductores encontrar una plaza disponible, disminuir el tiempo de búsqueda y mejorar la fluidez del tráfico. Sin embargo, el proyecto contará con una serie de pruebas para medir el rendimiento antes de ser implementado en el entorno. Al mismo tiempo, el sistema reforzará la eficiencia de los espacios de estacionamiento, tanto públicos como privados, en la ciudad.

### 1.5. Alcance del Proyecto

El alcance del proyecto es implementar un algoritmo de visión por computadora para identificar plazas de estacionamiento utilizando una única cámara conectada a un ordenador. El sistema se centrará en detectar las líneas delimitadoras y verificar su estado, mediante el proceso de capturas de imágenes e intercepciones de cajas.

Se instalará una cámara en una posición estratégica y elevada para garantizar una visión clara, minimizando obstrucciones como plantas u otros elementos que podrían afectar la precisión del sistema. Esta configuración optimizará la capacidad de la cámara para capturar imágenes detalladas de las líneas del parqueadero, permitiendo una detección precisa de la ocupación de cada plaza.

El sistema desarrollará una interfaz de usuario que proporcionará información actualizada. Los usuarios podrán consultar esta información antes de ingresar al área de estacionamiento, agilizando así la búsqueda y reduciendo la congestión vehicular y en las áreas circundantes. mejorando significativamente la experiencia de los usuarios al ofrecer una solución eficiente y accesible para la gestión de espacios de estacionamiento.

### 1.6. Metodología

Para el proyecto de parqueadero inteligente utilizando las técnicas de visión por computadora, se seguirá una metodología fenomenológica, hermenéutica y práctico-proyectual. Se comienza con una fase de conceptualización para comprender la experiencia del usuario al buscar estacionamiento. Luego, se emplea el segundo enfoque para interpretar los datos capturados por las cámaras, asegurando una comprensión precisa de los espacios disponibles y ocupados. Finalmente, el tercer enfoque donde se ajusta el diseño de hardware y software para optimizar la detección y gestión de los espacios en tiempo real.

### 1.6.1. Metodología Fenomenológica

La fenomenología se centra en analizar los fenómenos o experiencias significativas que se presentan a la conciencia, se difiere del enfoque que estudia los objetos independientemente de la experiencia [9]. Este método sirve para entender cómo los conductores perciben, experimentan la búsqueda y ocupación de espacios. Se investigará la percepción visual de las señales de ocupación, la precisión en la detección de espacios libres y ocupados, así como el diseño e implementación del sistema desde la perspectiva del usuario.

### 1.6.2. Método Hermenéutica

El método hermeneútico se centra en la interpretación comprensiva de fenómenos, partiendo de la relación dinámica entre el todo y sus partes, donde el sentido emerge a través del diálogo entre el intérprete y el objeto de análisis [10]. Se utilizará para interpretar y comprender los datos capturados por las cámaras del estacionamiento, el análisis de imágenes y posterior procesamiento.

### 1.6.3. Método Práctico Proyectual

El método práctico proyectual consiste en una serie de operaciones necesarias, bajo un orden lógico con base en la experiencia, cuya finalidad es obtener un máximo resultado. Este trabajo se centrará en la aplicación práctica de la conceptualización inicial hasta la implementación del sistema. Esto incluirá el diseño detallado de hardware y software, la configuración de cámaras y la integración de un sistema de procesamiento de datos en tiempo real. Garantizará que la solución no solo sea teóricamente sólida, sino también funcional y viable en un entorno real de parqueadero.

### 1.7. Marco Contextual

La creciente demanda del parque automotor junto con la limitada disponibilidad de estacionamiento ha generado congestión, estrés y pérdida de tiempo para los usuarios. Normalmente se utilizan técnicas tradicionales para la búsqueda de espacios, con pequeñas limitantes frente a las exigencias de la sociedad moderna que usa la tecnología en sus actividades cotidianas. En este contexto, surge la necesidad de optimizar los espacios de estacionamiento y mejorar la experiencia del usuario con el uso de técnicas de visión por computadora, para detectar la presencia de vehículos mediante el análisis de video en tiempo real captado por una cámara.

Este tipo de tecnología emplea algoritmos de reconocimiento que identifican características propias de los vehículos y define regiones de interés (ROI) para enmarcar cada plaza de estacionamiento delimitando con precisión los estados disponibles y ocupados. Además, al procesar las imágenes el sistema facilita al usuario la información actualizada sobre su disponibilidad.

# 2. CAPÍTULO II

## 2.1.Marco Conceptual

El marco conceptual abarca diversos temas fundamentales para la implementación del sistema de parqueadero. Comenzará con la integración de tecnología en estos espacios, su conectividad y la arquitectura de la red.

Un componente central es la visión por computadora aplicada a este contexto, con el uso de redes neuronales convolucionales (CNN), algoritmos de detección de objetos (YOLO) y la identificación de regiones de interés (ROI). Además, se considera el hardware específico para la captura de imágenes, el tipo de cámara y las condiciones de iluminación para la precisión del sistema. Se abordan los protocolos de comunicación con la integración de software para el flujo de información y plataforma IoT para visualización de datos.

## 2.2.Marco teórico

### 2.2.1. Parqueaderos inteligentes

Los parqueaderos inteligentes aluden a la estrategia de combinar la tecnología con la innovación a fin de emplear la mínima cantidad de recursos posibles inmersos en la movilidad, como el consumo de combustible, tiempo y el espacio para que los vehículos puedan encontrar estacionamiento de la manera más rápida y sencilla posible. El desafío de estás soluciones apunta no sólo a una cuestión de comodidad sino, principalmente, al uso eficiente de recursos tecnológicos, como es la instalación de sensores, aplicaciones móviles y otras herramientas para optimizar la experiencia del usuario y eficiente[11].

Los parqueaderos cuentan con varias soluciones existentes para un correcto funcionamiento, a pesar de que algunas implementaciones fueron exitosas presentan limitaciones. La escalabilidad es un factor clave, dada la dificultad de implementar una

infraestructura compleja para la recopilación y el análisis de datos en zonas urbanas. Los costos de implementación son un factor decisivo a la hora de elegir la mejor solución para un proyecto en curso. Las preocupaciones sobre la privacidad en relación con la recopilación de datos e imágenes de los ciudadanos son desafíos adicionales [12]

### 2.2.2. Conectividad: Arquitecturas alámbricas e inalámbricas

Dentro de los sistemas de parqueaderos inteligentes, las conexiones inalámbricas y cableadas juegan roles complementarios para la comunicación y gestión de plazas. Las tecnologías de comunicación para redes de sensores inalámbricas más usadas en la actualidad son Wi-Fi, ZigBee y Bluetooth, mientras que las conexiones cableadas como Ethernet se utilizan para la comunicación de alta velocidad y transmisión de datos a grandes distancias [13].

- Redes de área personal inalámbrica (WPAN): Presentan una limitación de alcance, ya que, los dispositivos que pretenden comunicarse deben estar separados por un rango menor, como el Bluetooth permite la transmisión de voz y datos mediante un enlace de radiofrecuencia en la banda ISM de 2.4 GHz [14].
- Redes de área local inalámbrica (WLAN): Es una red de cobertura geográfica limitada, velocidad de transmisión relativamente alta, bajo nivel de errores y administrada de manera privada, su comunicación es mediante microondas [14]. Los estándares más recientes y velocidades que ofrece la tecnología Wi-Fi se presenta en la siguiente Tabla 1.

***Tabla 1:*** *Comparación de Estándares IEEE 802.11*

| Estándar | Velocidad Máxima Teórica | Bandas de Frecuencia |
|---|---|---|
| IEEE 802.11a | 54 Mbps | 2.4 GHz |

| | | |
|---|---|---|
| IEEE 802.11b | 11 Mbps | 5 GHz |
| IEEE 802.11g | 54 Mbps | 2.4 GHz |
| IEEE 802.11n | Hasta 600 Mbps | 2.4 GHz y 5 GHz |
| IEEE 802.11ac | Hasta 6.9 Gbps (teórico) | 5 GHz |
| IEEE 802.11ax | Hasta 9.6 Gbps (teórico) | 2.4 GHz y 5 GHz |
| IEEE 802.11be | Hasta 46 Gbps (teórico) | 2.4 GHz, 5 GHz y 6 GHz |

ZigBee: Es un estándar propuesto por la ZigBee Alliance que se utiliza en la transmisión de datos digitales, está basado en el estándar IEEE 802.15.4, los objetivos principales de este es tener redes de baja tasa de envío de datos y maximización de la vida útil de las baterías [13].

Ethernet: es una red de área local (LAN) comúnmente utilizada para la comunicación de datos a alta velocidad. En un sistema de estacionamiento inteligente, se utiliza para conectar dispositivos como cámaras, sensores de ocupación, paneles de información y la unidad de control central [15].

### 2.2.3. Visión por computadora para detección de vehículos

Los sistemas de visión por computadora han experimentado un gran desarrollo en el campo de la inteligencia artificial en los últimos años. Uno de los aspectos a considerar para este crecimiento ha sido no limitarse solo a nichos como la robótica y la manufactura, sino también a otras áreas como la domótica, la detección inteligente, el análisis de imágenes médicas, la industria alimentaria, la conducción autónoma, entre otros. Desde el inicio, el objetivo de los sistemas de visión por computadora ha sido el procesamiento, análisis e interpretación automáticos de imágenes [16] .

En el campo de la visión por computadora aplicada al flujo vehicular se han propuesto diferentes soluciones como sistemas de monitoreo que utilizan redes neuronales convolucionales (CNN), por ello tienen la capacidad de registrar el volumen de circulación y la información de distintos tipos de vehículos, enfocándose en la detección y cálculo del flujo [17].

### 2.2.4. Métodos Tradicionales de detección

Antes del auge de las redes neuronales, la detección de objetos se realizaba mediante métodos tradicionales basados en modelos matemáticos y reglas definidas. Estos enfoques seguían un pipeline compuesto por la generación de regiones candidatos, mediante el uso de algunas técnicas como: ventanas deslizantes (Sliding Window), extracción de características (como SIFT, LBP, HOG), clasificación supervisada (como máquinas de vectores de soporte SVM). Sin embargo, estos métodos presentan limitaciones importantes en cuanto a precisión, velocidad y capacidad de generación antes variaciones en la escena[18].

### 2.2.5. Métodos basados en Deep Learning

***Redes neuronales convolucionales (CNN)***

Las redes neuronales convolucionales o CNN (Convolution Neural Network) son un tipo de especializado de redes neuronales diseñadas específicamente para el procesamiento de datos relacionados con señales, particularmente aquellas transmitidas a través de imágenes. Este tipo de red es una variante de los perceptrones multicapa y se inspira en el córtex visual humano para llevar a cabo tareas de visión artificial. Dentro de la historia de Deep Learning, las CNNs han sido cruciales por imitar funciones del cerebro humano con éxito, como la lectura de cheques. A pesar de desafíos iniciales, se han convertido en las redes neuronales lideres que facilitaron el avance general de esta tecnología [19].

Las CNN es una construcción matemática que generalmente se compone de tres tipos de capas: convolución, agrupación (pooling) y capas completamente conectadas. Las capas convolucionales y de agrupamiento se encargan de la extracción de características, mientras que la tercera capa se utiliza para la clasificación, como se presenta en la Figura 2. A medida que los datos pasan por múltiples capas, las características extraídas se vuelven más abstractas y jerárquicas [20].

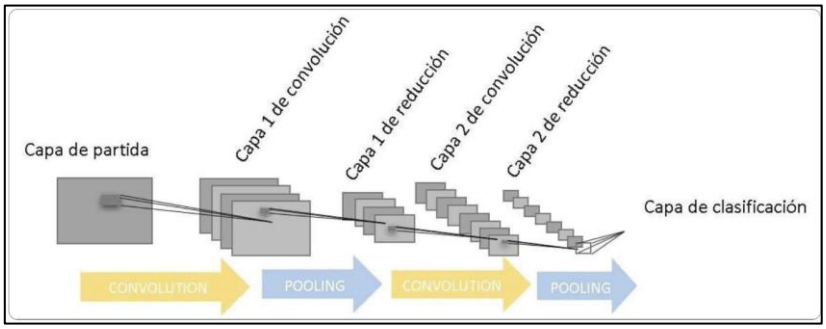


***Figura 1 - Arquitectura de una red neuronal convolucional.***

Bloques de la arquitectura de CNN

- Capa de convolución: la convolución es una operación lineal para extraer características de los datos de entrada (ver Figura 3). Donde se aplica un pequeño arreglo de números llamado filtro (kernel), sobre el tensor de entrada se calcula un producto elemento a elemento, y se va sumando para obtener el valor de salida en la posición correspondiente del tensor de salida, llamado mapa de características [20].

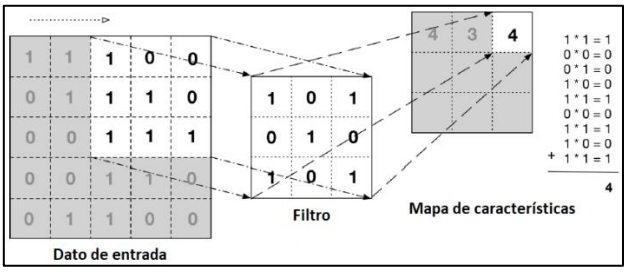


***Figura 2 - Operación de convolución***

- Capa de agrupación (pooling): proporciona una operación típica de submuestreo que reduce la dimensionalidad en el plano de los mapas de características, lo que ayuda a hacer el modelo más eficiente y resistente a pequeñas variaciones en la entrada. La técnica más común es la agrupación máxima como se presenta en la Figura 4, que el valor más alto dentro de pequeños parches (por ejemplo, 2x2) del mapa de características y descarta el resto, reduciendo las dimensiones espaciales (alto y ancho) a la mitad [20].

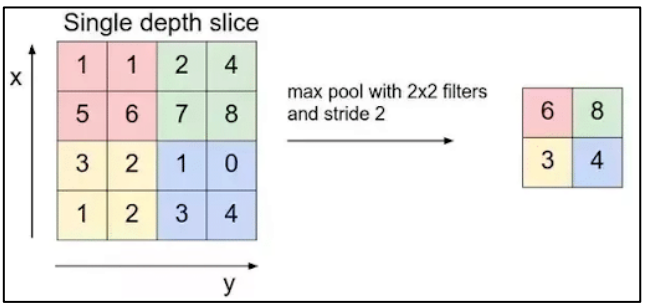


***Figura 3 - Capa de agrupación (pooling)***

- Las capas completamente conectadas: es la salida de la capa final de convolución o agrupación suelen aplanarse, es decir, transformarse en una matriz unidimensional (1D). Cada nodo de entrada está conectado a todos los nodos de salida mediante pesos aprensibles. La última capa completamente conectada suele tener un número de nodos igual al número de clases, y se sigue de una función de activación [20].

### 2.2.6. Algoritmos de detección de objetos con redes neuronales

Los algoritmos de detección de objetos son técnicas de visión por computadora que permiten identificar y localizar objetos específicos en imágenes o videos. Se divide la imagen en una cuadricula de celdas y predice cuadros delimitados y probabilidades en cada celda. Los algoritmos se clasifican ampliamente en dos categorías según la cantidad de veces que pasa una imagen en la red como se presenta en la Tabla 2.

***Tabla 2.*** *Comparativa de Algoritmo de Detectores de Objetos*

| **Característica** | **Una Sola Etapa** | **Dos Etapas** |
|---|---|---|
| **Velocidad** | Rápidos | Más lentos |
| **Proceso** | Detección y clasificación al mismo tiempo | Primero generan propuestas, luego clasifican |
| **Modelos representativos** | SSD, RetinaNet, YOLO | R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN |
| **Aplicaciones ideales** | Sistemas en tiempo real | Tareas que requieren alta precisión |

### *Detector multicaja de disparo único (SSD)*

SSD (Single Shot multi-box Detector), es un algoritmo que se especializa en la detección de objetos y son superiores a los algoritmos tradicionales en cuento a precisión y velocidad de detección. Esto se debe a su estructura basada en aprendizaje profundo de las redes neuronales. El funcionamiento básico consiste en la clasificación de objetos en una imagen y la localización de estos simultáneamente [21].

Uno de los aspectos más importantes de SSD es el desempeño a la hora de detectar objetos en diferentes escalas de tamaño en una sola imagen. Esto se logra mediante la utilización de múltiples tamaños de cajas, que se encuentran delimitadas por las diferentes capas de red neuronal. También, puede entrenar el modelo para detectar objetos en diferentes niveles de resolución[21].

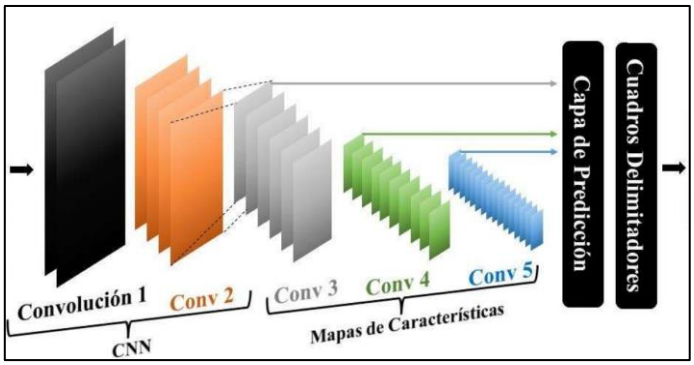


*Figura 4 - red neuronal convolucional (CNN) utilizada en SSD*

En la Figura 5, se describen los pasos de una Red Neuronal Convolucional (CNN) utilizada en el SSD para la detección de objetos. Los pasos que siguen son los siguientes:

Conv1) se aplica filtros convolucionales a la imagen de entrada. Conv2) se aplica filtros para extraer características. Conv3,4 y 5) profunda en la extracción de características, reduce el tamaño.

### *RetinaNet*

RetinaNet es uno de los mejores modelos de detección de objetos de una etapa, con una eficacia demostrada en el uso de objetos densos y de pequeña escala. Por ello, se ha convertido en un modelo popular para imágenes aéreas y satelitales. RetinaNet se creó mediante la incorporación de dos mejoras a los modelos existentes de detección de objetos de una sola etapa: las redes piramidales de características (FPN) y la pérdida focal [22]. Hay cuatro componentes principales de una arquitectura de modelo RetinaNet (ver figura 6).

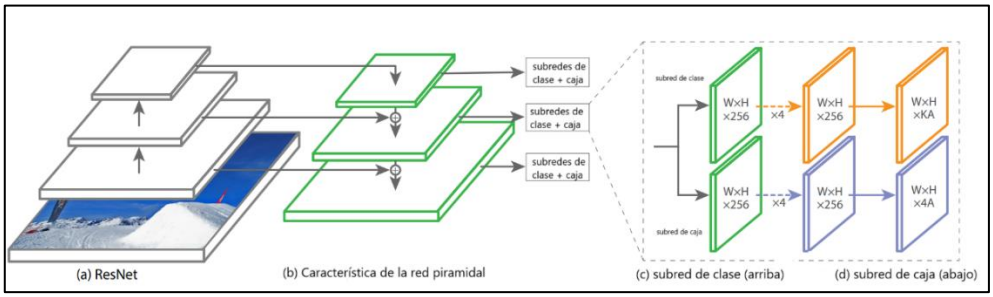


*Figura 5 - Arquitectura del modelo RetinaNet*

a) Vía ascendente: la red troncal (por ejemplo, ResNet) que calcula los mapas de características a diferentes escalas, independientemente del tamaño de la imagen de entrada o de la red troncal [23]
b) Vía descendente y conexiones laterales: la vía descendente sobre muestrea los mapas de características espacialmente más gruesos de los niveles piramidales superiores, y las conexiones laterales fusionan las capas descendentes y las capas ascendentes con el mismo tamaño espacial [23].
c) Subred de clasificación: predice la probabilidad de que un objeto esté presente en cada ubicación espacial para cada cuadro de anclaje y clase de objeto [23].
d) Subred de regresión: regresa el desplazamiento de los cuadros delimitadores desde los cuadros de anclaje para cada objeto de verdad fundamental [23].

