Instituto Tecnológico de Costa Rica

Área Académica de Ingeniería Mecatrónica

Diseño y validación de algoritmos de detección de objetos para la capa de percepción del Sistema Avanzado de Asistencia para Conducción (ADAS) en vehículos autónomos

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Mecatrónica con el

grado académico de Licenciatura.

Cristofher Solís Jiménez

2018109433

Cartago, junio de 2023

1.1 Declaratoria de Autenticidad

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema y asesoramiento técnico de miembros de NI Costa Rica.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Cartago, junio 2023

Cristofher Solís Jiménez
Céd: 1-1777-0828

1.2 Resumen

En el presente informe de proyecto final de graduación se presenta el diseño y validación de de algoritmos de detección de objetos para la capa de percepción del Sistema Avanzado de Asistencia para Conducción (ADAS) en vehículos autónomos, este tiene el fin de permitir realizar pruebas de concepto y validación de algoritmos que serán utilizados en sistemas ADAS comerciales.

El sistema diseñado se trata de un software capaz de identificar y rastrear objetos en tiempo real, se enfoca en objetos de principal relevancia para la aplicación de automóviles autónomos (Autos, Peatones, Ciclistas). El cual se validó por medio de una técnica conocida como "*Software in the Loop*", en la cual se recreó un ambiente virtual diseñado con la herramienta llamada NI Monodrive, que simula un automóvil recorriendo una ruta y respondiendo ante los diferentes obstáculos que se presenten en dicha simulación. El sistema tiene una relación directa con el sistema de control que posee la unidad de control electrónica (ECU) el cual se encarga de tomar decisiones de control dependiendo de las salidas obtenidas por la capa de percepción diseñada.

Se planteó un análisis económico derivado del beneficio del proyecto en donde se consideran beneficios suaves y duros e indicadores financieros para evaluar la rentabilidad del proyecto, como lo son el TIR Y VAN.

Palabras clave: Python, C++, Monodrive, Detección, Rastreo, Precisión, Control, Software in the Loop, ADAS, ECU, Visión Artificial.

1.3 Abstract

In this final graduation project report, the design and validation of object detection algorithms for the perception layer of the Advanced Driver Assistance System (ADAS) in autonomous vehicles are presented. The objective is to enable proof of concept and validation of algorithms that will be used in commercial ADAS systems.

The designed system is a software capable of identifying and tracking objects in real time, focusing on objects of primary relevance for autonomous vehicle applications (Cars, Pedestrians, Cyclists). It was validated using a technique known as "Software in the Loop," in which a virtual environment was recreated using the tool called NI Monodrive. This virtual environment simulates a car traveling along a route and responding to various obstacles that arise during the simulation. The system has a direct relationship with the control system of the electronic control unit (ECU), which makes control decisions based on the outputs obtained from the designed perception layer.

An economic analysis was conducted to derive the project's benefits, considering both soft and hard benefits and financial indicators to evaluate the project's profitability, such as the Internal Rate of Return (IRR) and Net Present Value (NPV).

Keywords: Python, C++, Monodrive, Detection, Tracking, Accuracy, Control, Software in the Loop, ADAS, ECU, Computer Vision.

1.4 Agradecimiento

Expreso mi agradecimiento formalmente a NICR y a sus profesionales por brindarme la oportunidad de aprender y crecer profesionalmente por medio de este proyecto de graduación, en especial a aquellos que estuvieron ahí para ayudarme en el proceso.

A mi tutor de proyecto y profesor por su disposición a ayudar y la gran atención e interés que mostró a lo largo del proceso, muchas gracias por ayudarme a que esto fuese posible y gracias por procurar un buen proceso de aprendizaje en cada oportunidad.

De manera sencilla, pero con la mayor de mis gratitudes, a mi familia y amigos por su cariño, apoyo y ánimos a lo largo de mi vida, es en gran parte a ustedes que soy la persona y profesional que soy hoy en día.

Lista de contenidos

2 Tabla de contenido

1.1 Declaratoria de Autenticidad 10

1.2 Resumen 11

1.3 Abstract 12

1.4 Agradecimiento 13

2.1 Lista de abreviaciones 7

1 Introducción 1

1.1 Entorno del Proyecto 1

1.2 Definición del problema 3

1.3 Justificación 5

1.4 Síntesis del problema 6

1.5 Objetivo General 6

1.6 Objetivos Específicos 7

2 Marco Teórico 8

2.1 Conducción Automatizada 8

2.1.1 Unidades de Control Electrónicas (ECU) 8

2.1.2 Sistemas ADAS 10

2.1.3 Capa de percepción de sistema AEB 11

2.1.4 Inteligencia artificial 14

2.1.5 Visión Artificial 18

2.1.6 Simulación de escenarios 19

2.1.7 Sistemas de Control para aplicaciones ADAS 20

2.1.8 Normativa 21

3 Metodología 23

3.1 Planeación 23

3.1.1 Identificar las necesidades del cliente 23

3.1.2 Estado del arte 26

3.1.3 Alternativas de solución 38

3.1.4 Generación de conceptos 39

4 Propuesta de Diseño 48
4.1 Entrenamiento de algoritmos de IA para los sistemas de detección 48
4.2 Diseño del rastreo de objetos 50
4.3 Cálculo de distancias en las imágenes 54
4.4 Generación del Entorno virtual para validación 58
4.5 Integración con capa de control 60
4.6 Optimizaciones 61
4.7 Proceso de validación 63
5 Resultados y Análisis 63
5.1 Análisis de Resultados: 63
5.1.1 Prueba de Validación 1: 63
5.1.2 Prueba de validación 2 71
5.1.3 Prueba de validación 3 79
5.1.4 Prueba de validación 4 92
5.1.5 Prueba de validación 5 94
5.1.6 Reflexión final sobre los resultados: 94
5.2 Análisis financiero 95
6 Conclusiones y Recomendaciones 98
6.1 Conclusiones 98
6.2 Recomendaciones 100
7 Referencias Bibliográficas 101
8 Apéndices 106

Lista de figuras

Figura 1. 1 Diagrama Causa - Efecto 6

Figura 2. 1 Ciclo de trabajo del algoritmo de toma de decisiones 14
Figura 2. 2 Ejemplo de gráficas de pérdidas y de precisión [21] 16
Figura 2. 3 Ejemplo de matriz de confusión [21] 18

Figura 3. 1 División en subproblemas 26
Figura 3. 2 Semejanza de triángulos utilizada en el método triangular. [42] 34

Figura 4. 1 Imágenes de referencia para la clase de automóvil (a) a 5 metros de distancia, b) a 3 metros de distancia y c) a 1 metro) 56
Figura 4. 2 Imágenes de referencia para la clase de ciclista (a) a 5 metros de distancia, b) a 3 metros de distancia y c) a 1 metro) 57
Figura 4. 3 Imágenes de referencia para la clase de automóvil (a) a 5 metros de distancia, b) a 3 metros de distancia y c) a 1 metro) 58
Figura 4. 4 Editor de escenarios de Monodrive 59
Figura 4. 5 Ejemplos de las técnicas de aumentación de datos utilizadas (a) Exposición a la luz, b) Cambios de brillo, c) Agregar ruido en la imagen y d) Saturación) 62

Figura 5. 1 Pérdidas con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la primera solución. 65
Figura 5. 2 Precisión con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la primera solución. 65
Figura 5. 3 Matriz de confusión para el entrenamiento de la mejor combinación de hiperparámetros para la primera solución. 66
Figura 5. 4 Pérdidas con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la segunda solución. 67
Figura 5. 5 Precisión con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la segunda solución. 67
Figura 5. 6 Matriz de confusión para el entrenamiento de la mejor combinación de hiperparámetros para la segunda solución. 68
Figura 5. 7 Pérdidas con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la tercera solución. 69
Figura 5. 8 Figura 16. Precisión con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la tercera solución. 69
Figura 5. 9 Matriz de confusión para el entrenamiento de la mejor combinación de hiperparámetros para la tercera solución. 70
Figura 5. 10 Ejemplo de imagen sencilla 71
Figura 5. 11 Ejemplo de imagen moderadamente compleja 72
Figura 5. 12 Ejemplo de imagen compleja 72
Figura 5. 13 Matriz de confusión para la primera solución utilizando el conjunto de datos de imágenes sencillas 73

Figura 5. 14 Matriz de confusión para la primera solución utilizando el conjunto de datos de imágenes moderadamente complejas. 74
Figura 5. 15 Matriz de confusión para la primera solución utilizando el conjunto de datos de imágenes complejas. 74
Figura 5. 16 Matriz de confusión para la segunda solución utilizando el conjunto de datos de imágenes sencillas 75
Figura 5. 17 Matriz de confusión para la segunda solución utilizando el conjunto de datos de imágenes moderadamente complejas 76
Figura 5. 18 Matriz de confusión para la segunda solución utilizando el conjunto de datos de imágenes complejas 76
Figura 5. 19 Matriz de confusión para la tercera solución utilizando el conjunto de datos de imágenes sencillas 77
Figura 5. 20 Matriz de confusión para la tercera solución utilizando el subconjunto de datos de imágenes moderadamente complejas 78
Figura 5. 21 Matriz de confusión para la tercera solución utilizando el subconjunto de datos de imágenes complejas 78
Figura 5. 22 Ejemplo de rastreo fallando entre dos fotogramas. 93

Lista de tablas

Tabla 3. 1 Interpretación de necesidades 24
Tabla 3. 2 Establecimiento de prioridad 25
Tabla 3. 3 Cuadro comparativo entre lenguajes de programación 35
Tabla 3. 4 Lista resumen de la búsqueda interna y externa de soluciones 37
Tabla 3. 5 Métricas definidas para cumplir con las diferentes necesidades encontradas 38
Tabla 3. 6 Concepto A 39
Tabla 3. 7 Concepto B 40
Tabla 3. 8 Concepto C 40
Tabla 3. 9 Concepto D 40
Tabla 3. 10 Concepto E 40
Tabla 3. 11 Concepto F 41
Tabla 3. 12 Resultados de experimentos del modelo comercial Yolov5-ADAS en el conjunto de datos "KITTI" [51] 42
Tabla 3. 13 Calificación y filtrado de conceptos 42
Tabla 3. 14 Selección de conceptos 43
Tabla 3. 15 Prueba de validación 1 ..... **44**
Tabla 3. 16 Prueba de validación 2 44
Tabla 3. 17 Prueba de validación 3 45
Tabla 3. 18 Prueba de validación 4 45
Tabla 3. 19 Prueba de validación 5 46
Tabla 3. 20 Resumen de las pruebas de validación 47