***YOLO***

Es un enfoque de tiro único, se utiliza para la detección de objeto. Se divide la imagen en una malla de celdas y se asigna a cada celda la responsabilidad de detectar objetos en su área. Cada celda genera una caja delimitadora para los objetos que detecta y asocia con ellos una probabilidad que indica que el objeto se encuentra en la imagen [21].

Yolo, divide una imagen en una cuadrícula $S \times S$. Cada celda de la cuadricula se encarga de detectar un objeto si el centro del objeto esta dentro de esa celda. Cada celda predice varios cuadros delimitadores y puntuación de confianza, que indica la probabilidad de que el cuadro contenga un objeto y precisión de la detección. La supresión no máxima (NMS) es crucial como paso de post-procesamiento. Se utiliza para eliminar cuadros delimitadores, redundantes o incorrectos al identificar y conservar solo el cuadro delimitador con la puntuación de confianza más alta [24].

En el caso de YOLOv3 este modelo posee su propia red neuronal convolucional, llamada arquitectura de Darknet53 se basa en el concepto de conexiones residuales, similar a la

arquitectura de ResNet. Esto permite que la información fluya directamente a través de las capas en lugar de pasar por múltiples capas, permitiendo construir redes más profundas y efectivas. Darknet53 está compuesta por 53 capas convolucionales y utiliza bloques residuales para construir una arquitectura profunda y eficiente [21].

En 2020, se presentaron sucesivamente dos versiones de YOLO v4 y YOLO v5. El algoritmo YOLO v5 alcanzó una precisión de casi 50 mAP en el conjunto de datos COCO, garantizando al mismo tiempo la velocidad de operación. Considerando los requisitos de precisión y velocidad de los algoritmos de detección de vehículos en escenarios de monitoreo de carreteras. El modelo de red se mejoró para escenarios de carreteras con el fin de optimizar aún más la precisión del algoritmo de detección [25] .

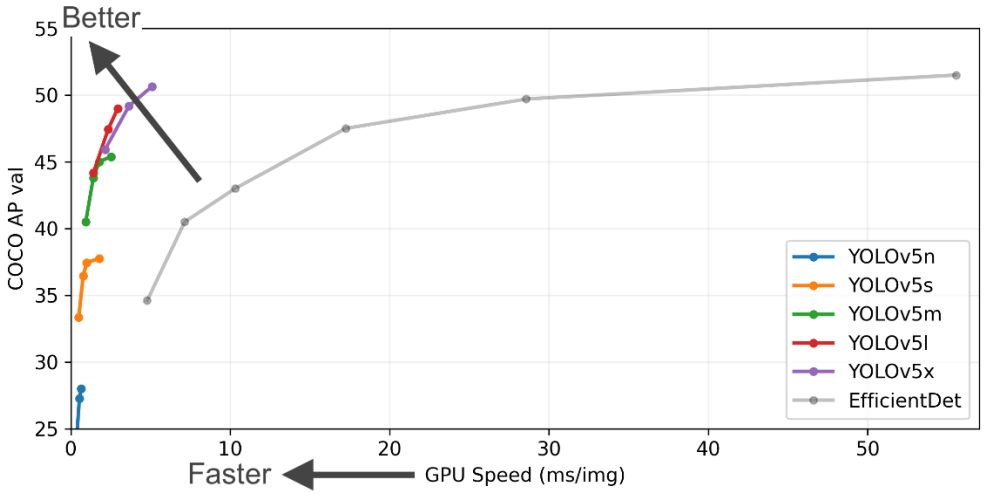


***Figura 6 - Yolov5-P5 640***

En la Figura 7, COCO AP val denota la precisión promedio (mAP) media en los umbrales de intersección sobre unión (IoU) de 0,5 a 0,95, medidos en el conjunto de datos COCO val2017 de 5000 imágenes en varios tamaños de inferencia (256 a 1536 píxeles) [26].

La velocidad de la GPU mide el tiempo de inferencia promedio por imagen en el conjunto de datos COCO val2017 utilizando una instancia AWS p3.2xlarge V100 con un tamaño

de lote de 32. Los datos de EfficientDet provienen del repositorio Google/automl con un tamaño de lote de 8 [26].

### 2.2.7. Métodos para definir una región de interés (ROI) y segmentación

Una región de interés (ROI), es una sección especifica de una imagen o un fotograma de video que se desea analizar, aumentando la eficiencia y velocidad de los algoritmos. Este concepto es esencial en el ámbito de la visión por computador para la detección de objetos, reconocimiento de patrones en las imágenes, análisis de video, entre otros. Además, las ROIs se pueden aplicar en diferentes campos o áreas como: medicina, robótica, sistemas de seguridad y vigilancia, sistemas de información geométrica y tecnología de los alimentos[27].

Las ROIs son estructura que suelen estar compuestas diferentes elementos, siendo los polígonos una de las formas más usadas debido a su simplicidad, se utilizan en diversos estudios y campos de investigación. Su uso es aplicado para resolver problemas de planificación de movimiento, desarrollando algoritmos que dividen un polígono en un conjunto de polígonos más pequeños con una determinada área, permitiendo su ruta y movimientos precisos [27].

***Algoritmo de proyección de rayos (Ray-Casting)***

El algoritmo de Ray-Casting se utiliza para determinar si un punto (centro de un pixel) está dentro de un polígono, el algoritmo traza un rayo desde el punto en una dirección fija (horizontal hacia la derecha) y cuenta las veces que el rayo intercepta los bordes del polígono[28].

- Si el número de intersecciones es impar, el punto está dentro del polígono (ver Figura 8).

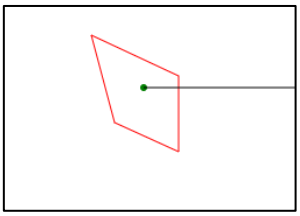

*Figura 7 - Intersección impar (Ray-Casting)*

- Si el número de intersecciones es par, el punto esta fuera del polígono (ver Figura 9).

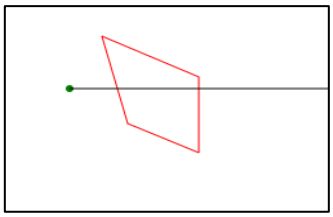

*Figura 8 - intersección par (Ray-Casting)*

### *Algoritmo del número de Bobinado*

El número de bobinado generalizado es un componente esencial del conjunto de herramientas de procesamiento geométrico, ya que permite cuantificar la cantidad de un punto dado dentro de una superficie, a menudo representada por una malla o una nube de puntos, incluso cuando la superficie es abierta, ruidosa o no es múltiple [29].

Existe el método alternativo para calcular un número de bobinado generalizado, basado únicamente en el límite de la superficie y las intersecciones de un único rayo con ella. En lugar de simplemente contar las intersecciones, calcula cuantas veces el polígono "envuelve" el punto. Si deslizas el punto en el borde a lo largo de todo el perímetro del polígono y se cuenta las veces que la banda elástica gira alrededor del punto central, ese es el número de bobinado [29].

- Si el número de bobinado es distinto de cero, el punto está dentro del polígono.
- Si el número de bobinado es cero, el punto está fuera del polígono.

***Uso de máscaras binarias***

La generación de máscaras es un resultado específico de la segmentación de imágenes, donde el resultado es una máscara binaria para cada objeto o región identificados. Una máscara es esencialmente una imagen en blanco y negro donde los píxeles blancos corresponden al objeto de interés y los píxeles negros representan el fondo u otros objetos. Estas máscaras proporcionan un contorno preciso de cada objeto segmentado, lo que permite un análisis detallado de la forma, el tamaño y la posición dentro de la imagen[30].

Una vez que se han identificado los píxeles dentro del polígono, la ROI representada como una máscara binaria. Como se presenta en la Figura 10, la nueva imagen donde los píxeles dentro de la ROI tienen un valor (1) y los píxeles fuera tienen otro valor (0) [30].

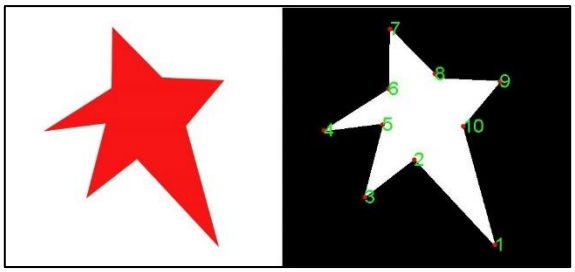


***Figura 9 - Uso de máscaras binarias***

### 2.2.8. Hardware para visión artificial

La visión artificial ha cobrado gran importancia en industrias dinámicas como la manufactura y la producción durante la última década. Con la rápida innovación y los avances en diversas áreas, como las técnicas de imagen, los sensores CMOS, la visión integrada, los estándares de transmisión de datos y las capacidades de procesamiento de imágenes. Las nuevas técnicas de imagen han abierto nuevas oportunidades de aplicación [31].

***La cámara***

El primer paso es la adquisición de imágenes. Esta adquisición consiste en recuperar una imagen de una fuente, generalmente sistemas de hardware como cámaras, sensores, etc. En un sistema de visión artificial, las cámaras se encargan de tomar la información lumínica de una escena y convertirla en información digital (píxeles) mediante sensores CMOS o CCD. Estos aspectos clave incluyen la resolución y el número total de filas y columnas de píxeles que admite el sensor. Cuanto mayor sea la resolución, más datos recopilará el sistema y con mayor precisión podrá evaluar las discrepancias del entorno. Sin embargo, un mayor número de datos requiere un mayor procesamiento, lo que puede limitar significativamente el rendimiento del sistema de visión artificial [31].

***Lentes***

La luz de la fuente debe enfocarse adecuadamente mediante un lente en el sensor para capturar la imagen con la máxima claridad. El lente debe proporcionar la distancia de trabajo, la resolución de imagen y el aumento adecuados para su sistema [31].

***Iluminación***

La iluminación es posiblemente el factor más importante en cualquier sistema de visión artificial y determina su éxito o fracaso. La configuración de iluminación debe proporcionar una iluminación uniforme en todas las superficies visibles de los objetos. Al configurar un sistema, también deben considerarse diversos factores, como la luz ambiental y las horas del día. Entre técnicas como la retroiluminación, la iluminación de campo claro, la iluminación rasante, la matriz lineal de ángulo bajo y la iluminación de campo oscuro, se pueden utilizar diversas técnicas para iluminar una escena [31].

### 2.2.9. Plataformas IoT y Gestión de Datos

En el ámbito del Internet de las cosas (IoT), existe plataformas que permite la recopilación, procesamiento, visualización y gestión de datos provenientes de dispositivos conectados.

***ThingsBoard***

ThingsBoard es una plataforma IoT de código abierto para la recopilación, procesamiento, visualización y gestión de dispositivos de datos. Permite la conectividad de dispositivos mediante los protocolos IoT estándar de la industria (MQTT, CoAP y HTTP) y es compatible con implementaciones tanto en la nube como locales (ver Figura 10). ThingsBoard combina escalabilidad, tolerancia a fallos y rendimiento para que nunca pierda sus datos [32].

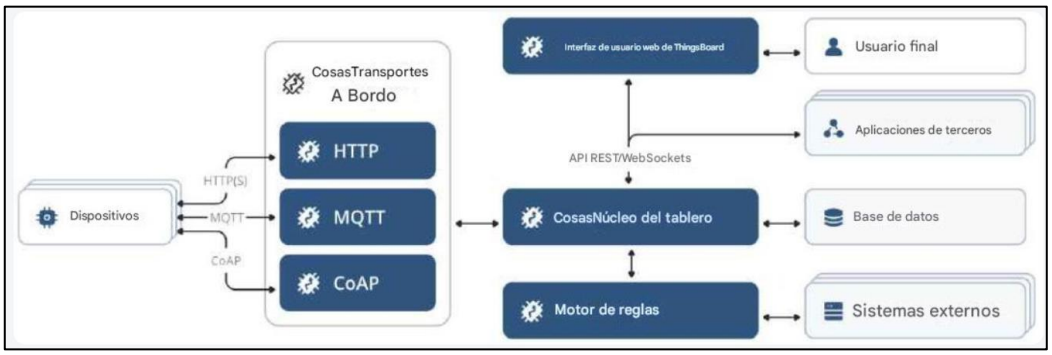


***Figura 10 - Plataforma ThingsBoard***

***Node-RED***

Node-RED es un proyecto desarrollado principalmente por IBM en 2013 y liberado en 2016, es multiplataforma, esto significa que corre en Windows, Macintosh, núcleos de Linux y Solaris. Al ser una herramienta de programación por flujo visual, permite ser utilizado por personal no especializado, inclusive por entusiastas y creativos. Desarrollado para consumir pocos recursos de cómputo [33].

El sistema informático Node-RED se incrusta perfectamente al Industrial Internet of Things (IIoT), al ser multiplataforma, multiparadigma, consumir pocos recursos, ofrecer

una baja curva de aprendizaje, pero, sobre todo, ofrece un alto poder de procesamiento y gestión de comunicación [34].

### 2.2.10. Protocolo de transmisión en tiempo real (RTSP)

El protocolo de transmisión en tiempo real (RTSP), es un protocolo de control de sesión que permite la transmisión de datos multimedia en tiempo real a través de redes IP. RTSP funciona como canal de control entre los servidores multimedia y los clientes que reproducen el contenido (ver Figura 11). Funciona en conjunto con otros protocolos, como RTP (Protocolo de transporte en tiempo real) y RTCP (Protocolo de control de transporte en tiempo real), que se encargan de la transmisión real de datos multimedia [35].

RTSP funciona empleando un modelo de solicitud-respuesta para la comunicación entre clientes y servidores. Este protocolo permite a los clientes enviar comandos al servidor, ordenándole que realice varias acciones, como iniciar una sesión, pausar la reproducción o buscar una marca de tiempo específica. Se utiliza ampliamente en una variedad de escenarios, incluidas las cámaras de vigilancia CCTV, donde facilita la transmisión en vivo y el monitoreo remoto. El puerto número 554 es el predeterminado y conocido que se designa para el tráfico RTSP. Cuando un cliente RTSP desea comunicarse con un servidor, establece una conexión con el puerto 554 del servidor [35].

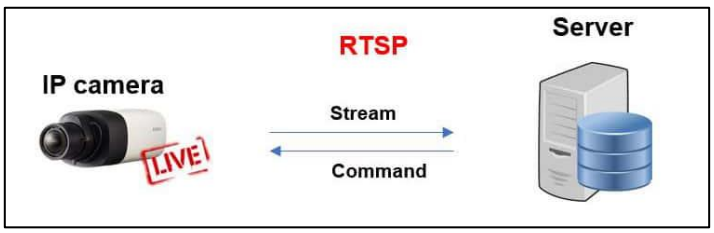


*Figura 11 - Protocolo RTSP*

RTSP es un protocolo de capa de aplicación, pero puede operar tanto sobre TCP como UDP para la capa de transporte (ver Figura 12). TCP está diseñado para ofrecer un flujo

de bytes de extremo a extremo altamente confiable en una red inestable. Se preocupa por la calidad del flujo de datos y prioriza la entrega ordenada y precisa de paquetes. En cambio, UDP se suele preferir para transmisiones de datos donde la inmediatez es esencial, la desventaja es que algunos paquetes de datos enviados podrían faltar o estar desordenados, provocando la perdida de algunos fotogramas o una ligera falla en la transmisión en vivo [36].

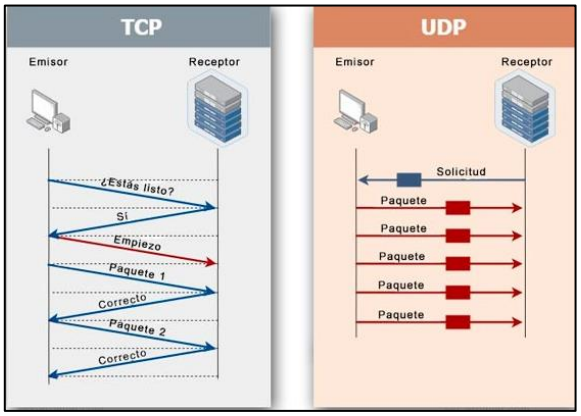


***Figura 12 - Protocolos de comunicación de red***

### 2.2.11. Protocolo de transferencia de Hipertexto (HTTP/HTTPS)

El protocolo de transferencia de hipertexto seguro HTPPS, es una versión mejorada y segura del protocolo HTTP. Se encarga de proteger la información que se transfiere entre el navegador de un usuario y el servidor de un sitio web mediante el cifrado de datos (ver Figura 13). Utiliza tecnologías como SSL (Secure Sockets Layer) y TLS (Transport Layer Security) para establecer conexiones seguras[37]. Proporciona tres capas esenciales de seguridad:

- Cifrado: Protege la información entre el navegador y servidor, asegurando que los destinatarios puedan acceder al contenido. Evitando que terceros intercepten.

- Integridad de datos:  Garantiza que la información transmitida no sea modificada o dañada durante el proceso, protegiendo de ataques maliciosos.
- Autenticación: permite que los usuarios se conecten al sitio web legítimo y no a una versión fraudulenta. Protegiendo del ataque tipo “man in the middle”.

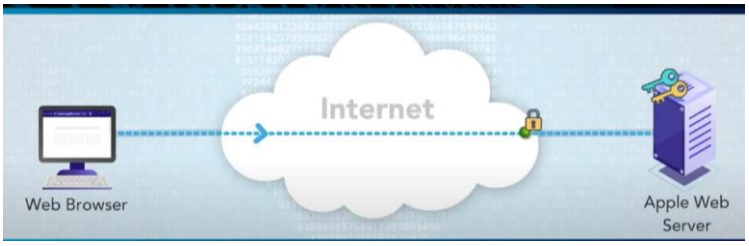


***Figura 13 - Protocolo de transferencia de Hipertexto***

# CAPÍTULO III

## 3. Desarrollo de la propuesta

### 3.1. Componentes de la propuesta

#### 3.1.1. Componente físico

Para garantizar un funcionamiento eficiente y fiable en el parqueadero automatizado, se han seleccionado componentes que se adaptan a las exigencias del entorno. Los elementos centrales son: una cámara Hikvision y el equipo de procesamiento (computador).

La vigilancia del estacionamiento está determinada por una cámara **Hikvision bala IP de 2 megapíxeles** (ver Figura 14). Esta herramienta ayudará a verificar la gestión del espacio y entre sus características técnicas se encuentra:

- **Resolución (2 MP - 1920 x 1080):** permite captar el video de manera nítida para identificar los vehículos y proceder a analizar el contenido con algoritmo de detección, ofreciendo una máxima protección y eficiencia.
- **Ángulo de visión amplio (lente fija de 2.8 mm):** esto es eficiente al cubrir una gran área en el estacionamiento y verificar los múltiples espacios.
- **Visión nocturna efectiva (IR hasta 30m con EXIR):** con la tecnología EXIR proporciona una iluminación uniforme, incluso en condiciones de baja luz u oscuridad total, funcionando sin problema 24/7.
- **Conectividad y compatibilidad (protocolos):** Soporta diferentes protocolos estándar (TCP/IP, DHCP, HTTP, RTP, RTSP, entre otros) para una correcta transmisión y visualización de los datos en tiempo real.
- **Compresión de video eficiente (H.265+/H.265):** utiliza tecnología avanzada de compresión para reducir el tamaño de archivos sin perder la calidad. Evitar la saturación de la red y almacenar grandes cantidades de grabaciones.


***Figura 14 - Cámara Hikvision***

Para el procesamiento y gestión de datos, el computador es el encargado de recibir la información de las cámaras, procesar, utilizar algoritmo de detección (Yolov5s) y gestionar el estado de las plazas. Para un correcto procesamiento se presenta las siguientes especificaciones:

- **Procesador (Intel Core i7-14700):** es fundamental para ejecutar los algoritmos detección y analizar el video en tiempo real, garantizando el procesamiento sin retrasos.
- **Memoria RAM (64GB):** permite al sistema manejar gran cantidad de datos de video y ejecutar múltiples procesos.
- **Disco de estado sólido (SSD Kingston SNV3S500G):** proporciona acceso rápido a los datos y agiliza la carga del sistema operativo.

### 3.1.2. Componentes computacionales

Entre los componentes computacionales para crear el entorno adecuado se encuentra en la Tabla 3.

***Tabla 3:*** *componentes computacionales*

| **Componentes computacionales** | **Características** |
|---|---|
| **Python 3.13.2** | • Es el entorno de ejecución principal y orquestador del pipeline de procesamiento. |

| | |
|---|---|
| | • Permite integrar sin problemas la adquisición de imágenes. |
| **Open CV** | • Es la biblioteca fundamental para el procesamiento de imágenes y videos a nivel de píxel y fotograma (frame).<br>• Establece conexión de flujos de video desde diversas fuentes (cámara IP). |
| **Tensor Flow** | • Es la infraestructura de backend que permite gestionar y ejecutar de manera eficiente modelos de redes neuronales entrenadas (YOLOv5). |
| **Ultralytics YOLOv5** | • YOLOv5 es la arquitectura específica y el modelo pre entrenado que realiza la detección de objetos.<br>• Procesa la imagen completa en una sola pasada a través de la red, proporcionando una velocidad de detección más alta. |
| **Makesense AI** | • Es una herramienta de anotación de imágenes. Utilizada para dibujar manualmente cajas delimitadoras precisas alrededor del vehículo y entrenar la precisión con la que YOLOv5 detecte. |
| **Node-RED** | • Es una plataforma de integración de sistemas basada en flujos y eventos, siendo la parte lógica donde conecta la detección y la gestión de datos.<br>• Recibe mensajes/eventos del script Python a través de protocolos de comunicación. |
| **ThingsBoard** | • Utilizada para la gestión de dispositivos IoT y la telemetría, su rol es vital para centralizar la información de todo el estacionamiento. |

## 3.2. Diseño de la propuesta

### 3.2.1. Arquitectura del sistema del parqueadero

La arquitectura del sistema de parqueadero automatizado está diseñada para optimizar la detección en tiempo real de los espacios disponibles y ocupados. Utilizando un enfoque de bajo costo y accesible de implementar. La arquitectura se presenta en la Figura 15, consta de cuatro etapas descritas a continuación:

- **Bloque de adquisición de imágenes:** es el punto de entrada del sistema, se emplea una cámara IP para capturar de manera continua y fiable el área de interés. Está se encuentra conectada físicamente por medio de cable UTP a un router como punto de acceso. Utilizando protocolos de red estándar (TCP/IP) para enviar el flujo de video en tiempo real a través del protocolo RTSP.
- **Algoritmo computacional:** Recibe el flujo de imágenes, se la aplica detección de objetos mediante YOLOv5 para identificar vehículos, generando cajas delimitadoras por cada frame. La definición del espacio se realiza dibujando manualmente una Región de Interés (ROI) poligonal.
- **Difusión de resultados:** los datos se presentan en una API Local para verificar el correcto funcionamiento de las plazas, los datos envían por a Node-RED por medio del protocolo HTTP, se crea un flujo de datos y se envía por telemetría a ThingsBoard.
- **Interfaz gráfica:** los datos llegan por telemetría y la información de las plazas se organiza y gestiona asociándola a dispositivo creado en la plataforma, representando cada uno de los espacios. El dashboard personalizado se hace accesible al usuario a través de una URL pública.

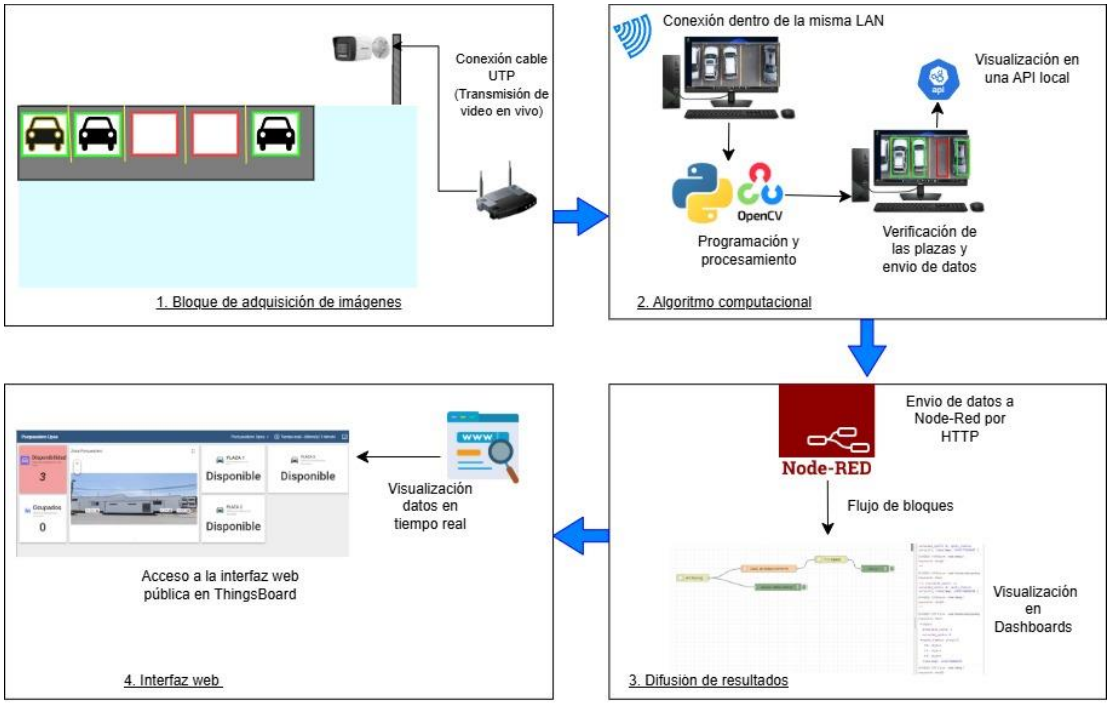


*Figura 15 - Arquitectura del parqueadero automatizado*

### 3.2.2. Instalación y ubicación de la cámara IP

La cámara IP se establece en los exteriores de la fachada del laboratorio de redes de la universidad, para garantizar un campo de visión rectangular perpendicular al plano de estacionamiento. En la Figura 16, se presenta el área de estudio acotada por las líneas de demarcación de tres plazas dentro del parqueadero. La ubicación geométrica optimiza la proyección de la cámara y asegura una adquisición continua del flujo de video para su posterior análisis y procesamiento.

*Figura 16 - Área de estudio*

La determinación de la cámara engloba el sector de estudio y se encuentra ubicada a una altura aproximadamente de 4m sobre el nivel del suelo, que proporciona un campo visual horizontal de 90°, ajustable para abarcar las tres plazas que se encuentran en la zona. La orientación de -10° respecto a la horizontal, se calibran alineando a las líneas del estacionamiento garantizando una correcta referencia geométrica para el proceso de detección (ver Figura 17).

***Figura 17 - Ubicación de la cámara***

El suministro simultáneo de energía y datos se realiza a través de una alimentación por Ethernet (PoE), conforme al estándar IEEE 802.3af, utilizando cableado estructurado UTP categoría 5e (ver Figura 18). La conexión tiene una longitud de 15 metros desde el punto de acceso (router) hasta el dispositivo alimentado (cámara). Esta configuración reduce significativamente la complejidad de instalación, minimiza los puntos de fallo y mejora la inmunidad a interferencias electromagnéticas, debido a los tendidos eléctricos.

***Figura 18 - Conexión de la cámara con el router***

### *3.2.2.1. Configuración de video en la cámara*

Para una transmisión de video eficiente desde la cámara, es necesario configurar ciertos parámetros como la resolución, tasa de bits máximo (bit rate), intervalo de fotogramas, entre otros. Estos ajustes son fundamentales para asegurar una calidad de imagen adecuada sin comprometer la estabilidad de la red. Para evitar pérdidas de paquetes durante la transmisión, se utiliza el protocolo TCP, garantiza una entrega confiable de los datos. En la Tabla 4 se presenta la configuración recomendada.

***Tabla 4:*** *Configuración de la cámara para el modelo*

| **Parámetro** | **Valor ajustado** | **Razón Técnica** |
|---|---|---|
| Resolución | 1280x720 | La cámara cuenta con dos tipos de resoluciones (1920x1080) y (1280x720). Sin embargo, se utiliza la resolución de 1280x720 debido a que requiere menos datos para transmitir, por lo tanto, menos ancho de banda. |
| Tasa de bits máximo (bit rate) | Fijo (2048 Kbps) | La cámara transmite el flujo de datos a un ritmo constante, manejando la misma velocidad de transferencia de datos. |

| | | |
|---|---|---|
| Intervalo de fotogramas (frame) | 15 FPS | Ofrece un rendimiento adecuado sin una perdida significativa de datos. |
| Códec | H.265 | Es más eficiente, ofreciendo una mejor calidad de video con la misma tasa de bits |

Para comunicar una cámara con resolución de 1280x720 a 15 cuadros por segundo (FPS), utilizando códec H.265 y una transmisión a través del protocolo RTSP/TCP, el ancho de banda requerido se estima en el rango de 1.5 Mbps a 3 Mbps.

#### *3.2.2.2. Topología de la red de comunicaciones*

Para que exista una correcta interoperabilidad entre los distintos componentes del sistema se empleará el uso del protocolo de comunicación TCP/IP, es fundamental definir de forma clara el esquema de red, las direcciones IP y los puertos utilizados (ver Figura 19). En la Tabla 5, se presenta las direcciones IP con el puerto TCP respectivo para la comunicación entre los dispositivos.

***Tabla 5:*** *Direcciones IP de dispositivos*

| Dispositivo | Dirección IP | Máscara | Puerto |
|---|---|---|---|
| Router | 192.168.1.1 | 255.255.255.0 | _____ |
| Cámara | 192.168.1.64 | 255.255.255.0 | 554 |
| Computadora | 192.168.1.100 | 255.255.255.0 | 80 |
| Servidor local (API) | 192.168.1.100 | 255.255.255.0 | 5000 |

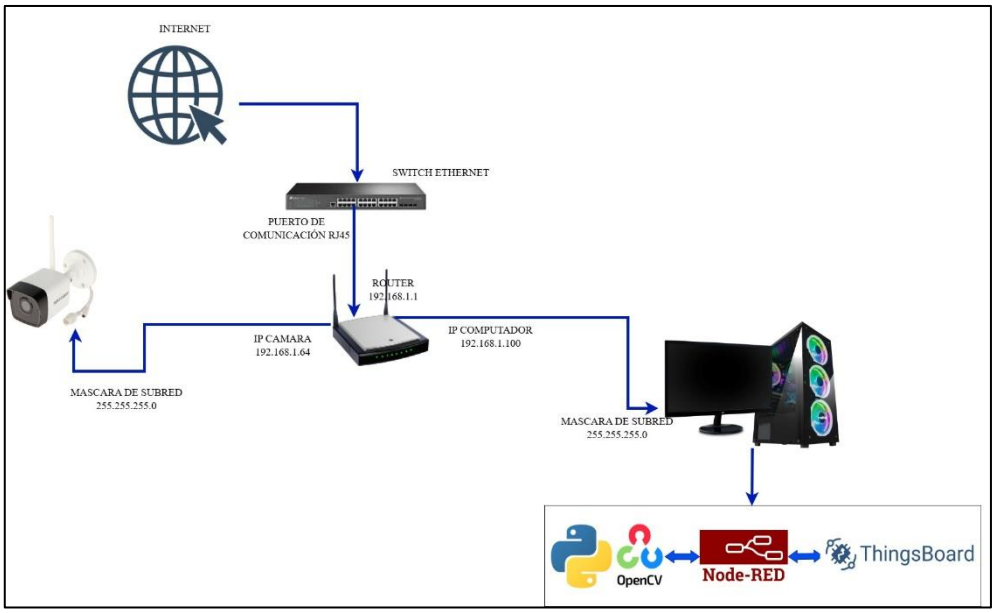


*Figura 19 - Topología de la red de comunicación*

### 3.2.3. Desarrollo del sistema de visión por computadora

Para el desarrollo del algoritmo de visión artificial, se empleará el software de Python, permitirá implementar las diferentes líneas de código necesarias para la lógica del algoritmo y el procesamiento de datos. Previamente a la fase de entrenamiento, será fundamental preparar el conjunto de datos (dataset). Para ello, la herramienta Makesense.AI se utilizará para etiquetar cada una de las imágenes del dataset. Posteriormente, el entrenamiento del modelo se realizará en la herramienta de Google Colab, que contiene un entorno de recursos computacionales como GPU y TPU. El modelo será entrenado para reconocer diferentes tipos de vehículos, considerando su exposición a variadas condiciones, tales como diferentes niveles de luminosidad, distintas condiciones ambientales y variaciones en la distancia.

#### *3.2.3.1. Construcción del dataset del modelo*

Para crear el dataset de datos, se crea una carpeta con el nombre de **images** que contiene todas las imágenes de vehículos en diferentes situaciones y una carpeta con el nombre de **labels** donde se guardaran las anotaciones.

La carpeta data, contiene el 100% de imágenes, donde se debe seleccionar el 80% para el entrenamiento y el 20% para validar. la carpeta de “train” contiene el conjunto de datos que se emplearán en la etapa del entrenamiento del modelo y aprendiendo a identificar patrones, características y relaciones relevantes. Por otro lado, la carpeta “val” corresponde al conjunto de validación de las imágenes que no se utilizaron en el proceso de entrenamiento.

El software online y gratuito MAKESENSE IA se utiliza para el etiquetado de los vehículos, se da **clic** en **Get Startud** para dar inicio con la importación de las imágenes (ver Figura 20).

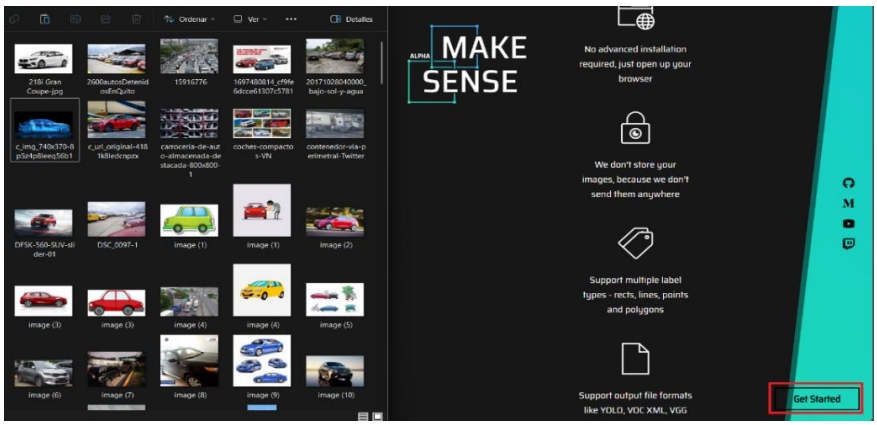


***Figura 20 - Importación de imágenes al software Makesense***

A continuación, se presenta una interfaz para colocar la carpeta con las imágenes seleccionadas para el etiquetado, aparecen dos opciones y se da clic en **“Object Detection”** (ver Figura 21)**.**

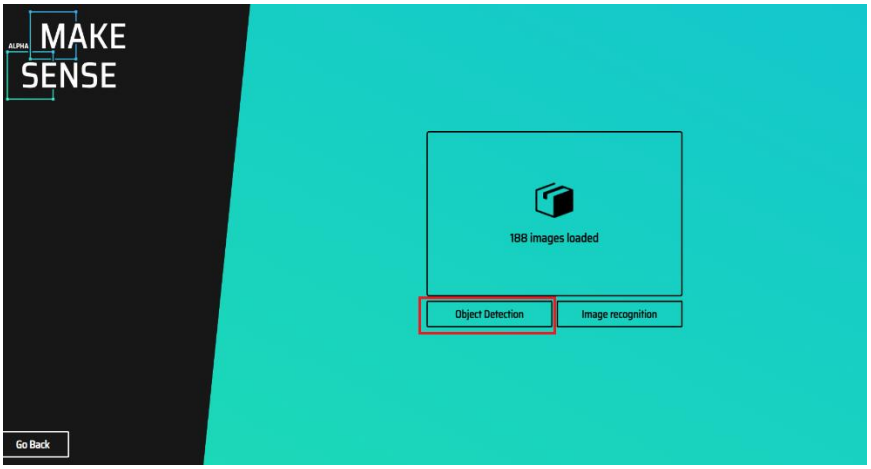


*Figura 21 - Importación de imágenes al software Makesense*

Luego de cargas las imágenes, aparece una pantalla para la creación de etiquetas (labels), se establece la etiqueta "carros" para identificar los vehículos, se da clic en **Start Project** (ver Figura 22)**.**

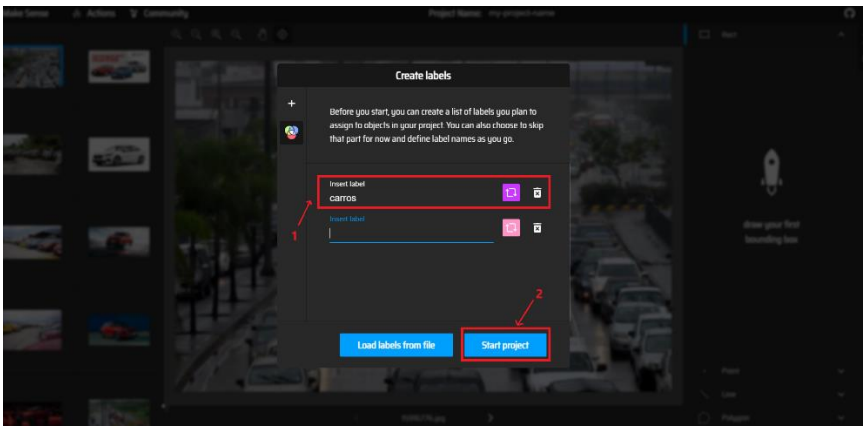


*Figura 22 - Creación de etiquetas de las imágenes*

Se presenta el área de trabajo donde se encuentran cada una de las imágenes de vehículos en diferentes situaciones, se encierra cada objeto y se procede a dar clic en **"Rect"** que proporciona la etiqueta, realizando el mismo proceso en cada una de las imágenes (ver Figura 23).


*Figura 23 - Etiqueta en carro*

Después de realizar cada anotación de las imágenes se da clic en **"Actions"** y elegir **"Export annotations"**, donde se selecciona "A.zip package containing files in YOLO format" y da clic en **Export** (ver Figura 24).