Tabla 4. 1 Hiperparámetros utilizados para el entrenamiento de la primera solución 49
Tabla 4. 2 Hiperparámetros utilizados para el entrenamiento de la segunda solución 49
Tabla 4. 3 Hiperparámetros utilizados para el entrenamiento de la tercera solución 50

Tabla 5. 1 Mejores combinaciones de parámetros para los entrenamientos de modelos de IA 64
Tabla 5. 2 Resumen de resultados del análisis de precisión durante el entrenamiento de los modelos de IA diseñados 71
Tabla 5. 3 Resumen de resultados del análisis de precisión en sets de datos de distintas dificultades. 79
Tabla 5. 4 Resultados del primer escenario para el prototipo 1, utilizando una entrada de vídeo local 81
Tabla 5. 5 Resultados del primer escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por UDP 81
Tabla 5. 6 Resultados del primer escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP 82
Tabla 5. 7 Resultados del primer escenario para el prototipo 2, utilizando una entrada de vídeo local 82
Tabla 5. 8 Resultados del primer escenario para el prototipo 2, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP 83

Tabla 5. 9 Resultados del primer escenario para el prototipo 3, utilizando una entrada de vídeo local 84
Tabla 5. 10 Resultados del primer escenario para el prototipo 3, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP 84
Tabla 5. 11 Resultados del segundo escenario para el prototipo 1, utilizando una entrada de vídeo local 85
Tabla 5. 12 Resultados del segundo escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por UDP 85
Tabla 5. 13 Resultados del segundo escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP 85
Tabla 5. 14 Resultados del segundo escenario para el prototipo 2, utilizando una entrada de vídeo local 86
Tabla 5. 15 Resultados del segundo escenario para el prototipo 2, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP 86
Tabla 5. 16 Resultados del segundo escenario para el prototipo 3, utilizando una entrada de vídeo local 87
Tabla 5. 17 Resultados del segundo escenario para el prototipo 3, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP 87
Tabla 5. 18 Resultados del tercer escenario para el prototipo 1, utilizando una entrada de vídeo local 88
Tabla 5. 19 Resultados del tercer escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por UDP 88
Tabla 5. 20 Resultados del tercer escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP 88
Tabla 5. 21 Resultados del tercer escenario para el prototipo 2, utilizando una entrada de vídeo local 89
Tabla 5. 22 Resultados del tercer escenario para el prototipo 2, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP 90
Tabla 5. 23 Resultados del tercer escenario para el prototipo 3, utilizando una entrada de vídeo local 90
Tabla 5. 24 Resultados del tercer escenario para el prototipo 3, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP 91
Tabla 5. 25 Tabla resumen de los resultados del experimento para medir tiempo de procesamiento 92
Tabla 5. 26 Tabla resumen de los resultados del experimento para medir la precisión del rastreo 93
Tabla 5. 27 Tabla resumen de los resultados del experimento para medir la precisión del rastreo 94
Tabla 5. 28 Tabla de presupuesto de costos de inversión 96
Tabla 5. 29 Beneficios generados gracias al desarrollo del proyecto 97

2.1 Lista de abreviaciones

Industria

NICR:	National Instruments Costa Rica
GM:	General Motors

Técnicos

ADAS:	Sistema Avanzado de Asistencia al Conductor
F-RCNN:	Red Neuronal Convolucional Basada en Regiones más Rápida
YOLO:	Algoritmo “Solo miras una vez”
OBD:	Diagnósticos a Bordo
CAN:	Red de Área de Controlador
ACC:	Control de Crucero Adaptativo
LDW:	Alerta de Salida de Carril
FCW:	Advertencia de Colisión Frontal
BSM:	Monitorización de Puntos Ciegos
AEB:	Freno Automático de Emergencia
SSD:	Detección de Disparo Único
UE	Metodología de Diseño Ulrich Eppinger
SD	Sistema Diseñado

Unidades

m:	Metros
m/s:	Metros sobre segundo
Rpm:	Revoluciones por minuto
ms	Milisegundos

1 INTRODUCCIÓN

1.1 Entorno del Proyecto

Por más de 40 años, NI ha desarrollado sistemas para pruebas automatizadas y para mediciones automatizadas los cuales han permitido que ingenieros de todo el mundo resuelvan todo tipo de problemas. El departamento de Servicios Profesionales de NI Costa Rica integra componentes de Hardware y Software para crear sistemas de pruebas de la mejor calidad para múltiples industrias. Muchas de estas soluciones requieren trabajar con sistemas bajo prueba altamente sofisticados y con altas restricciones de confidencialidad por parte del cliente. [1]

Junto con las áreas de enfoque que toma NI Costa Rica y el constante crecimiento y ambiente competitivo de la industria automotriz, las diferentes empresas fabricantes se ven en la obligación de realizar mejoras de manera progresiva con forme van desarrollando nuevos modelos de vehículos. Esto da origen a la necesidad de innovar con aplicaciones que permiten mejorar la experiencia del conductor y los pasajeros. Las unidades de control electrónicas han creado la posibilidad de generar muchas de estas mejoras, dentro de las que se encuentran, el sistema de control de suspensión, el control del catalizador y principalmente, Los Sistemas Avanzados de Asistencia en la Conducción también llamados ADAS por sus siglas en inglés. Estas asistencias tienen múltiples posibles aplicaciones como el revisar cuando existe un obstáculo en la cercanía del vehículo, cuando se aproxima un objeto a altas velocidades, entre otro tipo de información, que al ser analizada por los distintos algoritmos diseñados para este hardware es capaz de enviar señales de control a los distintos sistemas del automóvil con el fin de evitar accidentes. Son puestas a trabajar en Unidades de Control Electrónicas o ECU, las cuales constan de tarjetas embebidas que se encargan de cumplir distintas funciones importantes para el automóvil. [2]

Las Unidades de Control Electrónico (ECU) dan su origen en los años setenta, en esta época, las personas tenían la necesidad de mejorar el consumo de combustible debido a la crisis de petróleo que afectaba al planeta en ese momento, esto forzaba buscar maneras en las que los motores fueran capaces de funcionar de forma más efectiva y que lleguen a contaminar menos, estos motores antiguos utilizaban un dispositivo llamado distribuidor para controlar el ritmo en que se activaba la bujía y el carburador para controlar la mezcla de combustible utilizada. Este sistema demostró ser muy poco efectivo, por lo que con la llegada de los primeros microprocesadores proporcionó una tecnología capaz de realizar cálculos complejos a alta velocidad necesarios para controlar la

sincronización de la chispa y la mezcla de combustible, esta fue la primera ECU. A principios de la década de 1980, las ECU se convirtieron en un componente estándar en la mayoría de los vehículos. [3]

Debido a que estos sistemas tienen una influencia directa con el bienestar de los usuarios, es una prioridad el garantizar el correcto funcionamiento de este software, puesto que un falso positivo o negativo en el reconocimiento del sistema puede representar un accidente con consecuencias fatales. Cada ECU en el mercado es diferente tanto en hardware como en software, en la mayoría de los casos las empresas que las manufacturan mantienen una estricta confidencialidad sobre su funcionamiento, debido a esto, el equipo de NI debe trabajar directamente con los sistemas que el cliente provee para diseñar y crear los sistemas de pruebas. El acceso al hardware y software del cliente es restringido para todo tipo de usos que no sean estrictamente el proyecto asignado, por lo que NI tiene un uso limitado de estos para poder desarrollar sus propios sistemas. Como cada empresa tiene su propia manera de fabricar estas tarjetas embebidas ECU y generan sus propias aplicaciones, requieren que el personal que vaya a dedicarse a depurar las aplicaciones generadas aprenda a utilizar este software en campo. Para el caso de NI, se desea lograr capacitar a los nuevos empleados en cómo utilizar estas tarjetas, como desarrollar software y revisar su correcto funcionamiento sin la necesidad de depender del hardware de los clientes.

Por lo general, el hardware típico que se utiliza para estas pruebas no está disponible para demostraciones y capacitación internas. Por lo tanto, el tener prototipos de sistemas bajo prueba permite el correr y depurar aplicaciones reales al trabajar de forma remota o bien fuera de las instalaciones de los clientes. Dado el contexto ofrecido en los párrafos anteriores, para el equipo de Servicios Profesionales de NI Costa Rica es deseable contar con prototipos de sistemas bajo prueba como varios ECU de ADAS. Estos dispositivos se van a utilizar en entrenamientos de incorporación, para mostrar respuestas reales de aplicaciones de los clientes y permitir que el proceso de validación forme parte de las capacitaciones. El sistema diseñado se utilizará cuando sea necesario durante el desarrollo de proyectos, para la creación rápida de prototipos, pruebas de concepto, demostraciones, validaciones de sistemas y ejecución de aplicaciones de clientes con datos reales

1.2 Definición del problema

NICR es una compañía que vende servicios profesionales a empresas de tecnología, la mayoría de los clientes suelen ser tanto ingenieros como científicos, una de las áreas en que se desarrollan más avances para el beneficio de los clientes es en la validación de algoritmos implementados en tarjetas embebidas de autos inteligentes.

Los autos modernos pueden llegar a contener hasta 100 ECU distintas para cumplir con distintas tareas, los usos más comunes son los siguientes:

- Módulo de control de carrocería: regula el funcionamiento de secciones importantes de la carrocería como puertas, limpiaparabrisas, ventanas, luces, entre otros.
- Tren motriz: controla el funcionamiento del tren motriz del vehículo, existen varias variaciones dependiendo de si el sistema es de combustión, híbrido o eléctrico.
- Módulo de control del motor: opera las funciones de combustión en un motor.
- Unidad de control telemático: realiza un seguimiento del vehículo proporcionando indicaciones, cambios de ruta, navegación, entre otros.
- ADAS ECU: Controla todas las funciones de los sistemas avanzados de asistencia al conductor, como salida de carril, prevención de colisiones, y otras alertas que se enfocan en evitar accidentes. [4] Los sistemas ADAS tienen como misión el proveer seguridad y comodidad a los usuarios de los vehículos que poseen estos sistemas. El automóvil moderno es un complejo sistema de sistemas. Debido a la forma en que el sistema ADAS interactúa con la totalidad del vehículo por lo general, información detallada sobre su funcionamiento no está disponible para el público ni para la comunidad de investigación debido a temas de confidencialidad corporativa. [5]

Los empleados de NICR poseen múltiples proyectos en los que se necesita realizar pruebas extensivas con el fin de garantizar el funcionamiento correcto de las ECU que se contienen en los vehículos, principalmente las encargadas de controlar el sistema ADAS. Por ende, se utilizan sistemas de pruebas de "*Hardware in the loop*" (HIL), el cual es un poderoso sistema de simulación. Con la simulación HIL, el usuario puede simular las partes del sistema que plantean desafíos como disponibilidad, costo y seguridad.

Un sistema de pruebas HIL consta de tres componentes principales: un procesador en tiempo real, interfaces de Entradas/Salidas (E/S) y una interfaz de operador. El procesador en tiempo real es el núcleo del sistema de pruebas HIL. Proporciona ejecución determinística de la mayoría de los componentes del sistema de pruebas HIL como comunicación de E/S de hardware, generación de estímulos y ejecución de modelos. Un sistema en tiempo real generalmente es necesario para proporcionar una simulación precisa de las partes del sistema que no están presentes físicamente como parte de la prueba, para garantizar el correcto funcionamiento de este, se utilizan los dispositivos a pruebas (DUT), los cuales se configuran para llevar a cabo esta función de las pruebas HIL. Las interfaces de E/S son señales analógicas, digitales y de bus que interactúan con la unidad bajo 8 prueba, se pueden utilizar para generar estímulos, adquirir datos para registro, proporcionar interacciones entre sensores y actuadores y ejecución de modelos. Finalmente, la interfaz del operador se comunica con el procesador en tiempo real para proporcionar comandos de pruebas y visualización. A menudo, este componente también permite la administración de la configuración, automatización de la prueba, análisis y tareas de reportes. [6]

1.3 Justificación

En el departamento de Servicios Profesionales en NICR existe la necesidad de poder capacitar personal, realizar presentaciones, pruebas de conceptos, validaciones de sistemas sin la necesidad de depender de hardware que no es propio de la misma compañía. En muchas ocasiones por temas de confidencialidad no se permite que los empleados de NI presenten algoritmos prácticos que se han utilizado, puesto que las tarjetas embebidas o demás hardware que se utiliza pertenecen a los clientes.

Los Sistemas ADAS cada vez tienen mayor complejidad en su función, por lo que requieren de cada vez mayor poder computacional, es necesario para NI el poder capacitar a los nuevos integrantes con ejemplos reales de los sistemas con los que se suele trabajar, el tener la oportunidad de realizar capacitaciones realistas con versiones genéricas de los algoritmos que utilizan los clientes permite una mejor preparación en campo sin la necesidad de depender de agentes externos a NI, así como permitir crear prácticas de validación de algoritmos y presentaciones del funcionamiento de aplicaciones desarrolladas por miembros del equipo.

Tomando en cuenta lo expuesto anteriormente, se propone generar un dispositivo de prueba DUT capaz de simular el funcionamiento de un sistema ADAS, que a su vez pueda utilizar la función de frenado automático, con el fin de cubrir esta necesidad a la hora de capacitar personal, generar demostraciones y sesiones de prueba de algoritmos. Esto permitirá una mejor preparación para el equipo de trabajo ante las labores que se requieran realizar con las tarjetas embebidas de los 9 sistemas ADAS, así como la oportunidad de demostrar el funcionamiento de trabajos logrados anteriormente sin la necesidad de depender de hardware ajeno que en muchos casos no se encuentra disponible.

Diagrama Causa -Efecto (Ishikawa)

En la Figura 1. 1 se aprecia el diagrama causa – efecto desarrollado para definir el problema planteado.

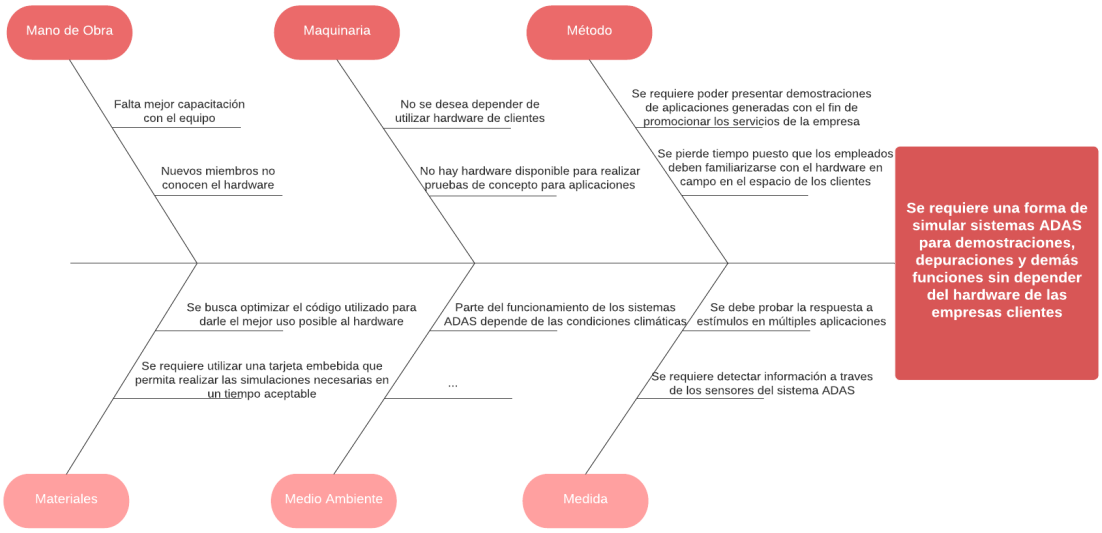


Figura 1. 1 Diagrama Causa - Efecto

1.4 Síntesis del problema

Inexistencia de hardware especializado para realizar depuraciones de algoritmos de control para Unidades de Control Electrónicas que se encargan del Sistema Avanzado de Ayuda de Conducción de Autos Autónomos.

1.5 Objetivo General

Evaluar algoritmos de detección de objetos para la capa de percepción en un sistema de frenado automático de emergencia, mediante el prototipado y validación en software.

1.6 Objetivos Específicos

- Investigar el estado del arte sobre los distintos métodos, en que se desarrolla la detección de objetos, para aplicaciones que necesiten reacción en tiempo real.

  Entregable: Estudio realizado sobre las múltiples arquitecturas y estrategias de desarrollo de aplicaciones de detección de objetos en tiempo real.

- Diseñar un sistema de visión artificial para un sistema ADAS que posea el funcionamiento general de un sistema de reconocimiento de elementos en ruta

  Entregables: Guía de diseño para el programa de detección de objetos.

  Indicador: El prototipo diseñado debe ser capaz de detectar objetos a través de video a 20 fotogramas por segundo

- Diseñar una aplicación en software para los 3 diseños de algoritmos que se encuentren mejor adaptados para la detección de objetos

  Entregable: Guía de diseño de los prototipos de software.

  Indicador: Los algoritmos deben cumplir con una precisión al detectar objetos de al menos 90%

- Integrar el sistema de visión junto con el sistema de control del simulador del sistema ADAS

  Entregable: Guía de diseño para la comunicación con el algoritmo de control de toma de decisiones

  Indicador: El sistema debe tener una tasa de error en la comunicación entre 2 y 5%

- Verificar el funcionamiento del sistema de visión artificial por medio de pruebas de concepto, así como enviar las señales de control de forma correcta.

  Entregable: Documento con los resultados obtenidos de la verificación del sistema de visión.

  Indicador: Sistema completo funcional el cual tenga una capacidad de clasificación de al menos 90%.

2 MARCO TEÓRICO

Dado el hecho de que este proyecto consiste en la unión de varias disciplinas o bien áreas de conocimiento, este capítulo pretende reforzar las bases de cada una de estas áreas y que así exista tanto una mayor comprensión del proyecto como aquellos aspectos relacionados a él. De este modo se separó el conocimiento teórico en secciones para agrupar aquellos conceptos más cercanos entre sí, resultando en 4 secciones: Conducción Automatizada, generalidades de proyectos.

2.1 Conducción Automatizada

La industria automotriz se ha reinventado múltiples veces a lo largo de las últimas décadas. Como se trata de un mercado sumamente competitivo, se requieren de avances tecnológicos constantes que permitan a las múltiples empresas encargadas de la manufactura de vehículos mantenerse al nivel de sus adversarios en el mercado. La Conducción Automatizada, (AD por sus siglas en inglés) se trata de uno de los desarrollos tecnológicos más importantes en la última década. Esta permite ventajas importantes para el usuario, desde evitar accidentes hasta facilitar considerablemente la conducción, permitiendo así que más personas sean capaces de manejar un vehículo sin importar la experiencia que posean. No obstante, al tratarse de un sistema que puede resultar peligroso tanto para el usuario como su entorno, se requiere de técnicas de validación lo más eficaces posibles como se detalla en [8], para lograr llevar a cabo la Conducción Automatizada se necesitan múltiples Unidades de Control Electrónicas, cada una se encarga de realizar ciertas funciones en el vehículo las cuales al unirse conforman el Sistema Avanzado de Ayuda a la Conducción (ADAS).

2.1.1 Unidades de Control Electrónicas (ECU)

Las Unidades de Control Electrónicas son un componente crítico para la industria automotriz, se trata de tarjetas embebidas o microcontroladores que se encargan de realizar múltiples funciones, como mantener el rendimiento del motor, controlar las emisiones, sistemas de seguridad entre otros. Las ECU dan su origen en la década de los años sesenta, en donde se utilizaron controladores para

aplicaciones aeroespaciales, principalmente se utilizaban las funciones del control del motor y los sistemas de navegación.

No sería hasta la década de los setenta donde se utilizarían controladores para aplicaciones de uso general, inicialmente se utilizó para el control de las emisiones de los motores, puesto que los modelos de la época por lo general no eran capaces de cumplir con las regulaciones necesarias para el uso cotidiano de combustible fósil [9]. En los años ochenta se vería un avance significativo con la evolución de la industria digital, en dicha década se generarían los microprocesadores digitales y los sistemas de control basados en software. Esto permitió el origen de sistemas de control de emisiones y rendimiento de motor mucho más precisos y eficientes. Asimismo, se dio origen a los diagnósticos a bordo (OBD por sus siglas en inglés) los cuales consisten en un sistema que tiene la función de controlar y monitorear las ECU y sus funciones. [10]

En la década de los noventa, los sistemas de control electrónicos tomaron fuerza en el mercado, ya no se limitan únicamente a controlar el funcionamiento del motor y el control de emisiones, sino que se agregan nuevas funcionalidades como sistemas de bolsas de aire en caso de accidentes y frenos antibloqueo. En esta época también surge en Alemania, el protocolo de comunicación conocido como Red de Área de Controlador (CAN por sus siglas en inglés), creado por Robert Bosch GmbH, una compañía de ingeniería y electrónica alemana. Este protocolo permite una comunicación eficaz entre las múltiples ECU presentes en el vehículo, ya que se puede enviar información como velocidad, temperatura y niveles de combustible, permitiendo así el desarrollo de sistemas más sofisticados. Las redes CAN resultaron ser tan buen diseño, principalmente por su alta confiabilidad, bajo costo de manufactura, entre otras ventajas, que se siguen utilizando hasta la fecha en vehículos en todo el mundo. [11]

En el siglo 21, la tecnología de los ECU ha evolucionado y es más avanzada que nunca, se pueden aplicar algoritmos complejos e implementar aplicaciones que utilicen inteligencia artificial para todo tipo de labores, como lo son seguridad, optimización de uso de combustible, entre otro, de ahí se derivan los sistemas avanzados de ayuda a la conducción (ADAS) en los cuales se enfoca el diseño y desarrollo del presente proyecto.[12]

2.1.2 Sistemas ADAS

Los sistemas avanzados de asistencia al conductor son de los avances más recientes en la industria automotriz, son capaces de cumplir múltiples labores las cuales se van a describir a continuación con base en los estándares establecidos por [13].