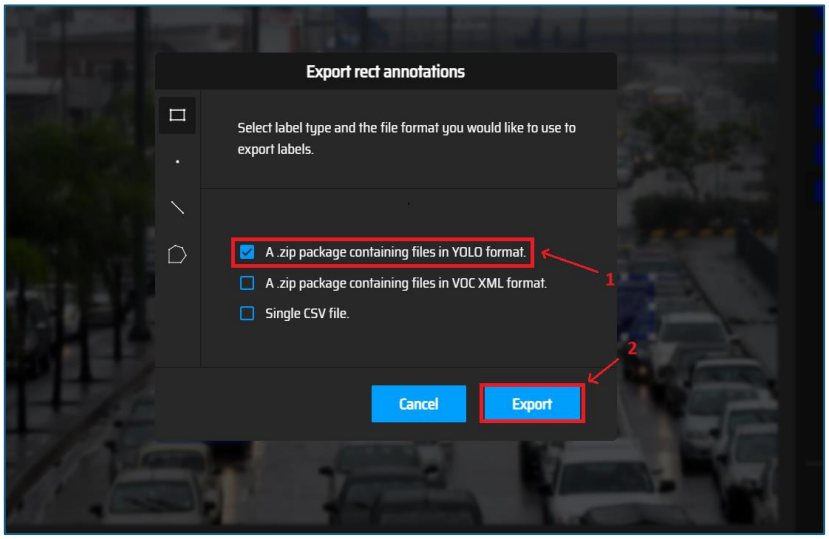


*Figura 24 - Exportación de las anotaciones de imágenes*

### *3.2.3.2. Entrenamiento del modelo de detección en Google Colab*

Se ingresa en el navegador a la página de Google Colab, se selecciona el entorno de ejecución para cambiar el tipo de acelerador por hardware dando a escoger opciones entre NONE -GPU o TPU, en el entorno se escoge GPU porque la ejecución del programa es más rápida (ver Figura 25).

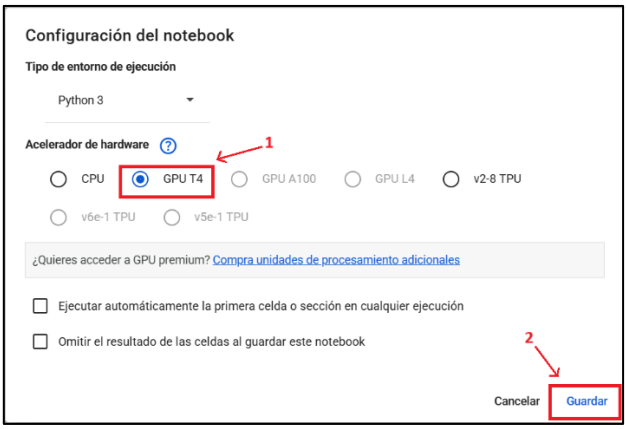


***Figura 25 - Entorno de configuración de notebook***

Se abre el archivo OBJECT_DETECTION_YOLOv5.ipynb en Google Colab, que proporciona una guía para realizar el entrenamiento del modelo para diferentes clases. En la primera línea de código se clona del repositorio de Ultralytics y se instalan las dependencias necesarias (ver Figura 26).

```
#clone YOLOv5
!git clone https://github.com/ultralytics/yolov5 # clone repo
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow
```

Se importa la librería torch y os necesarias para el entrenamiento. Además, se agregan librerías adicionales como Matplotlib, Opencv, Numpy, Ipykernel, Torch y Panda.


***Figura 26 - Entorno de Google Colab***

Es necesario contar con la carpeta que contienen las imágenes y anotaciones, para ello se carga la carpeta data.zip y se descomprime con una línea de código (ver Figura 27).

```
!unzip -q /content/data.zip -d /content/
```

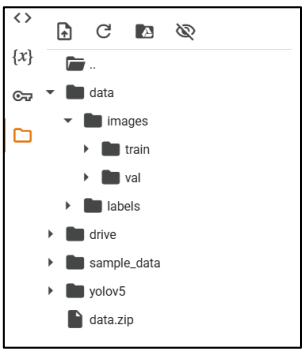


***Figura 27 - Carpeta data***

En la carpeta cargada de yolov5 se descarga el archivo coco128.yaml y realizar las modificaciones con el editor de preferencia, se realiza cambios en **train, val** y **names** con el nombre de la etiqueta de los carros, luego se carga el archivo en la carpeta da de Google Colab (ver Figura 28).

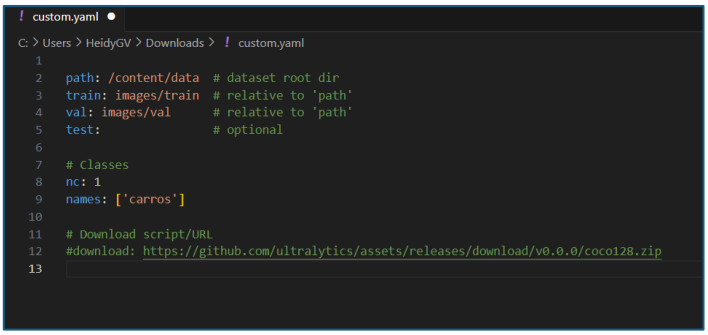


***Figura 28 - Modificación del archivo***

Para el entrenamiento se escogió la arquitectura de yolov5s (small), para la detección de carros en video en tiempo real utilizando una CPU, debido a que esta versión es conocida por la eficiencia computacional y menor consumo de recursos en comparación con grandes versiones de YOLO.

Dado que el procesamiento del flujo de video se realiza fotograma a fotograma, el modelo YOLOv5s debe analizar cada imagen individualmente. Al depender del procesamiento por CPU, la velocidad de análisis puede verse limitada, lo que podría generar un cuello de botella en el sistema, especialmente en escenarios de alta demanda o con múltiples flujos de video simultáneos.

Para la detección el modelo YOLOv5s, se entrena con un conjunto de datos compuesto por 400 imágenes. Un tamaño de batch size de 8 define cuantas imágenes se procesan por cada paso de actualización del modelo y el número de épocas 100 define cuantas veces el modelo ve todo el conjunto de datos durante el proceso.

Se utiliza la siguiente línea de código y la ejecución del modelo YOLOv5s (ver Figura 29).

```
! python train.py --img 400 --batch 8 --epochs 50 --data /content/yolov5/data/custom.yaml --weights yolov5s.pt –cache
```

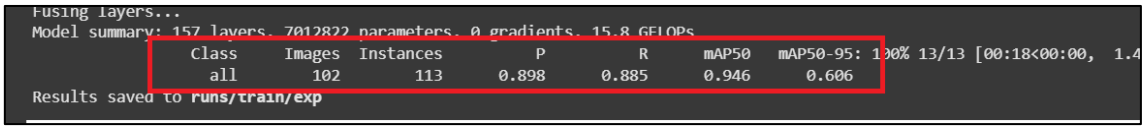


*Figura 29 - Modelo de YOLOv5s entrenado para vehículos*

Descarga del modelo de entrenamiento para el entorno Python (ver Figura 30).


*Figura 30 - Descarga del archivo con el entrenamiento*

## 3.3. Instalación de librerías y entorno para el detector de plazas del estacionamiento

Para un correcto funcionamiento de la detección de las plazas dentro del estacionamiento, se debe considerar la instalación librerías especificas en el entorno de **Python 3.13.2.** las librerías que se utilizan para la implementación del sistema se describen en la siguiente Tabla 6.

***Tabla 6:*** *Librerías para trabajar en Python*

| Librería | Detalles |
|---|---|
| opencv-python v.4.11.0.86 | Procesamiento y captura de imágenes/video |
| Numpy v.2.1.1. | Facilita el trabajo con arreglos multidimensionales (matrices) y ofrece funciones matemáticas. |
| Torch v.2.6.0 | Framework de aprendizaje automático, utilizado para crear redes neuronales profundas. |
| Pillow v.11.1.0 | Lectura y escritura de imágenes en diferentes formatos (JPEG, PNG, etc.) |

| | |
|---|---|
| Pandas v.2.2.3 | Se utiliza para la manipulación y el análisis de datos. |
| Request v.2.32.3 | Se utiliza para simplificar la realización de solicitudes HTTP, servidores web o APIs. |

### 3.3.1. Adquisición de la imagen y procesamiento

La adquisición de la imagen se realiza a través de una **cámara IP** utilizando el protocolo **RTSP** (Real Time Streaming Protocol) para la transmisión en tiempo real. Se emplea el protocolo **TCP** debido a su fiabilidad en la entrega ordenada y eficiente de paquetes, lo que es esencial para mantener la integridad del flujo de video.

Para acceder al stream, se define una variable llamada **CAMERA_URL** y se asigna un valor de cadena. Esta URL contiene información necesaria (usuario, contraseña, IP, puerto y la ruta específica del stream).

```
CAMERA_URL =
f"rtsp://{CAMERA_USER}:{CAMERA_PASS}@{CAMERA_IP}:554/Streaming/Channels/1?tcp"
```

Posteriormente se utiliza OpenCV (cv2) para capturar el video mediante la clase VideoCapture, posteriormente se transmite a una URL local usando Flask.

```
def initialize_camera():
    """Inicializar cámara"""
    global cap
    try:
   cap = cv2.VideoCapture(CAMERA_URL)
```

### 3.3.2. Configuración del modelo YOLOv5s

El modelo YOLO no recopila datos en video de transmisión, sino que operan sobre imágenes estáticas individuales. Cuando se procesa el video se extrae cada fotograma

(frame) del video y pasarla individualmente al modelo para el análisis. Se carga el modelo ya previamente entrenado se utiliza yolov5s, es una versión pequeña y rápida para procesamiento en tiempo real.

El umbral de confianza de 0.30 significa que el modelo solo reportará la detección con una puntuación igual o superior a ese valor, permitiendo mayor detección y reduciendo la probabilidad que el modelo pase por alto objetos reales (falsos negativos) (ver Figura 31).

```
# Cargar tu modelo personalizado
model = torch.hub.load("ultralytics/yolov5", "yolov5s" "custom", path="runs/train/exp/weights/best.pt")
model.conf = 0.30 #Umbral de confianza
```

Para extraer los resultados de detección de objetos de la salida de un modelo YOLO se denomina la variable "**detections**", donde los formatea en un DataFrame de Pandas que incluyen las coordenadas del cuadro delimitador.

```
# Obtener detecciones
detections = results.pandas().xyxy[0]
```

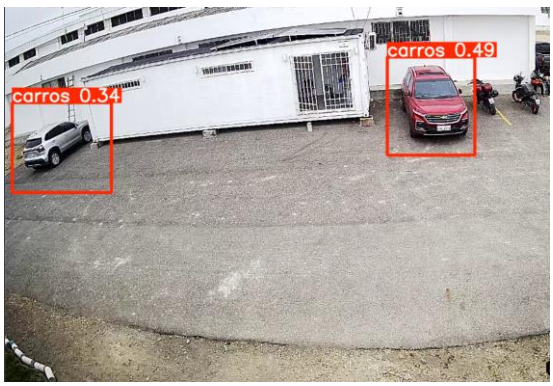


***Figura 31 - Funcionamiento de YOLOv5s para detectar vehículos***

### 3.3.3. Definición de plazas

Para definir las plazas se analiza el área de estudio, el estacionamiento cuenta con 3 plazas disponibles para vehículos, desde la visualización de la cámara se van a definir como polígonos las áreas delimitantes de cada uno de los espacios en el parqueadero. La definición correcta de las coordenadas del polígono permitirá la precisión del sistema y podrá identificar si un vehículo está dentro o fuera del espacio.

En la estructura están definidas en la lista **parking_sports**. Cada elemento de esta lista es un diccionario que representa una plaza individual:

- **"id":** es el número identificador para cada plaza (1,2,3).
- **"polygon":** los vértices del polígono se definen mediante diferentes listas de puntos [x, y]. Estos puntos delimitan con precisión las áreas de interés, lo que permite a la cámara capturar correctamente los datos correspondientes.

```
# Definir plazas
parking_spots = [
    {
        "id": 1,
        "polygon": [[88, 369], [24, 418], [54, 429], [125, 388]]
    },
    {
        "id": 2,
        "polygon": [[739, 335], [759, 398], [841, 394], [813, 319]]
    },
    {
        "id": 3,
        "polygon": [[844, 330], [895, 397], [968, 385], [906, 329]]
    }
```

### 3.3.4. Definición de espacios (Polígonos/ROIs)

A partir de la determinación de los puntos para conformar el polígono se define una función **check_spot_occupation**, para determinar si un espacio de estacionamiento está ocupado por un vehículo.

- Creación de una máscara para el polígono del espacio

Se determina el tamaño de una imagen temporal (máscara). Se asegura que las dimensiones sean al menos 1000 x 1000 pixeles, siendo suficientemente grande para contener tanto el polígono del espacio como el rectángulo del vehículo.

```
mask_height = max(y_max+1, 1000)
mask_width = max(x_max+1, 1000)
```

Se crea una imagen en negro (pixeles a 0) del tamaño calculado, esta será la máscara para el espacio de estacionamiento. En la línea del código **np** hace referencia a la librería Numpy y **dtype=np.uint8** significa que los pixeles serán enteros de 8 bits.

```
spot_mask = np.zeros((mask_height, mask_width), dtype=np.uint8)
```

Los puntos del polígono se convierten el formato para trabajar en OpenCV **cv2.fillPoly.**

```
poly_pts = np.array(polygon, np.int32).reshape((-1, 1, 2))
```

Se dibuja el polígono del espacio sobre sport_mask. Los pixeles que caen dentro del polígono se establecen con un valor de **1=blanco**, mientras que el resto en **0=negro** (ver Figura 32).

```
cv2.fillPoly(spot_mask, [poly_pts], 1)
```

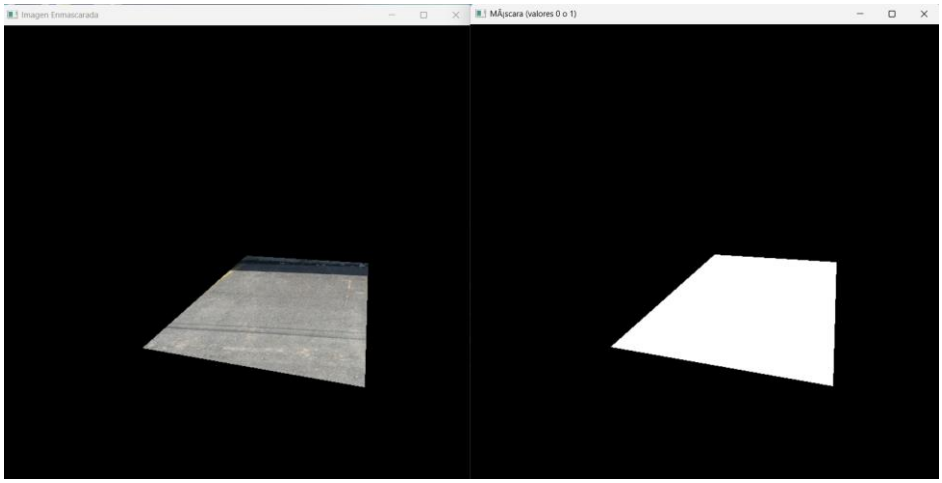

*Figura 32 - Mascara binaria en espacio poligonal*

- Creación de una máscara para el rectángulo delimitador

Se crea una imagen en negro del mismo tamaño que **spot_mask,** para la máscara del vehículo.

```
vehicle_mask = np.zeros_like(spot_mask)
```

Se dibuja el rectángulo del vehículo sobre la vehicle_mask, los pixeles se establecen a 1 y el -1, esto indica que el rectángulo se rellena.

```
cv2.rectangle(vehicle_mask, (x_min, y_min), (x_max, y_max), 1, -1)
```

- Cálculo del área de intersección

Se realiza una operación lógica AND entre las dos mascaras (spot_mask y vehicle_mask). El resultado de la intersección será una nueva mascara que proporciona el área donde el vehículo y el espacio del estacionamiento se superponen.

```
intersection = cv2.bitwise_and(spot_mask, vehicle_mask)
intersection_area = np.sum(intersection)
   # Calcular áreas
vehicle_area = (x_max - x_min) * (y_max - y_min)
spot_area = np.sum(spot_mask)
```

- Definición de umbrales

Los umbrales de ocupación se establecen de la siguiente manera: un espacio se considera ocupado si el área del vehículo supera el **40%** de superposición (ver Figura 33).

```
return (vehicle_overlap_ratio > 0.40)
```


***Figura 33 - Área de intercepción***

### 3.3.5. Envió a API para verificación del funcionamiento

Como se presenta en la Figura 34, se utiliza un API que permita verificar el funcionamiento del sistema y ver la disponibilidad de las plazas. Además de detectar las pérdidas de paquetes en la transmisión de tiempo real, ocasionados por inconvenientes como la desconexión de la cámara, el modelo, entre otros factores.

La monitorización del sistema se realiza a través del endpoint de la API HEALTH_CHECK_API_URL = http://tu-api.com/health, ver Anexo 3. Este endpoint es consultado con una frecuencia de 6 segundos para el envío de datos y verificación de los siguientes componentes críticos:

- La conexión de la cámara (cola de frames no vacía).

- Tiempo desde la última detección (evita stalls).
- Estado del modelo YOLOv5.
- Estado de las plazas.


***Figura 34 - API para verificación de plazas***

### 3.3.6. Configuración de Node-RED

Para la instalación y ejecución de Node-RED, se ingresa la página oficial, donde se descarga la última versión LTS de Node.js https://nodejs.org/en/ . Una vez instalado, se ingresa al CMD del sistema y se coloca el comando Node-red y se ingresa a través de la dirección IP local que proporciona el programa.

- Se ingresa al menú en el parte superior derecho
- Haga clic en “Administrar paletas”.
- Se da clic en la pestaña “Instalar”.
- Buscar las librerías necesarias para importar los nodos dentro del área de trabajo.
- Los nodos utilizados en el trabajo son: Node-red-contrib-thingsboard-rest-api y Node-red-dashboard (ver Figura 35).

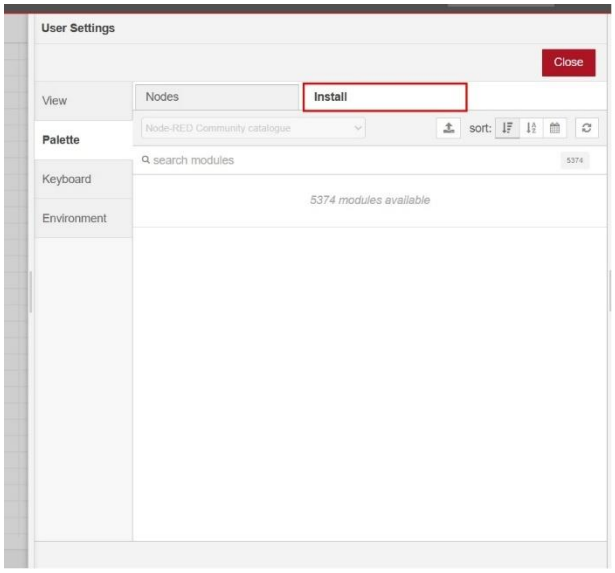


***Figura 35 - Administración de paletas e instalación de nodos***

Después de tener los nodos instalados, se procede a verificar en la tableta (lista de nodos del lado izquierdo) para comenzar a añadir los widgets.

### *3.3.6.1. Envió de datos del script Python a Node-RED*

Los datos son generados periódicamente cambiando de estado (disponible y ocupado), él envió a la plataforma, se realiza mediante peticiones POST al URL. Para establecer la comunicación HTTP en Node-RED, se debe configurar el nodo **HTTP in,** (ver Figura 36).

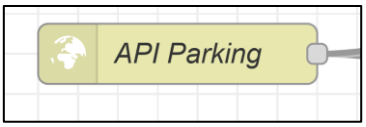


***Figura 36 - Node HTTP in***

Después de establecer la comunicación se configura el nodo, seleccionando el método **POST**, ya que este permite recibir datos en el cuerpo de la solicitud.

En el campo **URL**, se debe definir la ruta deseada (/api/parking). Está debe coincidir con la utilizada en el código Python, de modo que se mantenga una comunicación correcta entre el cliente y el servidor (Node-RED) (ver Figura 37).

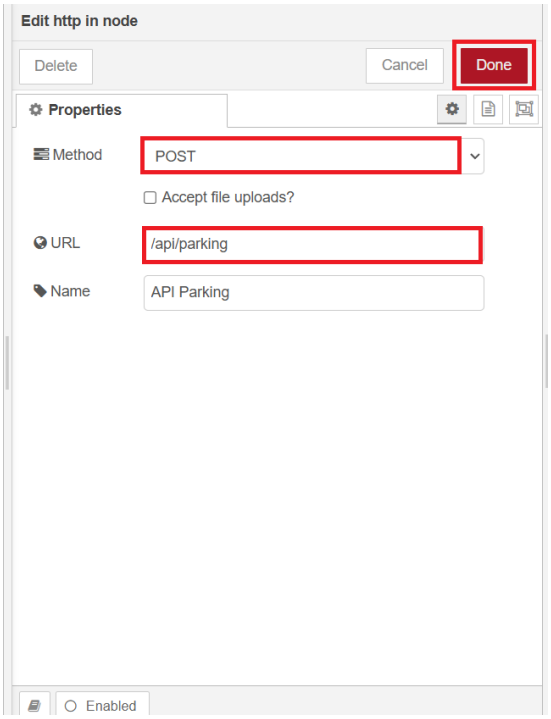


*Figura 37 - Configuración del nodo HTTP IN*

En el código de Python (ver Anexo 1), se configura de la siguiente manera:

- La función **send_to_node_red(),** está diseñada para enviar periódicamente datos del estacionamiento a Node-RED.
- Se declara funciones y variables globales: **parking_data** (datos del estacionamiento) y **last_sent_time** (cuando se enviaron los datos por última vez).
- Se prepara los datos que se van a enviar: el número de espacios disponibles, espacios ocupados, estado de cada espacio individual y la marca de tiempo.
- Se prepara los datos en formato JSON para los datos, el Headers HTTP apropiado y un timeout de 3 segundos para evitar el bloqueo.
- Como los cambios del estacionamiento son muy frecuentes, se reduce el tráfico de red enviando una solicitud por minuto sin perder información relevante. Para verificar la correcta llegada de datos se coloca un nodo **Debug** para verificar los datos.

### *3.3.6.2. Configuración de ThingsBoard*

Para usar la interfaz con el usuario se selecciona el programa ThingsBoard Community por ser gratuita en la nube. Se ingresa a través del portal cuyo dominio es https://demo.thingsboard.io/login. Al registrarse a la plataforma permite seleccionar el lenguaje y ambiente de gestión para configurar usuarios y recursos.

- Se presenta la pantalla principal y se da clic en el menú izquierdo donde se presentan varias opciones: Alarmas, paneles, entidades, perfiles, clientes, entre otras.
- Se da clic en la opción **Entidades** y se despliega las opciones, dando clic en **Dispositivos** (ver Figura 38).

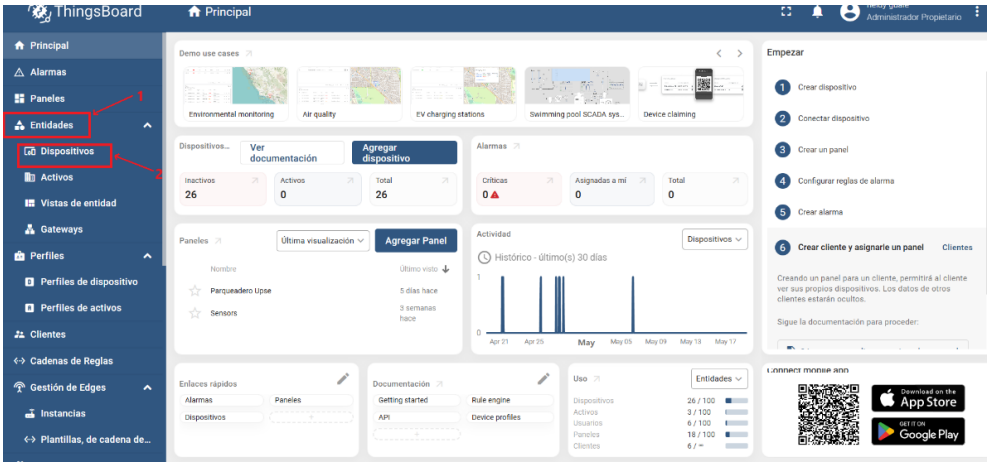


***Figura 38 - Pantalla general de ThingsBoard***

- Se da clic en **Dispositivos** y aparece el panel de trabajo.
- Para agregar un nuevo **Dispositivo**, se oprime en el icono del **signo más** como se presenta en la Figura 39.

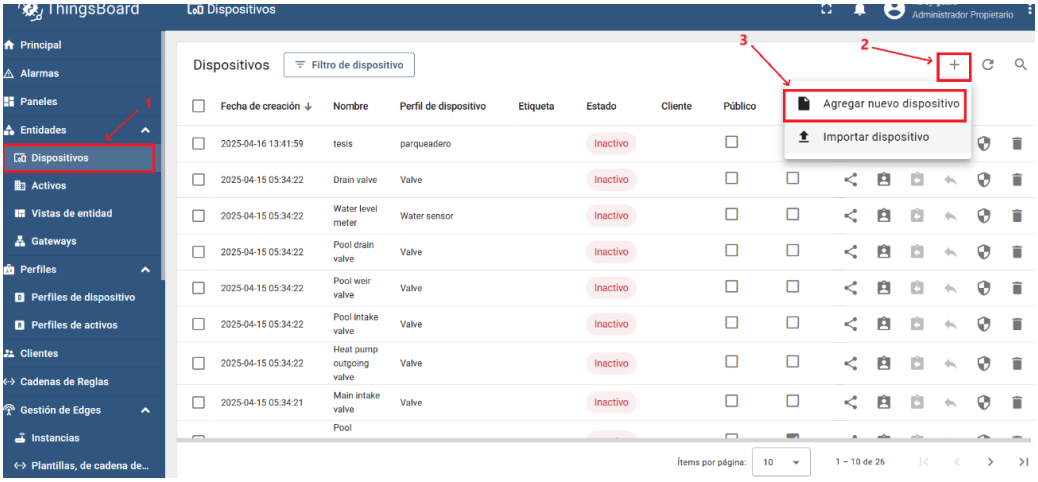


*Figura 39 - Agregar Dispositivos en ThingsBoard*

- En la figura 40, se ingresa los detalles del dispositivo tanto el nombre y perfil.

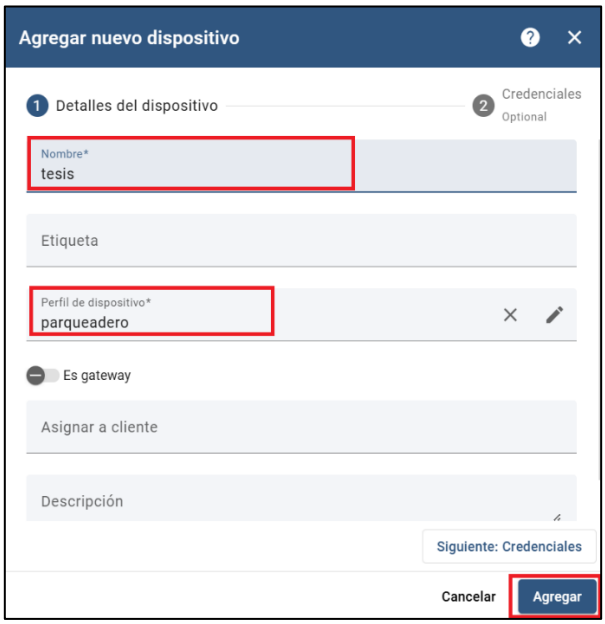


*Figura 40 - Asignar el nombre al nuevo dispositivo*

- En la Figura 41, se presenta la **interfaz 'Dispositivo creado. Prueba de conectividad'**. Se presenta que el protocolo HTTP elegido para recibir datos por telemetría.
- Para el comando de prueba, se ejecuta el siguiente comando en Windows: **curl -v -X POST** https://demo.thingsboard.io/api/v1/HA368vso2WyueB9sl**.** Al

ejecutar, se envían datos de prueba a la plataforma para verificar que el dispositivo se conecte correctamente.

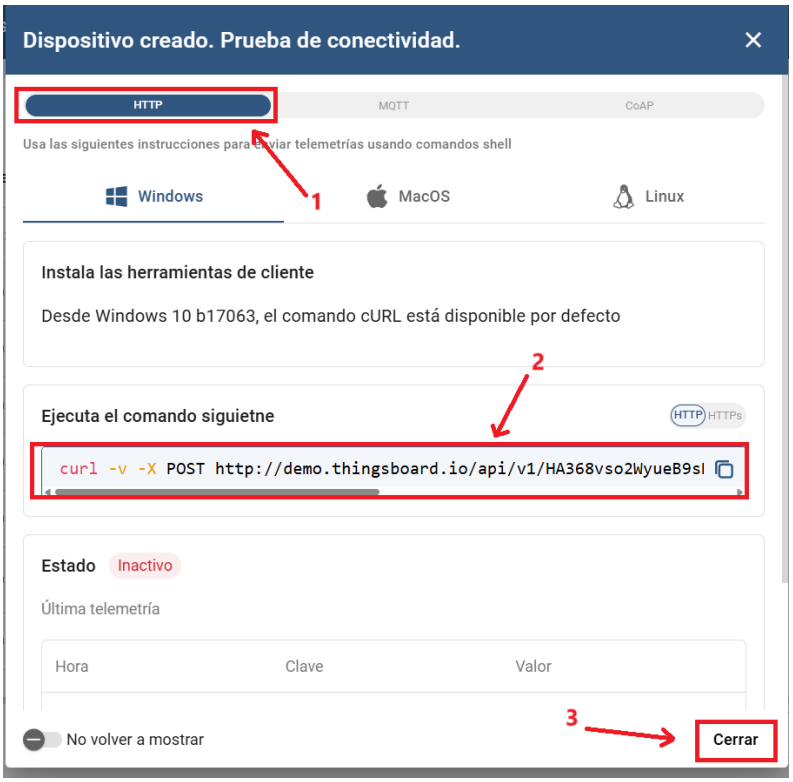


*Figura 41 - Interfaz de pruebas de conectividad*

### *3.3.6.3. Node-RED con ThingsBoards*

Se agrega el nodo **Function** en Node-RED para procesar y formatear los datos recibidos mediante solicitudes HTTP provenientes del cliente (ver Figura 42). Este nodo transforma el payload recibido para adaptarlo al formato de telemetría requerido por la API de ThingsBoard, ver Anexo 2.

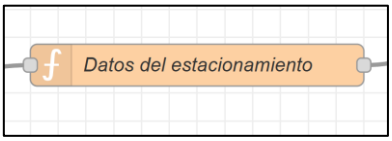


*Figura 42 - Nodo Function en Node-RED*

Se integra un nodo **HTTP request** donde se configura la **URL** que proporciona TB para conectarse y el correcto envío de datos por telemetría (ver Figura 43).

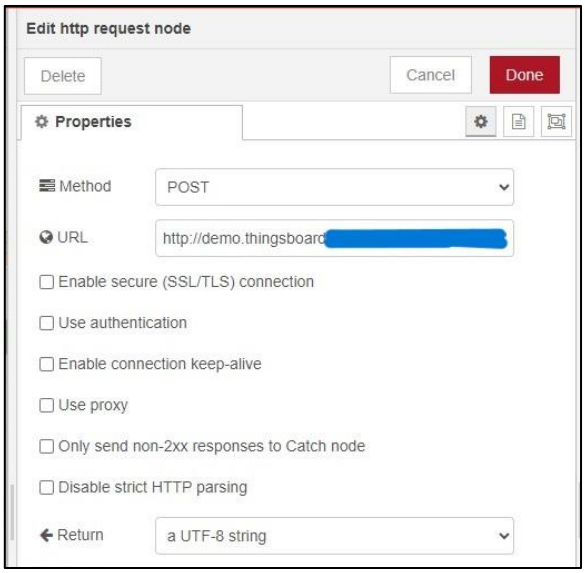


***Figura 43 - Nodo HTTP request en Node-RED***

En la Figura 44, se presenta la integración del flujo de nodos, en la parte derecha del flujo se visualizan los datos entrantes provenientes de Python y la confirmación del envió de la telemetría hacia ThingsBoard.

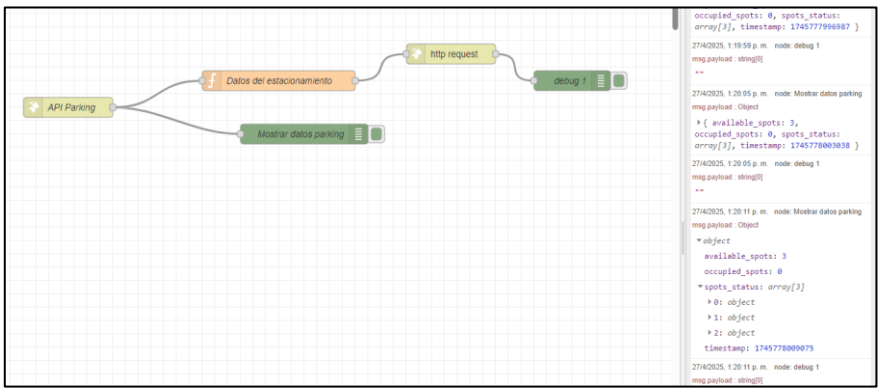


***Figura 44 - Resultados del flujo de nodos***

Para la verificación de los datos, se ingresa a los detalles del dispositivo en la parte de **Última telemetría**, donde se visualiza la ultima hora de llegada y los valores como la disponibilidad de las plazas, y la ocupación de estas (ver Figura 45).

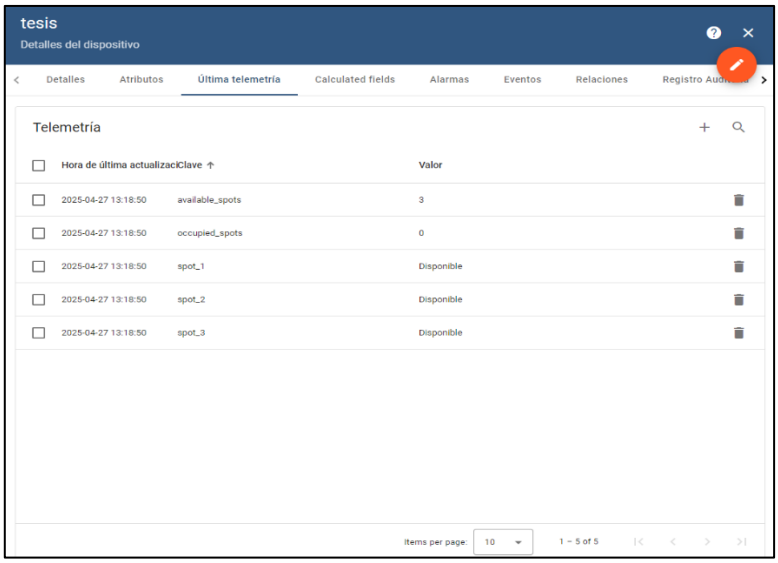


*Figura 45 - Llegada de datos por telemetría a ThingsBoard*

### 3.3.7. Creación de "Tablero" en ThingsBoard

Se da clic en **Paneles** y aparece el panel de trabajo. Para crear un nuevo **panel**, se oprime en el icono del **signo más** como se presenta en la Figura 46.

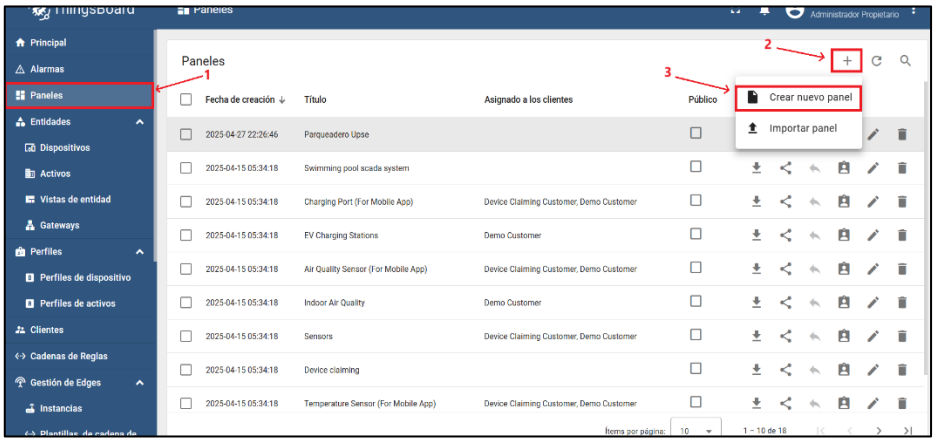


*Figura 46 - Creación de un nuevo Panel*

Al agregar un nuevo panel, se despliega una pantalla (ver Figura 47) donde se ingresan el título y una descripción. Esto permite definir el tema del panel antes de proceder al desarrollo de la interfaz de usuario.

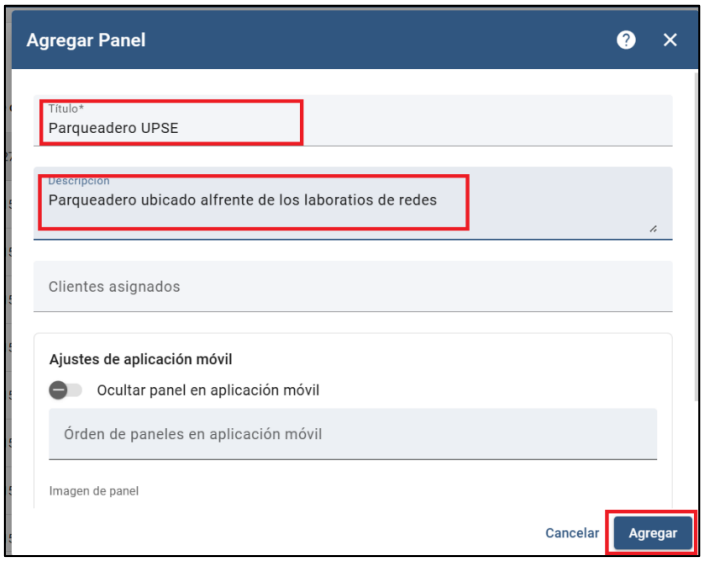


***Figura 47 - Definición del tema para el panel***

En la Figura 48, se presenta el área de construcción de la interfaz del usuario (UI) del panel. En este entorno se añaden y configuran los diversos widgets (gráficos, tablas, indicadores, etc.) para crear una interfaz de usuario personalizada y visualmente atractiva, destinada a mostrar la información relevante al usuario.


***Figura 48 - Área de diseño del panel***

Añade un widget para la visualización de los espacios del parqueadero, selecciona el tipo **Card** (ver Figura 49), se escoge **Value Card** para mostrar cada uno de los valores provenientes de los datos de telemetría.