2.1.2.1 Control de crucero adaptativo (ACC):

Se trata de una función que mantiene al vehículo a cierto valor de velocidad y lo aumenta o reduce dependiendo de la distancia que detecte que existe entre el propio vehículo y el que tenga en frente. La prioridad para este sistema es garantizar una distancia segura entre el vehículo anfitrión y el que tenga en frente. También cuenta con dos modos de funcionamiento, el ACC convencional y el ACC *Stop and Go*, ambos tienen una meta similar, no obstante, el ACC *Stop and Go* es más sofisticado en el sentido que si el vehículo que se encuentra en frente se detiene, el anfitrión se detendrá por completo, también tiene funciones como reiniciarse de forma automática y continuar con el recorrido de la misma forma que se realizaba antes de que el auto de enfrente se detuviera una vez la carretera se encuentre libre. [14]

2.1.2.2 Alerta de salida de carril (LDW):

Este sistema alerta al conductor si el vehículo se está desviando de su carril, utilizando cámaras u otros sensores para monitorizar las marcas del carril. Este sistema da origen a un estudio realizado en [15] en el cual se denota que un gran porcentaje de accidentes de tránsito en Estados Unidos se deben a conducción fuera de carril, por lo que se diseñó esta función con la finalidad de reducir ese tipo de accidentes, el sistema envía una señal tanto visual como auditiva al conductor cuando el vehículo está detectando que se salió de la vía en que debería conducir. Versiones más sofisticadas de LDW son capaces de tomar control del sistema de dirección del vehículo y corregir el error de navegación detectado.

2.1.2.3 Advertencia de colisión frontal (FCW):

Este sistema utiliza sensores para detectar si el vehículo se acerca a otro vehículo u obstáculo demasiado rápido, y alerta al conductor para que tome medidas para evitar una colisión. El sistema tiene la meta de enviar una señal visual y de audio al conductor con 2.7 segundos de anticipación ante una colisión inminente, lo cual según estándares en la industria equivale a suficiente tiempo para que la mayoría de las personas sean capaces de reaccionar, todo el sistema se basa en el cálculo que es capaz de realizar para el tiempo de colisión.

2.1.2.4 Monitorización de puntos ciegos (BSM):

Este sistema utiliza sensores para detectar vehículos y/o cualquier tipo de obstáculo en los puntos ciegos del conductor, y alerta al mismo si es inseguro cambiar de carril. Este sistema por lo general empieza su funcionamiento al sobrepasar cierto rango de velocidad (usualmente los 10 km/h) y de forma similar a los dos anteriores, le advierte al conductor de cualquier riesgo por medio de señales visuales vía luces en los espejos y, cuando se detecta a mayor proximidad, con un sonido de alerta.

2.1.2.5 Cámara de visión trasera:

Este sistema proporciona una vista del área detrás del vehículo para ayudar al conductor a retroceder de manera segura. La mayoría de estos sistemas añaden también un sensor de posición que enciende una señal de sonido cuando se detecta que el automóvil va a colisionar cuando se encuentra moviéndose en reversa.

2.1.2.6 Frenado de emergencia automático (AEB):

Este sistema puede aplicar automáticamente los frenos si detecta una colisión inminente y el conductor no responde a tiempo. Se trata de una evolución de las demás funciones descritas anteriormente, solo que se agrega el factor que ahora existe autonomía sobre el sistema de frenado. Para tener un funcionamiento óptimo, se subdivide en dos capas, una de percepción la cual se encarga de utilizar los distintos sensores para detectar obstáculos a su alrededor. Y una capa de control o de toma de decisiones, la cual a partir de la información que brinde la capa de percepción. [16]

2.1.3 Capa de percepción de sistema AEB

En el diseño de la capa de percepción de un sistema AEB se deben tomar en cuenta múltiples conceptos importantes para lograr una detección de obstáculos eficaz los cuales serán detallados a continuación:

2.1.3.1 Sensores:

Para lograr cumplir cualquier funcionalidad que se puede relacionar con la conducción autónoma, se requiere obtener información del entorno, los sistemas ADAS utilizan múltiples tipos de sensores para poder reconocer el ambiente en que el vehículo se encuentra. Los sistemas AEB utilizan principalmente tres tipos de sensores: Cámaras, radares y sensores ultrasónicos.

Los radares y sensores ultrasónicos se utilizan para funciones similares, ambos utilizan señales que rebotan en los objetos u obstáculos y cuanto estas señales vuelven al sensor, se aproxima la

distancia que se encuentra entre el sensor y el objeto detectado, no obstante, el principal problema de ambos sensores se encuentra en que no se puede determinar qué clase de obstáculo se detecta, por lo que pueden generarse falsos verdaderos en la información adquirida, generando problemas en el funcionamiento general del sistema.[17]

Debido a los problemas descritos anteriormente, el enfoque moderno le da mayor importancia al uso de cámaras de distintos tipos según se describe en [18] (En modelos de lujo se utilizan varias alternativas de forma simultánea):

- Cámaras estéreo: Las cuales utilizan dos lentes para generar una imagen en tres dimensiones que permite aproximar de forma más precisa las distancias entre objetos en las imágenes obtenidas.
- Monoculares: Estas generan una imagen en dos dimensiones, a pesar de ser menos precisas, representan una importante diferencia en cuanto a precio, además de que brindan información suficiente para el funcionamiento de los sistemas AEB.
- Infrarrojas: Estas cámaras tienen como principal función el permitir detectar obstáculos cuando los demás tipos de cámaras no son capaces, principalmente en ambientes con poca luz.
- Térmicas: La tecnología de imágenes térmicas permite utilizar las señales de calor para detectar obstáculos sin importar condiciones climáticas o poca iluminación.

Detección de objetos: El uso de las cámaras como sensores implica que debe existir algún mecanismo para lograr interpretar la información que brindan las imágenes dadas por las cámaras. El método por excelencia en interpretación de imágenes es utilizar la visión por computadora. Las ramas de la inteligencia artificial y visión por computadora son soluciones a problemas cotidianos que al utilizar estas herramientas se logra automatizar procesos que de otra forma tomarían cantidades extremas de tiempo. El uso de un algoritmo de detección de objetos a través de visión por computadora es el estándar en las aplicaciones actuales en autos autónomos en países como Japón, Alemania, Francia y los Estados Unidos. Se han logrado realizar detecciones en tiempo real a velocidades de hasta 130 km/h lo que les permite a los automóviles equipados con estos sistemas realizar lo más cercano hasta la fecha a una conducción completamente autónoma, según dicta [19].

2.1.3.2 Toma de decisiones:

Para que el sistema AEB sea funcional no basta con detectar un obstáculo en la vía, el vehículo debe ser capaz de reaccionar acorde a lo que se llega a detectar, para ello se diseñan sistemas de control también conocidos como algoritmos de toma de decisiones. Se debe tener un sistema a lazo

cerrado en el cual la entrada sea la información que detectan los múltiples sensores, el algoritmo procede a realizar las señales de control adecuadas a la situación, por ejemplo, disminuir la velocidad de forma porcentual según se acerca un obstáculo y de esa forma mientras se mantenga el automóvil en movimiento se debe seguir con ese ciclo de funcionamiento. Para comprobar el correcto funcionamiento de este algoritmo se suele utilizar una técnica llamada *Hardware in the Loop* o HIL por sus siglas en inglés. La cual consiste en poner a prueba el ciclo de trabajo en un ambiente controlado y corroborar constantemente que se cumple con las labores que se necesita que cumpla el algoritmo, una forma de realizar ese tipo de validaciones es conectando el computador principal del vehículo a un computador externo que corre una simulación del circuito simulado, entonces se pone a prueba los distintos sistemas del automóvil que deben reaccionar a los cambios del entorno simulado sin la necesidad de llevar un modelo sin terminar a las calles [20].

La Figura 2. 1 presenta en forma gráfica el ciclo de trabajo del sistema por diseñar en forma de bloques y la secuencia que este seguirá.

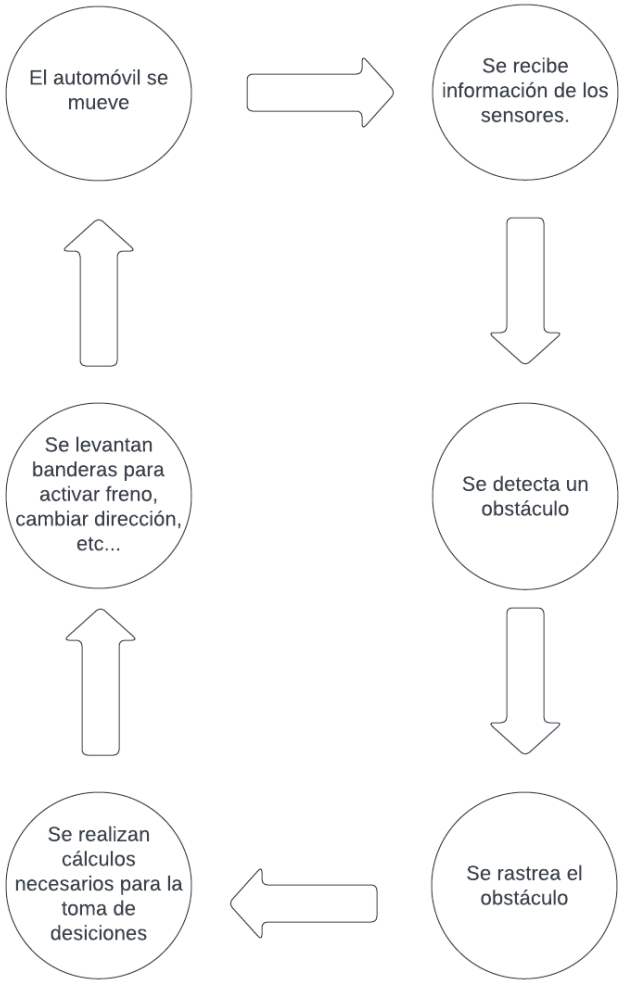


Figura 2. 1 Ciclo de trabajo del algoritmo de toma de decisiones

2.1.4 Inteligencia artificial

La inteligencia artificial es una rama de las ciencias de la computación que se encarga de crear agentes inteligentes, estos se tratan de algoritmos que pueden razonar, aprender y actuar de forma completamente autónoma, se trata de uno de los ámbitos más importantes en investigación alrededor del mundo en los últimos años. La forma más sencilla de un algoritmo de inteligencia artificial es llamada Red Neuronal, esta consiste en, como lo indica su nombre, simular el funcionamiento de las neuronas humanas, buscando que esta sea capaz de aprender información y enviar una respuesta ante un estímulo o en otras palabras brindar una salida correspondiente a una entrada específica.