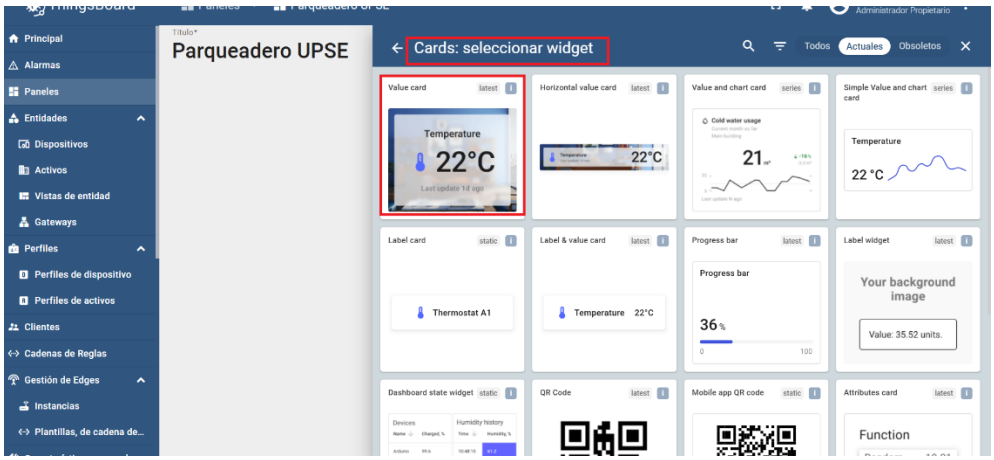


***Figura 49 - Selección del widget***

Crear una **Entity Alias**, con el nombre "Parqueadero". En el tipo de filtro, selecciona Única **entidad**, se elige el dispositivo y el nombre de la entidad correspondiente. Esto permite vincular correctamente la fuente de los datos que se van a visualizar (ver Figura 50).

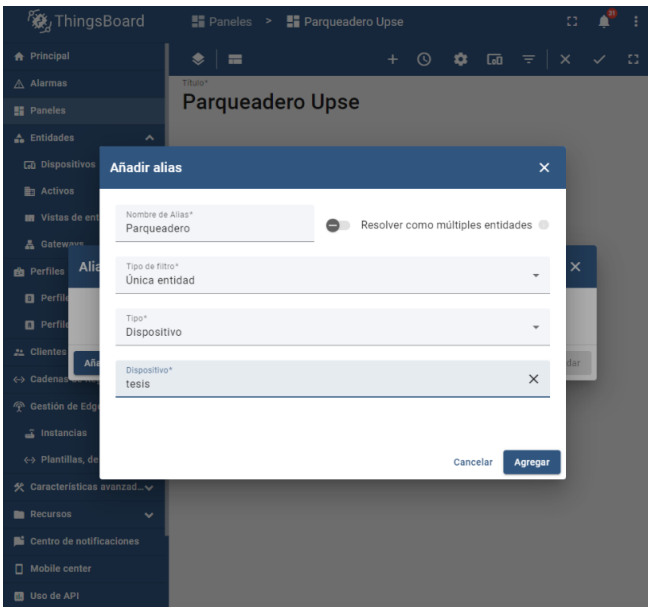


***Figura 50 - Creación del alias de entidad***

La visualización de los datos requiere seleccionar como origen un dispositivo previamente configurado. En este caso, la fuente identificada como **"tesis"** representa el emisor de los datos hacia el panel (ver Figura 51).

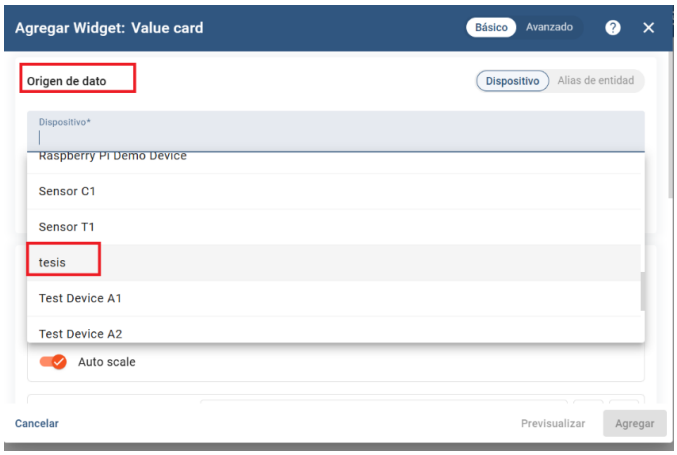


***Figura 51 - Configuración del origen de datos***

Tal como se observa en la Figura 52, al ingresar el origen de datos, se despliegan las **claves de telemetría** correspondientes: occupied_sport, available_sports, spot_1, spot_2 y spot_3.

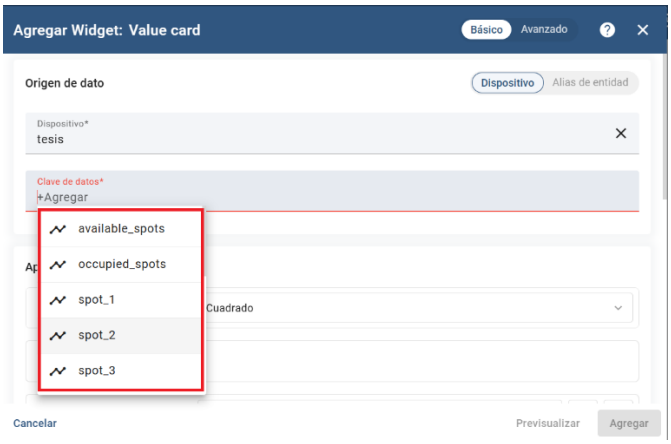


***Figura 52 - Configuración de las claves de datos***

Con los datos del parqueadero disponibles, se añaden widgets individuales para cada valor transmitido por telemetría, optimizando su presentación en la interfaz (ver Figura 53).

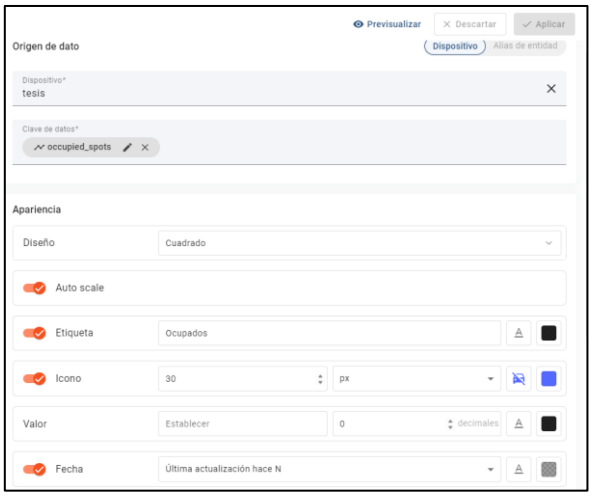


*Figura 53: Distribución de widgets según datos de telemetría*

Se selecciona un nuevo paquete de widget, se escoge **Maps** > **Image Map,** para colocar una imagen referencial del sector de estudio (ver Figura 54).

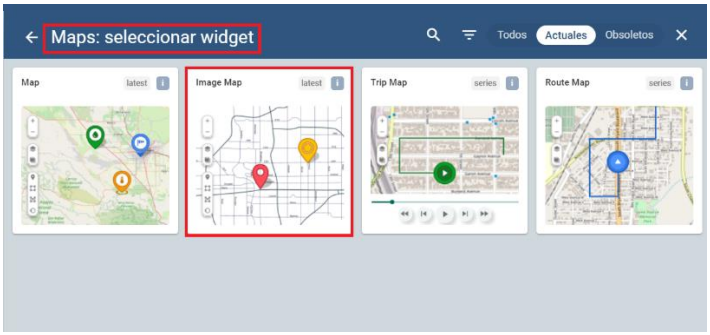


*Figura 54 - Selección de un nuevo widget*

Una vez seleccionado el widget, se modifica el titulo y la imagen referencial de la zona del parqueadero (ver Figura 55).

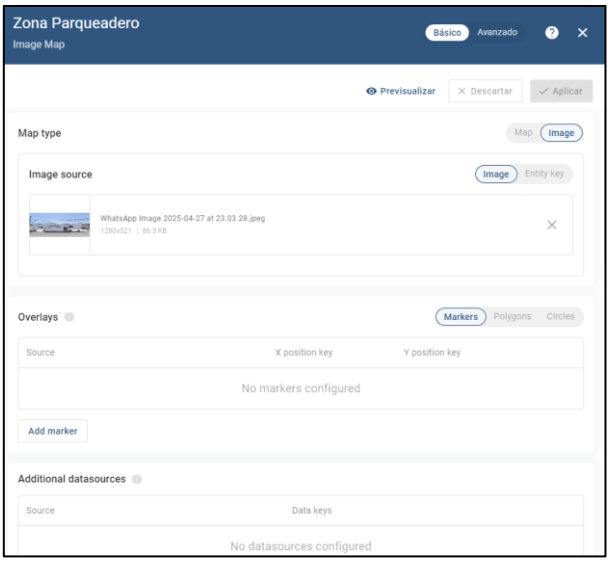


*Figura 55 - Configuración de Image Map*

El panel de control muestra una vista general del sistema, indicando el total de plazas disponibles, el número de plazas actualmente ocupadas, los datos específicos de cada plaza y una imagen de referencia del entorno (ver Figura 56).

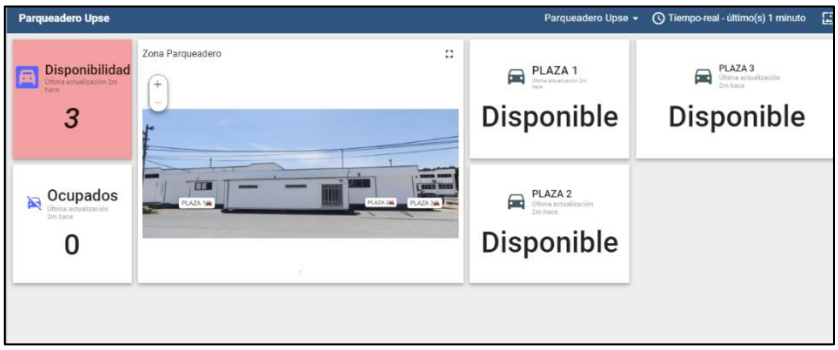


*Figura 56 - Panel de control para visualizar datos del parqueadero*

### 3.3.8. Base de datos

ThingsBoard (TB) utiliza PostgreSQL (o Cassandra/TimescaleDB) como base de datos principal para almacenar la información de los datos de telemetría y la configuración del propio sistema. Una vez conectado, TB crea sus propias tablas, índice y relaciones dentro de la base de datos de terceros.

La función de Node-RED está construyendo un mensaje en el formato admitido por TB para datos de telemetría (JSON con **ts** para timestamp y **values** para valores). Para verificar los valores se ingresa en **Dispositivos** y se va a la opción de **Ultima telemetría** para la verificación (ver Figura 57).

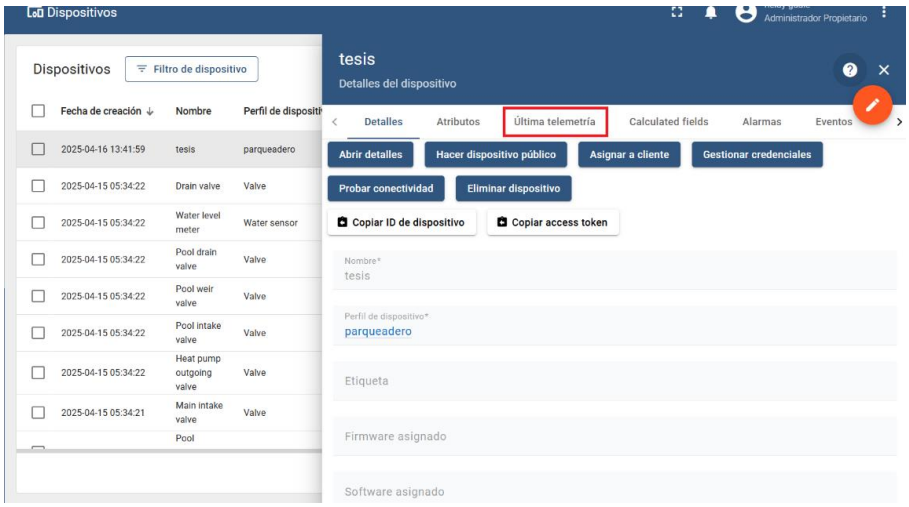


***Figura 57 - Verificación de datos de telemetría en ThingsBoard***

El mensaje pasa a través de motores de reglas de TB, se guardan en forma de nodo llamado Guardar Series de Tiempo (Save Timeseries). Estos nodos toman las claves y **valores** que están enviando (available_spots, occupied_spots, spot_1, spot_2, spot_3) y el **ts** (timestamp) como se presenta en la Figura 58.

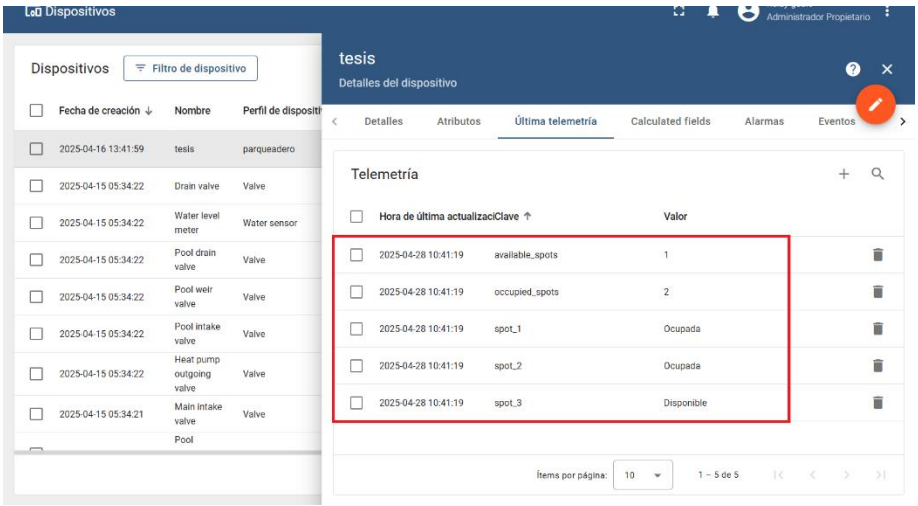


***Figura 58 - Valores y tiempo de llegada de datos***

Al seleccionar cada uno de los datos, se presiona el botón “Mostrar en Widget” para visualizar fácilmente los datos, sin necesidad de programación adicional (ver Figura 59).

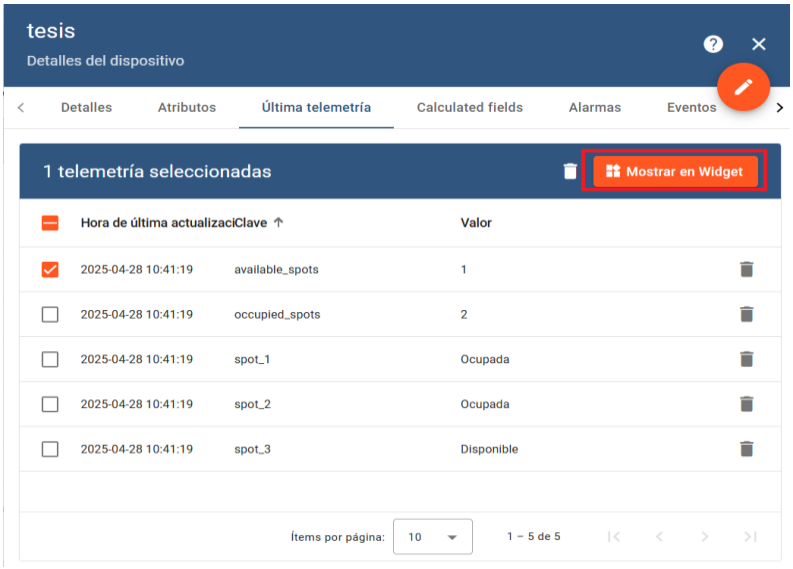


***Figura 59 - Visualización de datos en Widget***

En la figura 60, se presenta el módulo donde se puede consultar y visualizar la información en **tiempo real** e **histórico**. TB inserta estos datos en tablas específicas, donde las tablas más relevantes para la telemetría suelen ser **ts_kv_latest** y **ts_kv.** Se despliega un menú de configuración de la Ventaja de tiempo, que permite definir el rango temporal de los datos históricos a consultar.

- **Ts_ks:** se encarga de almacenar los datos de series de tiempo “históricos”. Cada fila contiene la entidad ID, clave de datos (available_sport), el valor y timestamp.
- **Ts_ks_latest:** se encarga de almacenar el último valor conocido para cada entidad de telemetría. Esto permite recuperar el estado actual del dispositivo.

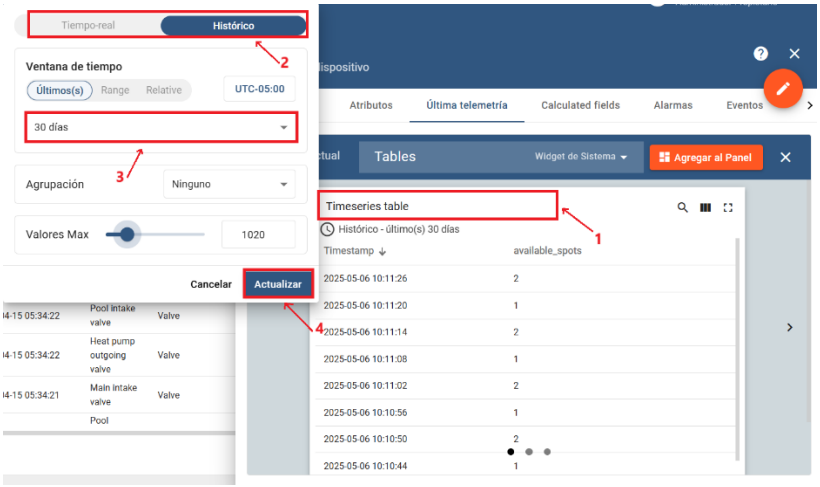


*Figura 60 - Módulo de visualización de datos*

# CAPÍTULO IV

## 4.1.Resultado del entrenamiento del modelo YOLOv5s

Para detectar vehículos de una manera precisa, el modelo debe ser capaz de distinguir entre diferentes tipos de objetos. Una arquitectura de 157 capas permite aprender características esenciales según el escenario expuesto. En la tabla 7, se presentan los resultados de la validación del modelo YOLOv5s, donde se realizó entrenamientos con diferentes cantidades de imágenes que varían según las condiciones climáticas, la distancia y la iluminación.

La tabla está compuesta por los siguientes enunciados:

- **Clases:** nombre de las clases que el modelo detecta (tipos de carros).
- **Imágenes:** número de imágenes utilizadas para la validación.
- **Instancias:** número de total de objetos detectados en las imágenes de validación.
- **Precisión (P):** porcentaje de predicción positiva, su rango es de 0 a 1.
- **Recall (R):** porcentaje de sensibilidad.
- **mAP_50:** precisión media promedio calculado con un umbral de intersección del sobre unión (IoU) 50%.
- **mAP_50-90:** promedio de la precisión media calculada en diferentes umbrales de IoU desde 0.50 hasta 0.90.

***Tabla 7:*** *Resultados de YOLOv5s*

| Clases | Imágenes | Instancias | P | R | mAP 50 | mAP 50-95 |
|---|---|---|---|---|---|---|
| **All** | 63 | 30 | 0.244 | 0.526 | 0.287 | 0.0819 |
| **All** | 150 | 57 | 0.998 | 1 | 0.995 | 0.818 |
| **All** | 450 | 105 | 0.998 | 1 | 0.985 | 0.875 |

La Figura 61, se basa en las gráficas de métricas de mAP_0.5, mAP_0.5:0.95, precisión y recall obtenidas durante la validación del modelo YOLOv5s, tal como se muestra en la Tabla 7. El modelo alcanzó una precisión del 99.8%, lo que indica que prácticamente todas las detecciones positivas fueron correctas, junto con un recall de 1 (100%), evidenciando que el modelo logró detectar la totalidad de los objetos relevantes en las imágenes.

El entrenamiento se realizó durante 50 épocas (eje x), con una duración aproximada de una hora utilizando la plataforma Google Colab, resultando la eficiente del proceso tanto en rendimiento como en recursos computacionales.