Los algoritmos de inteligencia artificial se utilizan para múltiples tareas, dentro de las cuales se encuentran:

- Clasificación: Clasifica información en categorías, por ejemplo, se pueden clasificar imágenes de animales según especie o bien clasificar textos según el idioma que utilicen.
- Predicción: A partir de lo aprendido en el pasado y la tendencia de los datos en este, se encarga de predecir qué va a pasar en el futuro, por ejemplo, se puede llegar a predecir el clima en una zona.
- Procesamiento de lenguaje: Se puede utilizar inteligencia artificial para detectar el idioma en que se encuentra un texto y traducirlo a otro, también puede generar su propio texto y hasta responder preguntas.
- Machine Learning: La inteligencia artificial puede utilizarse para mejorar procesos y mejorar su funcionamiento, por ejemplo, se puede entrenar una inteligencia artificial para que sea capaz de clasificar imágenes o predecir el comportamiento de alguna tendencia como el tipo de cambio del dólar.

2.1.4.1 Entrenamiento de una inteligencia artificial:

Para lograr desarrollar un algoritmo de inteligencia artificial, se debe seguir una serie de pasos, el primero es el obtener los suficientes datos, a esto se le llama un conjunto de datos o *dataset*, en este se almacenará todo tipo de información que se desee que el algoritmo sea capaz de entender, los conjuntos de datos se subdividen en tres secciones o bien subconjuntos, entrenamiento, validación y prueba. El primero se encarga de enseñarle al algoritmo la mayor cantidad posible de información para que este sea capaz de realizar sus funciones, el subconjunto de validación se utiliza para comprobar si el entrenamiento fue efectivo, a partir de este es que se generan las gráficas de pérdidas y de precisión de los modelos de IA. Las pérdidas definen qué tan alejadas se encuentran las predicciones del modelo con respecto a los valores correctos, mientras que la precisión es una medición de cuántas predicciones son correctas. Estas gráficas (que se ejemplifican en la Figura 2. **2**) permiten evaluar el comportamiento del modelo una vez que se utilizan con el subconjunto de datos de prueba, gracias al cual se puede corroborar el funcionamiento del modelo de IA, se espera que en la gráfica de precisión se vea una forma similar a una función exponencial cuyo final converja a un valor de forma asintótica, asimismo, se espera un comportamiento similar pero de forma decreciente para las gráficas de pérdidas, de igual manera es requerido que se note una

convergencia hacia un valor para determinar si el entrenamiento se completó de forma exitosa o si se requieren más iteraciones para tener resultados óptimos.[21]

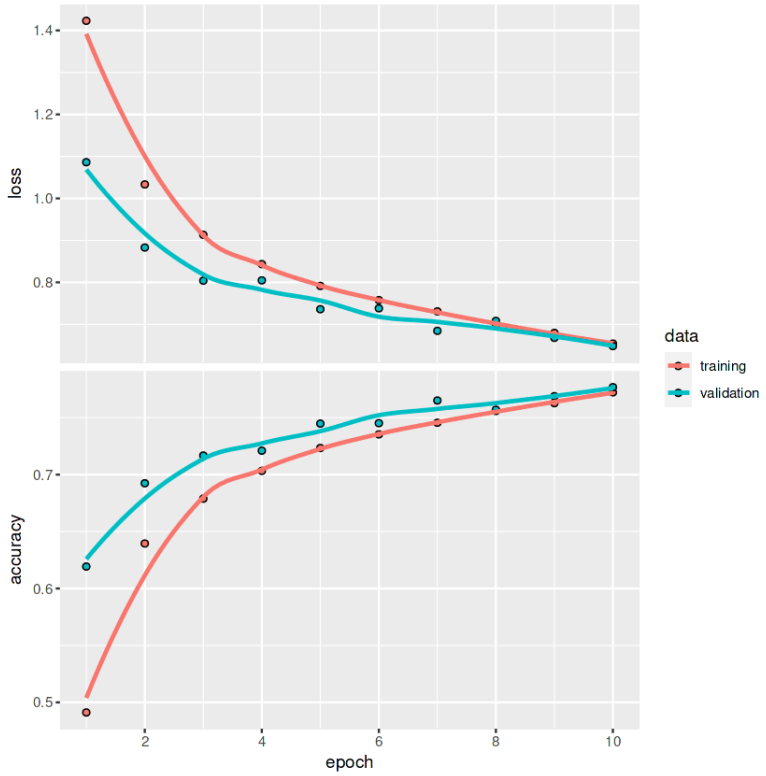


Figura 2. 2 Ejemplo de gráficas de pérdidas y de precisión tomada de [21]

Una vez se tienen los datos necesarios para llevar a cabo el entrenamiento, se deben de definir los diferentes hiperparámetros que van a definir el funcionamiento del modelo de IA, un hiperparámetro es un elemento que define el funcionamiento del modelo por generar, que no se aprende directamente del conjunto de datos durante el proceso de entrenamiento de un modelo y se tiene completo control sobre este. Estos hiperparámetros influyen en el comportamiento y rendimiento del modelo, pero deben ser establecidos de antemano por el desarrollador antes de comenzar el entrenamiento. Los más relevantes de estos son:

- Tasa de aprendizaje: Define la velocidad con la que el modelo aprende, un valor muy elevado tiende a generar modelos de forma más rápida, pero a su vez es muy probable que no aprendan lo suficiente para aplicaciones reales.
- Número de épocas/iteraciones: La cantidad de épocas define cuántas iteraciones se realizarán de entrenamiento, en cada iteración, se alimenta la información del conjunto de datos completo con el fin de actualizar la función de descenso de gradiente según los

resultados obtenidos en cada una de estas iteraciones. Es importante definir un límite de iteraciones prudente para evitar casos de sobre entrenamientos los cuales reducen significativamente la efectividad de los modelos de clasificación.

- Tamaño del lote de entrenamiento: Este hiperparámetro define cuánta información se procesa para entrenar al modelo a la vez, por ejemplo, si se está entrenando para detectar imágenes, durante el entrenamiento se definiría cuántas imágenes del conjunto de datos de entrenamiento se utilizarían a la vez, un valor elevado de este hiperparámetro permite un entrenamiento mucho más rápido, pero consume más poder computacional y se arriesga a tener menos precisión en el modelo final.

- Función de activación: es una función matemática que se aplica a la salida de una neurona o a la suma ponderada de las entradas de la neurona. Su propósito principal es introducir no linealidad en el modelo, lo que permite que la red neuronal aprenda y modele relaciones complejas en los datos. Existen múltiples funciones de activación como la sigmoide, ReLu, Softmax, entre otras.

- Selección de características: Este es el proceso de elegir qué características usar en el modelo. Las características que no son relevantes para la variable de destino pueden dañar la precisión del modelo. Normalmente para solventar este problema se suele utilizar un modelo previamente entrenado en el formato deseado como base para evitar tener características innecesarias. [21]

Es importante recalcar la diferencia del concepto de hiperparámetros con parámetros en el contexto de inteligencia artificial y visión por computadora, los hiperparámetros se tratan de elementos de los cuales el desarrollador tiene total control de sus valores, mientras que los parámetros son valores fijos que definen la arquitectura del funcionamiento del o los modelos de inteligencia artificial que se vayan a desarrollar.

Además de las gráficas de pérdidas y precisión, se puede validar si un entrenamiento de IA fue exitoso con la ayuda de una herramienta conocida como matriz de confusión, en esta se grafican las diferentes categorías de datos que la IA debe ser capaz de clasificar y en cuál de estas categorías el modelo está clasificando los datos, por lo que se ven cuadros en los que un eje representa la categoría real y en el otro la categoría en que la IA lo clasificó, tal y como se denota en la Figura 2. 3. Con la ayuda de esta herramienta, se puede corroborar la calidad con la que un modelo de clasificación que trabaje con varias clases, clasifique cada clase de datos de forma correcta.

Figura 2. 3 Ejemplo de matriz de confusión [21]

2.1.5 Visión Artificial

La visión artificial o visión por computadora, se refiere a la capacidad de máquinas como computadoras o robots para interpretar y analizar datos a partir de imágenes. Para lograr una aplicación de visión por computadora es necesario el uso de varios tipos de *hardware* y *software*.

2.1.5.1 Detección por medio de visión artificial:

En el contexto de la visión artificial, detección es el proceso de identificar un objeto y su posición relativa a la imagen o video que se esté procesando. Se trata de una etapa fundamental para cualquier tipo de aplicación que utilice visión artificial, como reconocimiento, seguimiento y clasificación de objetos, entre otras.

Para lograr detección de objetos, se requiere entrenar un algoritmo para que sea capaz de identificar objetos específicos en las imágenes que vaya a recibir de entrada. Existen varios tipos de algoritmos que pueden lograr esta función, por ejemplo, se puede inspeccionar por patrones o características generales del objeto como lo son el color, textura, forma, orientación, entre otras. La práctica más común para verificar el correcto funcionamiento de la detección es generar un cuadro delimitador el cual encierre la sección de la imagen donde el algoritmo cree que se encuentra el objeto. Aplicaciones más complejas suelen optar por utilizar algoritmos de inteligencia artificial para agilizar el proceso de entrenamiento y detección, obteniendo así mejores resultados en general. [21]

2.1.5.2 Seguimiento/Rastreo (Tracking):

El rastreo en el contexto de la visión por computadora se refiere al proceso de seguir el movimiento de los objetos detectados por la aplicación de detección de objetos, consiste en analizar múltiples fotogramas consecutivos e identificar características o patrones que pertenecen al mismo objeto o región a través de cuadros, y luego calcular el movimiento de ese objeto o región con el tiempo.

Existen diferentes enfoques para el seguimiento, incluyendo métodos basados en modelos, basados en características y basados en flujo óptico. El seguimiento basado en modelos utiliza un modelo previo del objeto a seguir, mientras que el seguimiento basado en características utiliza características distintivas como esquinas o bordes para seguir el objeto. Los métodos basados en flujo óptico estiman el movimiento de los objetos analizando los cambios en la intensidad de píxeles con el tiempo.

El seguimiento es una tarea fundamental en la visión por computadora y tiene muchas aplicaciones prácticas, como la detección de objetos, la vigilancia de videos, la robótica y los vehículos autónomos. En general para aplicaciones relacionadas con la conducción autónoma, se requiere una alta seguridad a la hora de rastrear objetos, puesto que gracias a esto es que es posible calcular la velocidad con la que se aproximan los objetos al vehículo y de esta forma se pueden tomar decisiones adecuadas a la situación. [22]

2.1.6 Simulación de escenarios

Los sistemas ADAS deben de cumplir con múltiples parámetros de funcionamiento antes de poder ser aprobados para salir al mercado, se debe de garantizar un funcionamiento adecuado, para lo cual se realizan pruebas de concepto. Existen múltiples formas de probar las funciones de los sistemas ADAS, pero las más populares en la industria son *Software in té Loop* y *Hardware in the Loop*. Ambas consisten en estrategias para la validación de productos, principalmente algoritmos que se utilizarían en autos inteligentes.

Para poder llevar a cabo una prueba de concepto de cualquiera de los dos tipos mencionados, se requiere de generar un escenario adecuado para probar los posibles casos a los que se desea garantizar un funcionamiento óptimo. Para esto existen múltiples *Software* especializados en la generación de escenarios lo más realistas posibles, seguidamente se utiliza este escenario simulado como entrada para los sensores (en este caso las cámaras frontales del automóvil) y que el auto

reaccione a los obstáculos presentes en la simulación en vez de obstáculos reales. Esto ahorra tiempo para implementar escenarios reales además de eliminar variables que no se desean tomar en cuenta para alguna iteración del diseño.

Dentro de los *software*s especializados existen LGSVL Simulator, Unity 3D, ANVEL (Laboratorio de Entornos Virtuales para Navegación Autónoma) y NI Monodrive. Estos programas son capaces de recrear espacios reales como carreteras, autopistas, parques, entre otros, en donde se pueden simular ambientes en donde se quiere poner a prueba el comportamiento de los distintos ECU existentes en un vehículo autónomo ante múltiples estímulos, sin la necesidad de poner en riesgo a ninguna persona o ningún equipo en caso de fallar.

La validación de tipo *Hardware in the Loop* (HIL) consiste en alimentar los sensores con entradas diseñadas previamente y que los actuadores físicos reaccionen ante ese escenario virtual y dependiendo de la respuesta, se repite el experimento para poder determinar aciertos y posibles mejoras en los diseños de sistemas que se desean implementar. Este tipo de validación se suele utilizar para la implementación de nuevas aplicaciones de *software* en ECU completamente funcionales y se desee optimizar el funcionamiento de este. [23]

La validación *Software in the Loop* (SIL) es una metodología de prueba basada en simulación utilizada en el desarrollo de software para verificar y validar la funcionalidad de este. En SIL, el software que se está probando se integra con un entorno simulado que emula el comportamiento y las entradas del sistema, lo que permite una prueba realista sin requerir que los componentes del hardware del sistema estén físicamente presentes.

Se utiliza comúnmente en el desarrollo de sistemas complejos, como vehículos autónomos o aeronaves, donde la prueba con hardware del mundo real puede ser costosa, llevar mucho tiempo y ser potencialmente peligrosa. El proceso de validación de SIL generalmente involucra los siguientes pasos: desarrollar un entorno de simulación, integrar el software a ser probado en la simulación, ejecutar pruebas con entradas y condiciones simuladas, analizar los resultados y realizar las modificaciones necesarias al software basándose en los resultados de la prueba. [24]

2.1.7 Sistemas de Control para aplicaciones ADAS

Los sistemas ADAS requieren de un control robusto para poder funcionar de forma óptima, para que este sistema de control sea eficiente, debe haber una transmisión de información de manera

efectiva, ya que este se encarga de recopilar información de los diferentes sensores, interpretarla y tomar decisiones adecuadas a la información recibida. Los algoritmos de control utilizados en ADAS varían según el sistema específico. Por ejemplo, los sistemas de advertencia de salida de carril utilizan algoritmos para detectar cuando un vehículo se está desviando de su carril y proporcionar una advertencia al conductor. Los sistemas de control de crucero adaptativo utilizan algoritmos para mantener una distancia de seguimiento segura del vehículo que está delante. Los sistemas de frenado de emergencia utilizan algoritmos para detectar una colisión inminente y aplicar los frenos automáticamente y de esta forma evitar posibles accidentes.

Para garantizar la confiabilidad y la seguridad de los sistemas de control ADAS, a menudo se diseñan con sensores y unidades de control redundantes, así como mecanismos a prueba de fallas que pueden tomar el control en caso de falla del sistema. Además, los sistemas de control ADAS deben cumplir con estrictas normas de seguridad y regulaciones para garantizar que cumplan con los más altos niveles de seguridad y rendimiento. Pese a la redundancia aplicada, es de vital importancia la velocidad con la que el sistema de control se comunique con los diferentes subsistemas del vehículo [25]

2.1.8 Normativa

En el ámbito de la conducción autónoma, se deben de seguir múltiples normativas para garantizar la seguridad de los usuarios de este tipo de vehículos, dentro de estas normas se encuentra la ISO 26262 la cual consta de una norma internacional de seguridad, esta brinda un marco para el desarrollo y prueba de sistemas de seguridad, incluyendo los que competen a los sistemas ADAS. La norma establece niveles de integridad de seguridad que se denominan con letras desde la A hasta la D. El nivel necesario que se debe cumplir para cada sistema depende de qué tan peligroso puede llegar a ser para la vida humana un posible malfuncionamiento. Por ejemplo, un nivel D (el más elevado) se requiere para funciones que controlen la velocidad y dirección del vehículo.

Según ISO 26262, todo tipo de aplicación que tome decisiones a partir de información obtenida por los sensores del sistema ADAS, debe ser suficientemente rápida para no comprometer al usuario a ningún tipo de peligro, por ello la norma establece estándares para cada sensor común en los autos modernos, por ejemplo, las cámaras utilizadas comúnmente trabajan a 20 fotogramas por segundo, por ende, toda aplicación desarrollada para obtener información de estas cámaras debe ser capaz de procesar la información a una velocidad suficientemente rápida para trabajar recibiendo información a 20 fotogramas por segundo y procesarla lo suficientemente rápido para tomar una decisión antes

de que se reciba el siguiente cuadro, el equivalente a 20 fotogramas por segundo son 50 milisegundos, por ende, todo algoritmo diseñado para un sistema ADAS que requiera información de parte de cámaras debe operar de forma ideal en menos de 50 milisegundos para lograr tener un funcionamiento ideal. [26]

La normativa SAE J3016 es un estándar desarrollado por la Sociedad de Ingenieros Automotrices |(SAE), la cual define y caracteriza niveles de automatización implementada para automóviles comerciales, existen seis niveles de automatización según este estándar, donde el nivel 0 significa que no posee ningún tipo de automatización y el nivel 5 se refiere a automóviles completamente autónomos.

El nivel 1 Se refiere a autos que poseen asistencias para la conducción, principalmente para el control de aceleración o dirección, pero nunca simultáneas. El nivel 2 se refiere a automatización parcial, en esta, el vehículo es equipado con sistemas que trabajan con la aceleración y la dirección del auto de forma simultánea, pero el conductor sigue siendo responsable de monitorear el entorno de conducción y tomar el control en caso de ser necesario. El nivel 3 se trata de una automatización condicional, en esta el vehículo es capaz de de realizar algunas tareas de conducción de forma independiente, pero el conductor debe estar preparado para tomar en control cuando sea necesario. El nivel 4 se refiere a una alta automatización en este nivel, el automóvil es capaz de realizar todas las labores de conducción de forma independiente, pero debe haber un conductor preparado para tomar en control en caso de ser necesario. Finalmente, el nivel 5 es para una automatización completa en la cual no se requiere ningún tipo de intervención.[27]

El Programa Europeo de Evaluación de Automóviles Modernos (EURO NCAP) se trata de una organización independiente que evalúa el desempeño de los autos modernos en Europa. Fue fundada en 1997 y se ha convertido en una fuente confiable e importante a la hora de mejorar los estándares de seguridad en los automóviles a nivel global. EURO NCAP realiza pruebas de concepto de tipo destructivas para evaluar el comportamiento de los distintos vehículos cuando se ven sometidos a accidentes, asimismo también realizan pruebas de sistemas como frenado automático, límites de velocidad y sistemas de precaución en general, lo cual ha apoyado al desarrollo de los sistemas ADAS. Las clasificaciones de seguridad dadas por EURO NCAP son utilizadas por empresas manufactureras, consumidores y autoridades reguladoras en todo el mundo. Las calificaciones han ayudado a crear conciencia sobre la seguridad de los automóviles y han promovido que los fabricantes mejoren el rendimiento de seguridad de sus vehículos. También han influido en el desarrollo e implementación de nuevas normas y estándares de seguridad.

La normativa EURO NCAP define que aplicaciones de detecciones de objeto deben de cumplir con al menos un 70% para aplicaciones que únicamente detectan y avisan al conductor, pero para casos en los que el algoritmo toma control de los demás sistemas del automóvil, se desean valores de precisión sobre el objeto deseado mayores al 90% cuando el obstáculo se encuentra en menos de 5 metros de distancia del automóvil que utiliza el sistema.[28]

3 METODOLOGÍA

3.1 Planeación

La metodología para seguir es la conocida como “Ulrich Eppinger” (UE), en esta se sigue un proceso de diseño definido por múltiples etapas: Identificar las necesidades del cliente, Establecer especificaciones objetivo, Generar conceptos de producto, Seleccionar conceptos de producto, Establecer las especificaciones finales y Planear el desarrollo descendente.