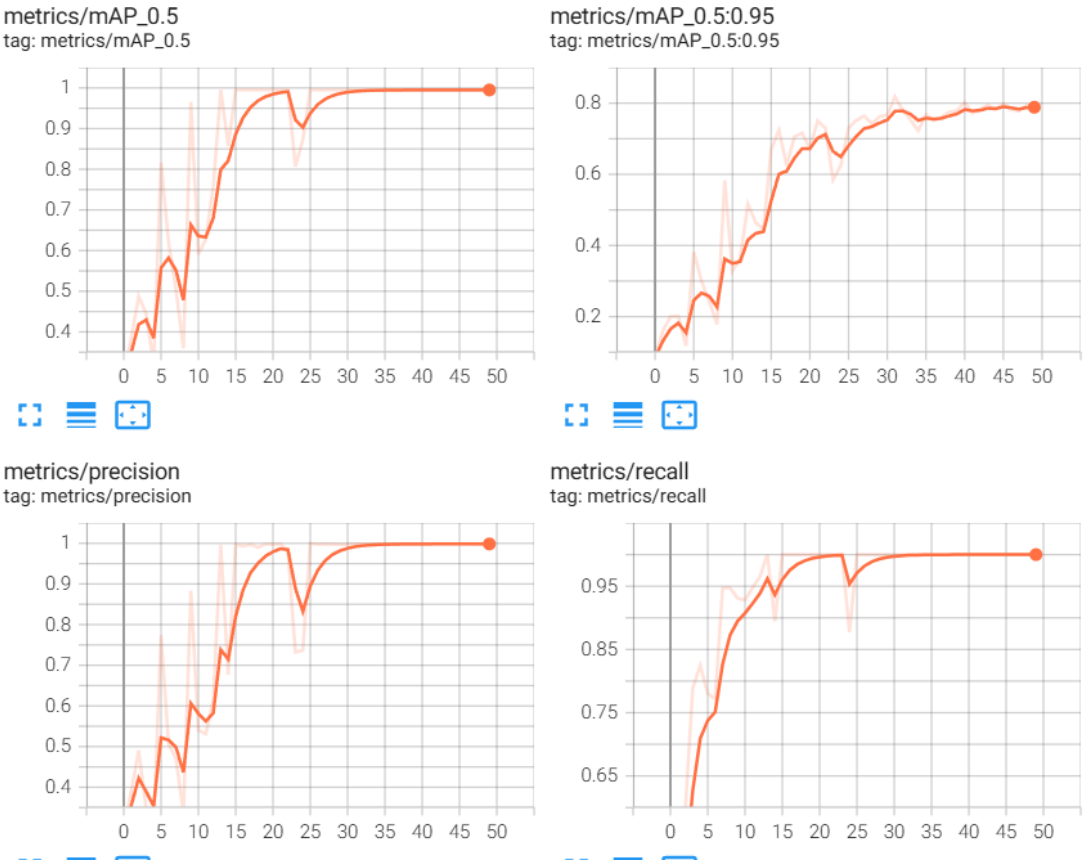


*Figura 61 - Resultados de las métricas del entrenamiento de YOLOv5s*

En la figura 62, se evidencia el funcionamiento correcto del modelo de detección YOLOv5s, esté identifica automóviles a partir de imágenes presentadas en la pantalla de un teléfono móvil, a pesar de tratarse de una imagen secundaria, el modelo logra detectar correctamente los objetos etiquetados como "carros". La predicción con mayor confianza alcanza un valor de 0.64, correspondiente al vehículo más cercano.

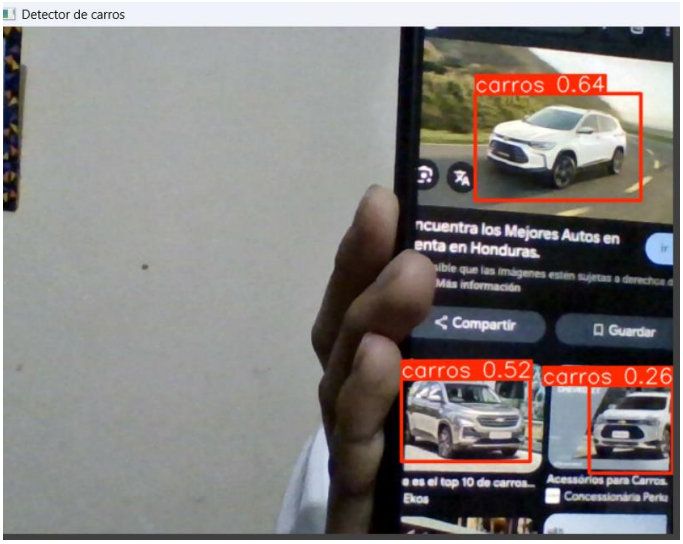


*Figura 62 - Prueba 1 de vehículos mediante YOLOv5s con imágenes de dispositivo móvil*

En la Figura 63, se muestra un segundo experimento utilizando un video grabado en una zona con alta circulación vehicular. Durante el entrenamiento del modelo YOLO, se observa que puede detectar vehículos borrosos con una confianza aproximada de 0.25. A medida que los vehículos se acercan a la cámara y se ven con mayor nitidez, la precisión de detección mejora significativamente.

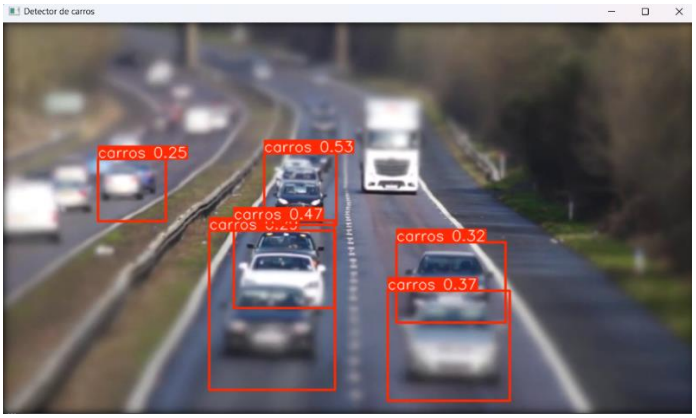


*Figura 63 - Prueba 2 detección de vehículos mediante YOLOv5s con video*

En la Figura 64, se utiliza una cámara en tiempo real. Aunque los valores de precisión varían dinámicamente durante la transmisión, se observa un rendimiento eficiente del modelo, alcanzando una precisión de 0.26 con vehículos ubicados a mayor distancia.


*Figura 64 - Prueba en la transmisión en tiempo real*

## 4.2.Pruebas de algoritmos de detección

### 4.2.1 Algoritmo de Ray Casting

Aunque se propuso el algoritmo de Ray Casting como solución para identificar el espacio ocupado y libre dentro de un polígono, su implementación enfrentó serias limitaciones en entornos de monitoreo en tiempo real, principalmente debido a fallos en la aceptación y manejo eficiente de los fotogramas.

El algoritmo propuesto se basa en determinar la ocupación de una plaza al verificar si el punto central de un vehículo detectado cae dentro de un polígono de estacionamiento. Sin embargo, su implementación en un entorno de tiempo real con una alta frecuencia de frames puede llevar a imprecisiones y falsos positivos. Los problemas radican principalmente en la inestabilidad de las coordenadas del "punto" del vehículo y la sensibilidad del Ray Casting a las condiciones límites y la precisión geométrica, lo que lo convierte en un método poco robusto para transmisión en tiempo real. En la Figura 65, las plazas 1 y 2 se encuentran ocupadas, sin embargo, se presenta la plaza 3 vacía, pero el sistema la detecta erróneamente como ocupada generando falsos positivos. Estos errores, tanto de omisión como de identificación errónea, se traduce en una eficiencia general del método en un 50% para detección de plazas, generando datos pocos fiables y difícil de gestionar.


*Figura 65 - Primera prueba de algoritmo de Ray Casting*

Como se aprecia en la Figura 66, se realizó una segunda prueba del método de Ray Casting para identificar la ocupación de las plazas del estacionamiento. En la visualización de la cámara se aprecia que existe 2 vehículos en la plaza 1 y 2 respectivamente, pero sus áreas se encuentran marcadas como disponibles. En múltiples fotogramas se registran “falsos negativos”, indicando que hay vehículos presentes pero el método no logra reconocer, además del “falso positivo” en la plaza 3 donde se señala la plaza ocupada inexistente. Esta alta tasa de falsos negativos y positivos se refleja en un rendimiento general del código, que genera datos pocos fiables y compromete la eficacia del sistema para asignar plazas.


*Figura 66 - Segunda prueba del algoritmo de Ray Casting*

### 4.2.2 Máscara binaria

La implementación de máscaras binarias combinado con el cálculo del área de intersección de los vehículos y la aplicación de umbrales representa una solución inherentemente más robusta y precisa que el método de Ray Casting para el sistema de detección de estacionamiento en tiempo real (ver figura 67).

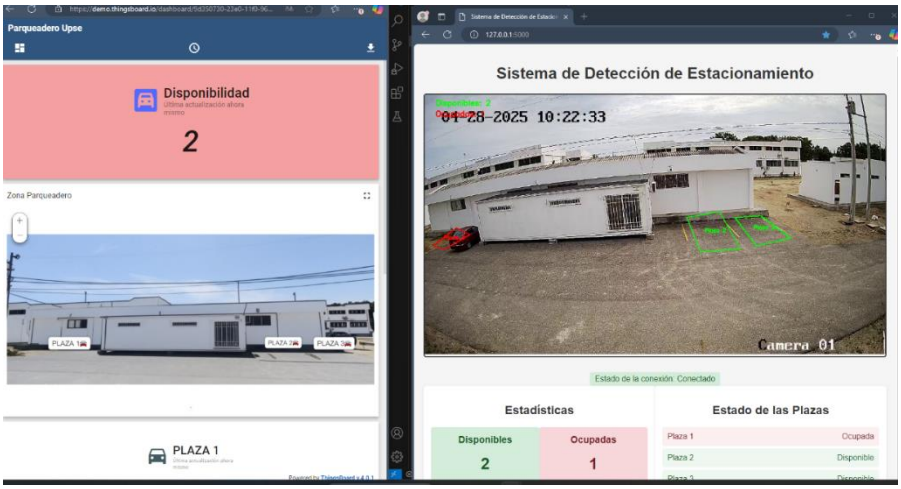


*Figura 67 - Implementación de mascara binaria*

Se realiza un periodo de observación en el que se escoge horarios donde se encuentran ocupadas al menos 2 plazas para comprobar la efectividad del modelo durante la afluencia de vehículos. Él envió de datos a Node-RED sobre el estado del estacionamiento (disponible y ocupada) se envían en un intervalo de aproximadamente 6 segundos (10 datos por minuto), debido a que se requiere actualizaciones instantáneas de esta información, a la vez se presentan en la parte de ultima telemetría en ThingsBoard (ver figura 68).

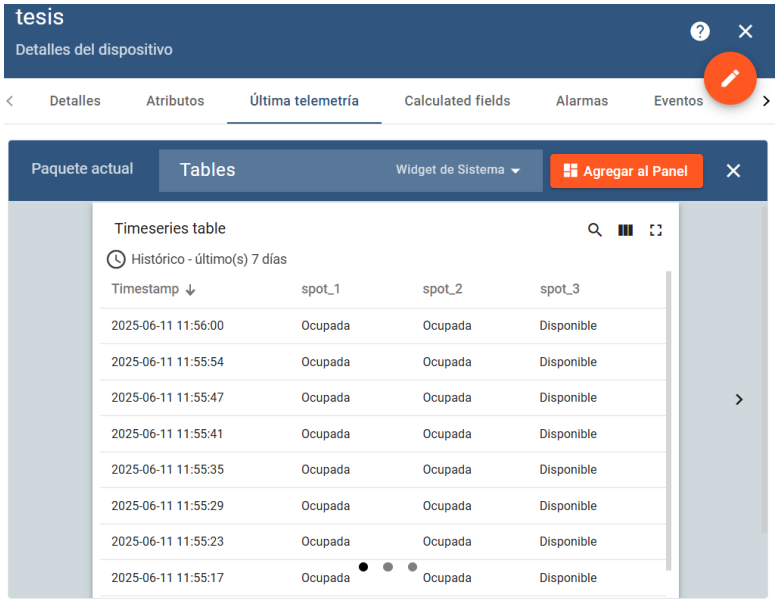


***Figura 68 - Datos de ultima telemetría en ThingsBoard***

Para analizar la eficiencia del modelo, se selecciona el intervalo comprendido entre las 11:00 a.m. y las 12:00 p.m. debido a que, en dicho lapso, se observa una alta concentración de vehículos pertenecientes a docentes y estudiantes que hacen uso del laboratorio. Aunque las actividades académicas inician a partir de las 8:00 a.m., los registros de afluencia vehicular indican un incremento significativo a partir de las 9:00 a.m. hasta las 12:00 pm.

A partir de los datos recopilados, se realiza una matriz de confusión (ver Tabla 8), donde se compara la predicción del modelo con la ocupación real de las plazas, estos valores son obtenidos en la telemetría de ThingsBoard, su intervalo de llegada es de 6 segundos, dando un total de 600 datos (fotogramas) en el transcurso de 1 hora.

Los resultados se clasifican dependiendo de su estado como:

- Verdadero negativo (VN) = plaza estaba ocupada y modelo dijo ocupada.
- Falso Positivo (FN) = plaza estaba ocupada y modelo dijo disponible.
- Verdadero positivo (VP) = plaza estaba disponible y modelo dijo disponible.
- Falso Negativo (FN) = plaza estaba disponible y modelo dijo ocupado.

En la columna 1 se encuentra determinado por la parte Real y en la fila 1 la predicción del algoritmo, cada una establecida por los tiempos (Ocupado (0) y Disponible (1)).

- Cuando la **plaza 1 y 2** están **ocupados**, el modelo detectó 574 Verdaderos Negativos demostrando un alto grado de eficiencia. A pesar de estar ocupado el modelo marcó como **disponible**, generando 26 casos de Falsos Positivos (ver Figura 68).

***Tabla 8:*** *Matriz de confusión*

| **Real / predicción** | **Ocupado (0)** | **Disponible (1)** |
|---|---|---|
| **Ocupado (0)** | VN = 574 | FP = 26 |
| **Disponible (1)** | FN = 0 | VP = 0 |

Con estos valores se realiza el cálculo de la métrica de precisión:

$$Precisión = \frac{VP}{VP + FP} = 95.6\%$$

La precisión del modelo general alcanzó un 95.6%, esto indica un desempeño altamente confiable al momento de predecir si la plaza del estacionamiento se encuentra ocupada. Sin embargo, los falsos positivos se presentan recurrentemente en las plazas 1 y 2, y de manera simultánea en intervalos de tiempo que duran segundos, lo que sugiere que el problema no proviene del modelo de detección en sí, sino a los fallos en la transmisión de video. Esta pérdida de información puede provocar que el modelo reciba imágenes distorsionadas, lo cual afecta directamente su capacidad para detectar las plazas.

Por otro lado, la plaza 3 mostró un comportamiento mucho más estable, logrando una tasa del 100% de verdaderos positivos, lo que implica que en todos los casos en los que estuvo disponible, el modelo lo detectó correctamente (ver Figura 69).


*Figura 69 - Día del estudio para la detección de plazas*

# CONCLUSIONES

La elaboración del sistema de estacionamiento automatizado para vehículos conto de dos partes, la primera del entrenamiento de YOLOv5s para la detección de automóviles, su versión más pequeña de este algoritmo evito consumo de recursos del computador. Él modelo tuvo una precisión del 99.8% y un valor de IoU de 0,875, asegurando la identificación correcta de vehículos a largas distancias y escenarios sujetos a condiciones de iluminación.

La segunda parte con el uso de máscaras binarias para la región de interés (ROI) a partir de la delimitación de para cada una de las plazas, se realiza una intercepción entre está y el área del vehículo con un umbral de 25%, siendo eficiente para enmarcar el recuadro en ocupado o disponible. Los resultados del 95,6% de precisión durante el día de prueba, demuestra una alta fiabilidad para entornos con alto auge de vehículos, con un porcentaje bajo de falsos negativos por problemas de conexión.

Para el procesamiento, se ubica estratégicamente la cámara IP a una altura de 4 metros, que permita un amplio rango de cobertura del sector, el sistema está configurado para una resolución de 1280x720 pixeles a 15 cuadros por segundo (FPS), evitando problemas de carga de datos. En el código de Python se analiza cada cinco fotogramas para evitar cuellos de botellas y otorgar un mejor rendimiento.

Para la presentación de los datos al usuario, se desarrolla paneles con datos en tiempo real en la plataforma ThingsBoard, estos llegan en forma de telemetría provenientes de Node-RED, permite el monitoreo remoto a través de la web, garantizando la confiabilidad por sus actualizaciones de cada 6 segundos. Además, el sistema no tiene sobrecargas de datos siendo adecuada en diferentes entornos.

## RECOMENDACIONES

- Se sugiere utilizar cámaras con mejor estabilidad de conexión y optimizar en entornos de prueba con diferentes condiciones de iluminación para validad su adaptabilidad y robustez.
- Descartar algoritmo de Ray Casting como alternativa, dado que su uso en aplicaciones sensibles a errores no brinda una precisión en la detección de espacios.
- Se recomienda mantener la plataforma ThingsBoard para monitoreo, debido a que permite grandes funciones y no genera sobrecargas de datos en las transmisiones en tiempo real.
- Los puntos de coordenadas para formar polígonos irregulares deben ser amplios, debido a que los conductores parquean sus vehículos de diferentes maneras. Permitiendo que el umbral de confianza sea óptimo para cada espacio.