La primera etapa es el análisis y la determinación del problema, se utilizaron varios insumos principalmente por parte de declaraciones del cliente, así como regulaciones existentes en la industria involucrada (En este caso la industria automotriz). A partir de esta etapa se concluyó en la síntesis del problema descrita en la sección 1.4 de este documento. Una vez se definió claramente el problema por solucionar, se puede proceder con las demás etapas de la metodología establecida en [29]

3.1.1 Identificar las necesidades del cliente

La segunda etapa consiste en determinar las necesidades que el Sistema Diseñado (SD) debe cumplir para la generación de conceptos de solución del problema. Las necesidades obtenidas en el proceso de interpretación de las necesidades de los clientes del proyecto se ven reflejadas en la **Tabla 3. 1**. Una vez se interpretaron dichas necesidades, se requiere asignar distintos niveles de prioridad los cuales se definen según respuestas del cliente, estas se denotan en la **Tabla 3. 2**.

Tabla 3. 1 Interpretación de necesidades

Tema	Necesidades interpretadas
Percepción	• El SD debe detectar y rastrear objetos en tiempo real. • El SD debe cumplir un tiempo de procesado de imágenes menor que 50 ms. • El SD debe poder clasificar objetos en 3 clases distintas.
Prototipado	• El SD utiliza poco poder computacional para funcionar. • El SD es fácil de implementar en hardware. • El SD demuestra capacidad de rastrear los objetos que ya detectó anteriormente. • El SD requiere de poca memoria RAM.
Compatibilidad / Reutilización	• Las salidas del SD deben estar estandarizadas para poder utilizarse en futuras aplicaciones. • El SD debe tener su debida documentación y guía de uso para futuras implementaciones.
Calidad	• El SD debe tener un promedio de precisión de al menos 80 por ciento • El SD debe cumplir con estándares de código eficiente.

Tabla 3. 2 Establecimiento de prioridad

Nivel. Descripción	Necesidades
5. La función es de importancia crítica. No consideraría un producto sin ella	1. El SD debe detectar y rastrear objetos en tiempo real. 2. El SD debe cumplir un tiempo de procesado de imágenes menor que 50 ms. 3. El SD debe poder clasificar objetos en 3 clases distintas. 4. El SD demuestra capacidad de rastrear los objetos que ya detectó anteriormente. 5. Las salidas del SD deben estar estandarizadas para poder utilizarse en futuras aplicaciones.
4. La función es altamente deseable, pero consideraría un producto sin ella	6. El SD utiliza poco poder computacional para funcionar. 7. El SD es fácil de implementar en hardware. 8. El SD debe tener su debida documentación y guía de uso para futuras implementaciones. 9. El SD debe tener un promedio de precisión de al menos 80 por ciento
3. Sería bueno tener esa característica, pero no es necesaria	10. El SD requiere de poca memoria RAM.
2. La función no es importante, pero no me importaría tenerla	11. El SD debe cumplir con estándares de código eficiente.
1. La función no es deseada	Ninguna necesidad entró en esta prioridad

La tercera etapa del proceso se muestra en la Figura 3. 1, esta etapa se encarga de dividir el problema en múltiples subproblemas los cuales tiene que resolver el SD final. Al subdividir los problemas se puede analizar a mayor profundidad y eficiencia el cómo se puede cumplir con las

diferentes necesidades destacadas anteriormente. Si se interpreta el SD como una caja negra, esta posee dos entradas, información en forma de video y la señal de arranque que se le va a dar al sistema. De ahí se desprenden varios subproblemas, como lo son, el detectar objetos, clasificar estos objetos, rastrearlos a través del tiempo. A su vez se debe determinar la distancia entre la cámara y los objetos detectados y la velocidad con la que estos se acercan al SD, estos datos son de vital importancia para hacer el manejo de banderas que requiere el sistema de control.

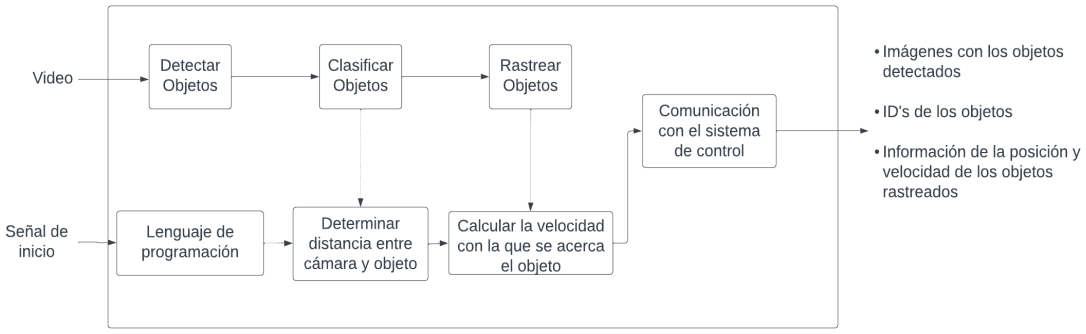


Figura 3. 1 División en subproblemas

3.1.2 Estado del arte

En la siguiente etapa se realizó una investigación sobre opciones en el mercado para posibles soluciones para los subproblemas encontrados.

3.1.2.1 Estrategias de detección de objetos:

El primer subproblema detectado es el de detectar objetos de forma automática, para ello existen múltiples estrategias que se pueden utilizar, las tres estrategias que mejor se adaptan a la aplicación deseada se describen a continuación:

Uso de máscaras: En el ámbito de la visión por computación, el uso de máscaras es una técnica utilizada para extraer partes específicas de las imágenes que son de especial relevancia para la aplicación que se esté desarrollando. Una máscara es una imagen binaria compuesta de píxeles blancos y negros que al ser convolucionadas con la imagen de entrada, permiten destacar un área de interés, una técnica que surge a partir de ello es el uso de capas umbrales, estas consisten en escoger un valor de tonalidad y todo píxel que sea mayor o menor a ese valor se convierten en píxeles

blancos o negros respectivamente, esto permite generar una segmentación de objetos bastante efectiva.

Las máscaras tienen otros usos, como lo son el filtrar ruido presente en las imágenes, detectar bordes o destacar un color en específico. Las máscaras son particularmente útiles en aplicaciones como la detección de objetos, donde el objetivo es identificar y ubicar objetos específicos dentro de una imagen. Al utilizar máscaras para aislar el objeto de interés, los algoritmos pueden detectar el objeto de manera más fácil y precisa y distinguirlo de otras partes de la imagen. En general, las máscaras son una herramienta poderosa en la visión por computadora para aislar y analizar regiones específicas de una imagen, y se utilizan ampliamente en una variedad de aplicaciones, desde el procesamiento de imágenes hasta el aprendizaje automático. [30]

Por medio de colores: En múltiples aplicaciones, el uso de color puede resultar ser una estrategia efectiva para detectar objetos en imágenes, para esto, existen múltiples espacios de colores que permiten codificar de forma efectiva cada diferente tonalidad y de esta manera, determinar y clasificar objetos. El espacio de color más utilizado en aplicaciones de *software* se trata del RGB, este utiliza los tres colores primarios y asigna un valor entre 0 y 255 para cada uno de estos, dependiendo del valor de cada color, se realiza una combinación y se obtiene un color específico. Para una aplicación de detección, se pueden utilizar los valores específicos para la combinación deseada, no obstante, si existen múltiples tonalidades del mismo color o no existe un contraste adecuado en la imagen, el detectar únicamente un color específico puede generar problemas.

Otro espacio de color utilizado en aplicaciones similares es el HSI, este sistema utiliza 3 datos, el tono, la saturación y la intensidad del color, esto permite realizar una segmentación de colores más efectiva y no tan limitada a tener un único valor de cada color primario, en general permite más flexibilidad a la hora de detectar objetos que se vean afectados por cambios de iluminación o bien de la perspectiva de la cámara. En sí, utilizar la técnica de utilizar espacios de colores depende completamente del entorno del problema que se desee atacar, puesto que su utilidad puede verse afectada gravemente si la aplicación se encuentra en cambios constantes en su ambiente. [31]

Utilizando algoritmos de IA: La visión por computadora y la inteligencia artificial han sido demostradas de complementarse de forma excepcional, ya que la IA ha sido comprobada de funcionar de forma positiva para tareas de clasificación, al poder clasificar información, se resuelve

uno de los principales retos de la visión por computadora que es el encontrar algún tipo de patrón en la imagen para poder relacionarlo con el tipo de objetos que se desean detectar, en grandes rasgos, la detección de objetos por medio de visión artificial busca estrategias para clasificar la información en la imagen. Asimismo, las técnicas que se utilizan en visión por computadora permiten facilitar el trabajo de los algoritmos de IA, ya que pueden llegar a filtrar y simplificar los datos que deben de ser procesados. [32]

3.1.2.2 Algoritmos de detección en tiempo real

Para solventar el subproblema que se refiere a la confección de algoritmos de detección y rastreo de objetos en tiempo real, en el mercado existen aplicaciones similares las cuales utilizan principalmente tres arquitecturas para clasificar datos de imágenes en el menor tiempo posible:

You Only Look Once (YOLO): La arquitectura *You Only Look Once* o "Solo miras una vez" en español, se trata de una arquitectura diseñada en 2016 por Joseph Redmon, Santosh Divvala, Ross Girshick, y Ali Farhadi de la Universidad de Washington. Ellos introdujeron al mundo la primera versión del algoritmo "solo miras una vez", el cual fue diseñado para detectar múltiples objetos en tiempo real con alta precisión. Se caracterizó por utilizar una sola red neuronal convolucional para realizar el proceso de detección y clasificación de objetos, lo cual permitió que el algoritmo fuese sumamente rápido. Versiones posteriores han mejorado la arquitectura de la red neuronal, permitiéndole utilizar técnicas de detección cada vez más complejas y con resultados más precisos, versiones posteriores como YOLOv4 y YOLOv5 han sido utilizadas en sistemas ADAS de automóviles inteligentes a lo largo del mundo. La versión más actual de esta arquitectura (YOLOv8) fue lanzada como fuente abierta el 10 de enero de 2023 y se mantiene en un avance constante tanto por la compañía fundada por sus creadores originales (Ultralytics) y gracias a la comunidad generada en el github oficial, siendo esta una de las más grandes en la plataforma. [33]

Como la gran mayoría de algoritmos de visión artificial, YOLO sigue una serie de pasos para lograr la detección y rastreo de objetos, inicialmente se obtiene la imagen a utilizar y se realiza un preprocesado, en el cual se normaliza la imagen y se alimente a la red neuronal con la información contenida en esta, luego la red neuronal convolucional se dedica a extraer la información de la imagen, a través de esto se genera la predicción o detección de objetos y se generan cuadros delimitadores que se encargan de señalar la sección de la imagen en la cual se encuentra el objeto detectado, luego se clasifica en las clases con las cuales fue entrenada la red neuronal inicialmente y finalmente se filtran resultados repetidos para obtener una salida mucho más acertada y evitar información sobrante.