## Anexos

***Anexo 1:*** *Código envió de datos de Python a Node-RED*

```
def send_to_node_red():
    """Función para enviar datos a Node-RED"""
    global parking_data, last_sent_time
    SEND_INTERVAL = 60.0
    while True:
        try:
            current_time = time.time()


            if current_time - last_sent_time >= SEND_INTERVAL:
                data_to_send = {
                    "available_spots": parking_data["available_spots"],
                    "occupied_spots": parking_data["occupied_spots"],
                    "spots_status": parking_data["spots_status"],
                     "timestamp": int(current_time * 1000)
                }


                try:
                    response = requests.post(
                        NODE_RED_URL,
                        json=data_to_send,
                        headers={"Content-Type": "application/json"},
                        timeout=3
                    )
```

```
            if response.status_code == 200:
                print(f"Datos enviados: {data_to_send}")
            else:
                print(f"Error HTTP {response.status_code}: {response.text}")

            last_sent_time = current_time

        except requests.exceptions.Timeout:
            print("Timeout al conectar con Node-RED")
        except requests.exceptions.RequestException as e:
            print(f"Error de conexión: {str(e)}")

        time.sleep(1)

    except Exception as e:
        print(f"Error crítico en send_to_node_red: {str(e)}")
        time.sleep(5)
```

***Anexo 2:*** *Código fuente del nodo function en Node-RED*

```
// En el nodo function de Node-RED:
var data = msg.payload;
// Usar timestamp directamente (ya está en milisegundos)
var messageTimestamp = data.timestamp || Date.now();
// Crear objeto para ThingsBoard
var msg = {
  payload: {
    ts: messageTimestamp, // <-- Milisegundos ya desde Python
    values: {
      available_spots: Number(data.available_spots) || 0,
      occupied_spots: Number(data.occupied_spots) || 0
    }
  }
};
// Añadir estado de cada plaza (se mantiene igual)
if (Array.isArray(data.spots_status)) {
  data.spots_status.forEach(spot => {
    if (spot.id && spot.status) {
      msg.payload.values['spot_'+spot.id] = spot.status;
    }
  });
}
return msg;
```

***Anexo 3:*** *Código fuente del parqueadero con algoritmo computacionales*

```
import cv2
import torch
import numpy as np
from flask import Flask, Response, jsonify
import threading
import time
import requests
import json

app = Flask(__name__)
#  cámara
CAMERA_IP = "…… "
CAMERA_USER = "……..."
CAMERA_PASS = "……"
CAMERA_URL =
f"rtsp://{CAMERA_USER}:{CAMERA_PASS}@{CAMERA_IP}:554/Stream
ing/Channels/1?tcp"

# Configuración de Node-RED
NODE_RED_URL = "http://localhost:1880/api/parking"
SEND_INTERVAL = 0.50
# Variables globales
global_frame = None
parking_data = {
    "available_spots": 0,
    "occupied_spots": 0,
    "spots_status": []
}
last_sent_time = 0
cap = None
# modelo Yolo
model = torch.hub.load('ultralytics/yolov5', 'custom',
path='C:/Users/HeidyGV/Documents/modeloyolo/best.pt',
                       force_reload=True)
model.conf = 0.3  # Umbral de confianza
model.iou = 0.5
# Definir plazas (verificar coordenadas)
parking_spots = [
    {
        "id": 1,
        "polygon”: [56, 277], [1, 348], [47, 364], [139,
286]]
    },
    {
        "id": 2,
        "polygon”: [708, 216], [744, 319], [866, 304],
[824, 208]]
    },
    {
```

```
        "id": 3,
        "polygon”: [849, 218], [895, 307], [1010, 293],
[931, 223]]
    }
]


def initialize_camera():
    """Inicializar cámara"""
    global cap
    try:
        cap = cv2.VideoCapture(CAMERA_URL)
        if not cap.isOpened():
            print("Error: No se pudo abrir la cámara")
            return False

        #   cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)
        #  cap.set(cv2.CAP_PROP_FPS, 15)

        print("Cámara inicializada correctamente")
        return True
    except Exception as e:
        print(f"Error al inicializar cámara: {e}")
        return False


def check_spot_occupation(vehicle_bbox, polygon):
    """Verificar si un vehículo ocupa una plaza"""
    x_min, y_min, x_max, y_max = vehicle_bbox

    # Create a blank mask and draw the polygon on it
    mask = np.zeros((max(y_max + 1, 1000), max(x_max + 1,
1000)), dtype=np.uint8)
    poly_pts = np.array(polygon, np.int32).reshape((-1, 1,
2))
    cv2.fillPoly(mask, [poly_pts], 1)

    # Create a mask for the vehicle bounding box
    vehicle_mask = np.zeros_like(mask)
    cv2.rectangle(vehicle_mask, (x_min, y_min), (x_max,
y_max), 1, -1)

    # Calculate intersection
    intersection = cv2.bitwise_and(mask, vehicle_mask)
    intersection_area = np.sum(intersection)

    # If intersection area exceeds threshold, consider it
occupied
    vehicle_area = (x_max - x_min) * (y_max - y_min)
    min_overlap_ratio = 0.40  # Vehicle overlap ratio
```

```
    return intersection_area / vehicle_area >
min_overlap_ratio


def send_to_node_red():
    """Enviar datos a Node-RED cada minuto"""
    global parking_data, last_sent_time

    while True:
        try:
            current_time = time.time()

            if current_time - last_sent_time >=
SEND_INTERVAL:
                data_to_send = {
                    "available_spots":
parking_data["available_spots"],
                    "occupied_spots":
parking_data["occupied_spots"],
                    "spots_status":
parking_data["spots_status"],
                    "timestamp": int(current_time * 1000)
                }

                try:
                    response = requests.post(
                        NODE_RED_URL,
                        json=data_to_send,
                        headers={"Content-Type":
"application/json"},
                        timeout=3
                    )

                    if response.status_code == 200:
                        print(f"Datos enviados:
{data_to_send}")
                    else:
                        print(f"Error HTTP
{response.status_code}: {response.text}")

                    last_sent_time = current_time

                except requests.exceptions.Timeout:
                    print("Timeout al conectar con Node-
RED")
                except requests.exceptions.RequestException
as e:
                    print(f"Error de conexión: {str(e)}")

            time.sleep(1)
```

```
        except Exception as e:
            print(f"Error crítico en send_to_node_red:
{str(e)}")
            time.sleep(5)


def camera_and_detector():
    """Función combinada que captura y procesa
directamente"""
    global global_frame, parking_data, cap

    # Inicializar cámara
    if not initialize_camera():
        return
    frame_count = 0
    process_every_n_frames = 5
    print("Iniciando captura y detección...")
    while True:
        try:
            # Capturar frame directamente
            ret, frame = cap.read()

            if not ret:
                print("Error al capturar frame,
reintentando...")
                # Reinicializar cámara si falla
                cap.release()
                time.sleep(2)
                if not initialize_camera():
                    time.sleep(5)
                    continue
                continue

            frame_count += 1

            # Procesar solo cada N frames para reducir
carga
            if frame_count % process_every_n_frames == 0:
                try:
                    # Procesar frame con el modelo
                    results = model(frame)
                    detections = results.pandas().xyxy[0]

                    # Filtrar por confianza y clases de
vehículos
                    vehicles = model
                    occupied_spots = 0
                    spots_status = []

                    # Verificar ocupación de cada plaza
                    for spot in parking_spots:
```

```
                        spot_id = spot["id"]
                        polygon = spot["polygon"]
                        spot_occupied = False

                        for _, vehicle in
vehicles.iterrows():
                            bbox = [
                                int(vehicle['xmin']),
                                int(vehicle['ymin']),
                                int(vehicle['xmax']),
                                int(vehicle['ymax'])
                            ]

                            if check_spot_occupation(bbox,
polygon):
                                spot_occupied = True
                                break

                        # Actualizar estado de la plaza
                        if spot_occupied:
                            occupied_spots += 1
                            spots_status.append({"id":
spot_id, "status": "Ocupada"})
                        else:
                            spots_status.append({"id":
spot_id, "status": "Disponible"})

                    available_spots = len(parking_spots) -
occupied_spots

                    # Actualizar datos globales
                    parking_data = {
                        "available_spots": available_spots,
                        "occupied_spots": occupied_spots,
                        "spots_status": spots_status,
                        "timestamp": time.time()
                    }

                except Exception as e:
                    print(f"Error al procesar con el
modelo: {e}")
                    # Continuar sin procesar pero mostrando
el frame
                    vehicles = []
            else:
                # Si no procesamos, usar datos del frame
anterior
                vehicles = []

            # Crear frame de salida con visualizaciones
            output_frame = frame.copy()
```

```
            # Dibujar bounding boxes de vehículos
detectados
            if frame_count % process_every_n_frames == 0:
                try:
                    for _, vehicle in vehicles.iterrows():
                        x1, y1, x2, y2 =
int(vehicle['xmin']), int(vehicle['ymin']),
int(vehicle['xmax']), int(
                            vehicle['ymax'])
                        cv2.rectangle(output_frame, (x1,
y1), (x2, y2), (255, 0, 0), 2)
                        cv2.putText(output_frame,
f"{vehicle['name']}: {vehicle['confidence']:.2f}",
                                    (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
                except:
                    pass

            # Dibujar plazas de estacionamiento
            for spot in parking_spots:
                pts = np.array(spot["polygon"], np.int32)
                # Determinar color basado en estado actual
                color = (0, 0, 255) if any(
                    s['id'] == spot['id'] and s['status']
== 'Ocupada' for s in parking_data["spots_status"]) else
(0,

255,

0)
                cv2.polylines(output_frame, [pts], True,
color, 2)

                # Agregar etiqueta de plaza
                sum_x = sum(p[0] for p in spot["polygon"])
                sum_y = sum(p[1] for p in spot["polygon"])
                centroid = (sum_x // len(spot["polygon"]),
sum_y // len(spot["polygon"]))
                cv2.putText(output_frame, f"Plaza
{spot['id']}",
                            (centroid[0] - 30,
centroid[1]),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5,
color, 2)

            # Mostrar estadísticas
            cv2.putText(output_frame, f"Disponibles:
{parking_data['available_spots']}",
                        (30, 30), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (0, 255, 0), 2)
```

```
            cv2.putText(output_frame, f"Ocupadas:
{parking_data['occupied_spots']}",
                        (30, 60), cv2.FONT_HERSHEY_SIMPLEX,
0.7, (0, 0, 255), 2)

            # Actualizar frame global para streaming
            global_frame = output_frame

        except Exception as e:
            print(f"Error en captura/detección: {e}")
            time.sleep(1)


def generate_frames():
    """Generador para streaming web"""
    global global_frame

    while True:
        if global_frame is not None:
            try:
                _, buffer = cv2.imencode('.jpg',
global_frame, [cv2.IMWRITE_JPEG_QUALITY, 80])
                frame_bytes = buffer.tobytes()
                yield (b'--frame\r\n'
                       b'Content-Type: image/jpeg\r\n\r\n'
+ frame_bytes + b'\r\n')
            except Exception as e:
                print(f"Error al generar frame: {e}")
                time.sleep(0.1)
        else:
            time.sleep(0.1)


# Rutas Flask
@app.route('/video_feed')
def video_feed():
    return Response(generate_frames(),
                    mimetype='multipart/x-mixed-replace;
boundary=frame')


@app.route('/parking_status')
def parking_status():
    return jsonify(parking_data)


@app.route('/')
def index():
    return """
    <!DOCTYPE html>
    <html>
```

```
    <head>
        <title>Sistema de Monitoreo de
Estacionamiento</title>
        <style>
            body { font-family: Arial, sans-serif; margin:
20px; }
            .container { max-width: 1200px; margin: 0 auto;
}
            .video-container { text-align: center; margin:
20px 0; }
            .status-panel {
                background: #f5f5f5;
                padding: 20px;
                margin: 20px 0;
                border-radius: 8px;
            }
            .status-item { margin: 10px 0; font-size: 18px;
}
            .available { color: green; font-weight: bold; }
            .occupied { color: red; font-weight: bold; }
            img { max-width: 100%; height: auto; border:
2px solid #ddd; }
        </style>
    </head>
    <body>
        <div class="container">
            <h1>Sistema de Monitoreo de
Estacionamiento</h1>

            <div class="video-container">
                <img src="/video_feed" alt="Live Video
Feed">
            </div>

            <div class="status-panel">
                <h2>Estado del Estacionamiento</h2>
                <div id="status-info">
                    <div class="status-
item">Cargando...</div>
                </div>
            </div>
        </div>

        <script>
            // Actualizar estado cada 5 segundos
            setInterval(function() {
                fetch('/parking_status')
                    .then(response => response.json())
                    .then(data => {
                        const statusDiv =
document.getElementById('status-info');
```

```
                        statusDiv.innerHTML = `
                            <div class="status-item
available">Plazas Disponibles:
${data.available_spots}</div>
                            <div class="status-item
occupied">Plazas Ocupadas: ${data.occupied_spots}</div>
                            <div class="status-item">Total
de Plazas: ${data.spots_status.length}</div>
                            <div class="status-item">Última
Actualización: ${new Date().toLocaleTimeString()}</div>
                        `;
                    })
                    .catch(error => {
                        console.error('Error:', error);
                        document.getElementById('status-
info').innerHTML =
                            '<div class="status-item">Error
al obtener datos</div>';
                    });
            }, 5000);
        </script>
    </body>
    </html>
    """


if __name__ == '__main__':
    try:
        # Iniciar hilo para envío de datos a Node-RED
        node_red_thread =
threading.Thread(target=send_to_node_red, daemon=True)
        node_red_thread.start()

        # Iniciar hilo para captura y detección
        camera_thread =
threading.Thread(target=camera_and_detector, daemon=True)
        camera_thread.start()

        # Iniciar servidor Flask
        print("Iniciando servidor Flask en puerto 5000...")
        app.run(host='0.0.0.0', port=5000, debug=False,
threaded=True)

    except KeyboardInterrupt:
        print("Cerrando aplicación...")
        if cap:
            cap.release()
    except Exception as e:
        print(f"Error al iniciar la aplicación: {e}")
        if cap:
            cap.release()
```

## Bibliografía


[1] C. A. Erazo Entrada, “PROTOTIPO DE DETECCION DE APARCAMIENTOS LIBRES MEDIANTE VISION ARTIFICIAL EN UN PARQUEADERO DE LA UNIVERSIDAD TECNICA DEL NORTE,” Ibarra, Apr. 2019.

[2] A. M. Cedeño-Luna, S. M. Vasquez-Camuendo, and J. J. García-Vinces, “Análisis del flujo vehicular y peatonal en la Universidad Técnica de Manabí.,” *MQRInvestigar*, vol. 8, no. 3, pp. 2779–2802, Aug. 2024, doi: 10.56048/MQR20225.8.3.2024.2779-2802.

[3] D. Rivera Arroyave, J. Jaramillo Tobón, R. Arcila Rodriguez, and D. Múnera, “SmartParkUdeA : Sistema IoT para el estacionamiento inteligente de vehículos en ciudad universitaria,” in *IEEE Colombian Conference in Communications and Computing (COLCOM2020) student track*, 2020.

[4] X. Ding and R. Yang, “Vehicle and Parking Space Detection Based on Improved YOLO Network Model,” in *Journal of Physics: Conference Series*, 2019. doi: 10.1088/1742-6596/1325/1/012084.

[5] Ronald Alexis Segarra Guzman, “Diseño de un sistema de parking automatico mediante tecnicas de visión artificial,” Universidad Católica de Cuenca, Cuenca, 2022.

[6] Diego Javier Veintimilla Portilla and Yonder Fernando Siguencia Carrillo, “Diseño de un sistema inteligente de parqueo vehicular mediante videograbación e implementación de un prototipo de prueba para la FIEE,” Escuela Politécnica Nacional , Quito , 2014.

[7] INEC, “Anuario de Estadísticas de Transporte, 2023,” Ecuador, Aug. 2024.

[8] Redacción Primicias, “Casi 30.000 vehículos regresan de Santa Elena a Guayaquil entre el 3 y 4 de enero,” https://www.primicias.ec/sociedad/vehiculos-regreso-santa-elena-guayaquil-enero-feriado-anio-nuevo-86743/.

[9] D. Fuster, “Investigación cualitativa: Método fenomenológico hermenéutico Qualitative Research: Hermeneutical Phenomenological Method,” *Propósitos y Representaciones*, vol. 7, no. 1, 2019.

[10] L. Quintana and J. Hermida, “La hermenéutica como método de interpretación de textos en la investigación psicoanalítica,” 2019.

[11] telecom, “Smart Parking: ¿Qué es y cuáles son sus beneficios?,” https://www.telecom.com.ar/web/blog/smart-parking.

[12] G. P. C. P. da Luz, G. M. Sato, L. F. G. Gonzalez, and J. F. Borin, “Smart Parking with Pixel-Wise ROI Selection for Vehicle Detection Using YOLOv8, YOLOv9, YOLOv10, and YOLOv11,” Dec. 2024.

[13] M. Quiñones, V. Gonazález, L. Quinoñes, C. Valdivieso, and W. Yaguana, “Diseño de un Sistema de Aparcamiento Inteligente Usando una Red de Sensores Inalámbricos,” *2015 10th Iberian Conference on Information Systems and Technologies, CISTI 2015*, 2015.

[14] J. Prieto Blázquez, *Introducción a los Sistemas de Comunicación inalámbricos*. 2015. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/40184329/Tecnologia_y_desarrollo_en_dispositivos_moviles_Modulo_1-libre.pdf?1447975425=&response-content-disposition=inline%3B+filename%3DTecnologia_y_desarrollo_en_dispositivos.pdf&Expires=1710948355&Signature=RW69F0

[15] Omnitron Systems Technology, “ Smart Parking Lot Network Connectivity,” https://www.omnitron-systems.com/smart-parking-lot-network-connectivity.

[16] L. Barba-Guaman, J. E. Naranjo, and A. Ortiz, “Deep learning framework for vehicle and pedestrian detection in rural roads on an embedded GPU,” *Electronics (Switzerland)*, vol. 9, no. 4, 2020, doi: 10.3390/electronics9040589.

[17] David Salazar-Solorzano, Jorge Flores-Ordoñez, and Luis Barba-Guaman, “Técnicas de visión por computadora para la estimación de la intensidad, la velocidad y la densidad en el flujo vehicular,” Loja , 2023.

[18] D. S. Gualoto López, ““Estimacion de Numero de Personas en Imagenes de Multitudes usando Aprendizaje de Maquina,” Ingenieria en Mecatronica, Universidad Tecnica del Norte, Ibarra, 2021.

[19] I. Aguado López, “Deep Learning: Redes Neuronales Convolucionales en R,” *Universidad de Sevilla*, 2020.

[20] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” 2018. doi: 10.1007/s13244-018-0639-9.

[21] L. Tan, T. Huangfu, L. Wu, and W. Chen, “Comparison of YOLO v3, Faster R-CNN, and SSD for Real-Time Pill Identification,” Jul. 30, 2021. doi: 10.21203/rs.3.rs-668895/v1.

[22] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection,” Feb. 2018.

[23] Esri Developer, “How RetinaNet works?,” https://developers.arcgis.com/python/latest/guide/how-retinanet-works/#:~:text=How%20RetinaNet%20works?-,Introduction,with%20aerial%20and%20satellite%20imagery.

[24] Rohit Kundu, “YOLO: Algorithm for Object Detection Explained [+Examples],” https://www.v7labs.com/blog/yolo-object-detection.

[25] Y. Zhang, Z. Guo, J. Wu, Y. Tian, H. Tang, and X. Guo, “Real-Time Vehicle Detection Based on Improved YOLO v5,” *Sustainability*, vol. 14, no. 19, p. 12274, Sep. 2022, doi: 10.3390/su141912274.

[26] Glenn Jocher, “Ultralytic/Yolov5,” https://github.com/ultralytics/yolov5?tab=readme-ov-file.

[27] Rubén Molano Gómez, “Geometría computacional aplicada al reconocimiento de regiones de interés en visión por computador,” Universidad de Extremadura, España, 2024.

[28] Computer Geekery, “Ray Casting Algorithm,” http://www.phillipplemons.com/posts/ray-casting-algorithm.

[29] C. Martens and M. Bessmeltsev, “One-Shot Method for Computing Generalized Winding Numbers,” Aug. 2024.

[30] Yiren Lu, “Top image segmentation models,” https://modal.com/blog/image-segmentation-article.

[31] R. Kashyapa, “Parts Of A Machine Vision System,” https://qualitastech.com/image-acquisition/parts-of-a-machine-vision-system/.

[32] ThingsBoard, “Open-source IoT Platform,” https://thingsboard.io/.

[33] C. Ashhwath, V. Rohitram, and G. Sumathi, “Smart Parking System using MQTT Communication Protocol and IBM Cloud,” *J Phys Conf Ser*, vol. 2115, no. 1, p. 012013, Nov. 2021, doi: 10.1088/1742-6596/2115/1/012013.

[34] J. Torres Ventura, A. H. Ruelas Puente, and J. R. Herrera García, “Rendimiento para la interoperabilidad entre Rasperry pi, ESP8266 y PLC con Node-RED para el IIoT,” *Ingenius*, no. 29, pp. 90–97, Jan. 2023, doi: 10.17163/ings.n29.2023.08.

[35] Alicia, “RTSP: Real-Time Streaming Protocol Overview 2025,” https://reolink.com/blog/what-is-rtsp/?srsltid=AfmBOopSrJRCUac_Ma-09_czzTy2AToLutDgCesmMODjSuP27P0vFsuL.

[36] Jacob Yoss, “UDP vs. TCP and Which One to Use for Video Streaming (Update).”

[37] B. T. M. S. A. A. S. R. David Gourley, “TCP Connections,” https://www.oreilly.com/library/view/http-the-definitive/1565925092/ch04s01.html#:~:text=When%20HTTP%20wants%20to%20transmit,see%20Figure%204%2D4).