La estructura básica de los modelos Yolo se compone de dos partes principales: la etapa de extracción de características y la etapa de detección. La primera parte del modelo Yolo es una red convolucional profunda que se utiliza para extraer características de la imagen de entrada. Usualmente, se utiliza una arquitectura previamente entrenada como Darknet-53 o una versión más liviana como *Tiny* Yolo, que tiene menos capas convolucionales. Después de la etapa de extracción de características, se agregan capas adicionales para realizar la detección de objetos. Esta parte del modelo se encarga de generar cuadros delimitadores (*Bounding boxes*) y clasificar los objetos detectados.

Generación de cuadros delimitadores: El modelo divide la imagen de entrada en una cuadrícula fija de celdas y cada celda predice múltiples cuadros delimitadores. Cada cuadro delimitador consiste en un conjunto de valores que representan las coordenadas de la caja (es decir, posición del objeto en la imagen) y una medida de confianza que indica la certeza de que la caja contiene un objeto. Además de los cuadros delimitadores, el modelo también realiza la clasificación de los objetos detectados en cada celda. Cada celda predice la probabilidad de varias clases de objetos posibles. Esto se logra mediante el uso de capas convolucionales y una capa de clasificación final que utiliza activaciones de funciones como Softmax o ReLu para obtener las probabilidades. [34]

Faster Region Convolutional Neural Network (F-RCNN): También conocido por su traducción al español "Red Neuronal Convolucional Basada en Regiones". Este algoritmo, fue desarrollado en 2016 por Jifeng Dai, Yi Li, Kaiming He y Jian Sun de Microsoft Research Asia. Se trata de un marco de detección de objetos de dos etapas. En la primera etapa, la imagen de entrada se pasa a través de una red neuronal convolucional (CNN) para extraer características. En la segunda etapa, se generan y refinan las propuestas de objetos utilizando la información extraída para obtener los resultados finales de detección de objetos.

La clave de la innovación de F-RCNN es que utiliza una arquitectura completamente convolucional para la segunda etapa, lo que permite que la red opere en toda una imagen de una vez y produzca detecciones de objetos para todas las posibles posiciones en la imagen en una sola pasada hacia adelante. Esto hace que F-RCN sea más rápido y eficiente que los métodos basados en regiones tradicionales que dependen de la extracción y clasificación de características por región. [35]

En F-RCNN, la red completamente convolucional consta de una Red de Propuestas de Región (RPN) que genera propuestas de región, seguida de una Red Convolucional Totalmente Basada en

Regiones (RCNN) que realiza la detección de objetos en estas propuestas. La RPN utiliza un enfoque de ventana deslizante para generar propuestas de región y se entrena para predecir las puntuaciones de objetividad y las coordenadas del cuadro delimitador para cada propuesta. El FCNN toma las propuestas de región generadas por la RPN y realiza la detección de objetos utilizando una arquitectura completamente convolucional. El RCNN utiliza mapas de puntuación sensibles a la posición para predecir la categoría del objeto y refinar las coordenadas del cuadro delimitador para cada propuesta de región. Los mapas de puntuación sensibles a la posición se aprenden durante el entrenamiento y codifican la ubicación espacial de cada objeto dentro de la propuesta de región.

Durante el entrenamiento, la red se optimiza utilizando una función de pérdida multitarea que combina la pérdida de clasificación y la pérdida de regresión del cuadro delimitador. La pérdida de clasificación mide la precisión de la clasificación de objetos, mientras que la pérdida de regresión del cuadro delimitador mide la precisión de las coordenadas del cuadro delimitador predichas. F-RCNN ha logrado resultados de vanguardia en conjuntos de datos de detección de objetos de referencia y se ha utilizado ampliamente en aplicaciones como conducción autónoma, vigilancia y robótica. [36]

Single Shot Detection (SSD): Traducido al español como Red Neuronal Convolucional Basada en Regiones. Se origina en 2016 por Wei Liu, Dragomir Anguelov y el equipo de investigación y desarrollo de Google, nace la detección de disparo único o SSD, la idea principal de esta arquitectura es el utilizar una única red neuronal para realizar la clasificación y el desarrollo de cuadros delimitadores al mismo tiempo, lo cual recorta tiempo de procesamiento.

La arquitectura SSD se compone de una red neuronal convolucional (CNN) de base y un conjunto de capas convolucionales que se utilizan para detectar objetos a diferentes escalas y relaciones de aspecto. La red de base suele ser una CNN previamente entrenada, que se utiliza para la extracción de características. Una vez que se extraen las características, se alimentan en un conjunto de capas de detección que son responsables de predecir las clases de objetos y las cajas delimitadoras. Las capas de detección están diseñadas para operar a diferentes escalas espaciales y son responsables de detectar objetos de diferentes tamaños. [37]

Durante el entrenamiento, la red SSD se optimiza utilizando una función de pérdida de múltiples tareas que combina tanto las pérdidas de clasificación como las de localización. Se ha demostrado que SSD logra resultados de vanguardia en la detección de objetos y se utiliza ampliamente en aplicaciones del mundo real, como automóviles autónomos, robótica y vigilancia de video. [38]

3.1.2.3 Rastreo de objetos

Las arquitecturas descritas anteriormente, permiten la detección y clasificación de objetos en tiempo real, no obstante, también se requiere de cumplir con la labor de rastreo de los objetos detectados previamente, el rastreo de objetos permite aproximar la velocidad con la que los objetos detectados se acercan al vehículo autónomo, consiste en que cada vez que se detecta un objeto, se le asigna un ID, el cual se utiliza para analizar su movimiento a través de los distintos fotogramas con forme se van recibiendo, es importante denotar que la precisión del rastreo es sumamente importante puesto que se utiliza para definir parámetros que tienen peso en la toma de decisiones y control del automóvil, en la industria automotriz existen dos algoritmos de rastreo que destacan por su efectividad en aplicaciones relevantes para la conducción autónoma:

Simple Online and Realtime Tracking (SORT): SORT o Seguimiento simple en línea y en tiempo real traducido al español, es un algoritmo de seguimiento que se puede utilizar para rastrear objetos en un flujo de video. Está diseñado para ser eficiente y funcionar en tiempo real, lo que lo hace particularmente útil para aplicaciones como la vigilancia o la conducción autónoma. La idea básica detrás de SORT es utilizar una combinación de predicción de movimiento y coincidencia de apariencia para rastrear objetos con el tiempo, utilizando una técnica llamada filtros de Kalman.

Detección: El primer paso en el seguimiento con SORT es detectar el objeto de interés en el primer fotograma del video. Esto se hace típicamente utilizando una técnica de visión por computadora como detección de objetos o segmentación. Su funcionamiento se puede describir de la siguiente manera:

- Inicialización: Una vez que se ha detectado el objeto, SORT inicializa una nueva pista para él. Esto implica asignar un identificador único al objeto y crear una estimación inicial del estado para su posición, tamaño y velocidad.
- Predicción: Después de que se ha inicializado el objeto, SORT utiliza un modelo de movimiento para predecir dónde estará el objeto en el siguiente fotograma. Esta predicción se basa en la posición y la velocidad previas del objeto.
- Coincidencia: En el siguiente fotograma, se detecta nuevamente el objeto y SORT intenta emparejarlo con la pista que se creó en el paso 2. Esto se hace utilizando características de apariencia como color, textura o forma.

- Actualización: Si el objeto se empareja con éxito con la pista, SORT actualiza la estimación del estado de la pista con la nueva información de detección. Si el objeto no se puede emparejar con ninguna pista existente, SORT crea una nueva pista para él.

- Eliminación: Finalmente, SORT elimina las pistas que no se han actualizado durante un cierto número de fotogramas. Esto ayuda a evitar el seguimiento de falsos positivos u objetos que han abandonado la escena

En general, SORT es un algoritmo efectivo y eficiente para el seguimiento de objetos en un flujo de video. Puede manejar una gran cantidad de objetos simultáneamente y puede rastrear objetos incluso si desaparecen temporalmente de la vista o se ocultan entre sí. Se busca un estándar de al menos 70%, este puesto que en ambientes reales existen múltiples variables (iluminación, condiciones climáticas, etc.) que pueden provocar que el sistema de rastreo "pierda de vista" al objeto detectado. [39]

ByteTrack: ByteTrack o traducido al español "Seguimiento de múltiples objetos mediante la asociación de cada cuadro de detección", es un algoritmo de seguimiento de múltiples objetos que se basa en el enfoque de "asociar cada caja de detección" ("*associating every detection box*"). La idea básica es asignar una identidad única a cada objeto en cada fotograma del video, utilizando todas las cajas de detección que se han detectado en ese fotograma.

De forma similar al algoritmo SORT, ByteTrack sigue una lista de pasos o acciones para poder realizar el rastreo de objetos de forma efectiva:

- Detección: El primer paso es detectar las cajas de objetos en cada fotograma del video utilizando una red de detección de objetos, como YOLO o Faster R-CNN.

- Asociación de cajas de detección: Luego, se asigna una identidad única a cada objeto en cada fotograma utilizando todas las cajas de detección en ese fotograma. Para hacer esto, ByteTrack utiliza un algoritmo de emparejamiento que considera la similitud espacial y visual entre cada par de cajas de detección.

- Predicción: Después de que se ha asignado una identidad a cada objeto en el fotograma actual, ByteTrack utiliza un modelo de predicción de movimiento para predecir la posición del objeto en el siguiente fotograma. Esta predicción se basa en las posiciones previas del objeto y la velocidad de movimiento.

- Verificación: En el siguiente fotograma, ByteTrack detecta nuevamente las cajas de objetos y las asocia con las identidades asignadas en el paso anterior. Si la caja de detección se asocia correctamente con la identidad correcta, la identidad se mantiene. Si la caja de detección no se puede asociar con una identidad existente, se crea una nueva identidad para el objeto.
- Eliminación: Finalmente, ByteTrack elimina las identidades que no se han actualizado durante un cierto número de fotogramas. Esto ayuda a evitar el seguimiento de objetos que han desaparecido de la escena.

En general, ByteTrack es un algoritmo de seguimiento de objetos preciso y robusto que puede manejar múltiples objetos en una escena. La clave de su éxito es el enfoque de "asociar cada caja de detección", que permite que cada objeto tenga una identidad única en cada fotograma, incluso cuando los objetos se superponen o cambian de forma. [40]

3.1.2.4 Aproximación de distancias

Otro subproblema detectado para el desarrollo del SD es la necesidad de determinar la distancia entre la cámara y los objetos que se lleguen a detectar en las imágenes obtenidas, para ello existen múltiples metodologías las cuales permiten generar aproximaciones a la distancia deseada, las técnicas más frecuentemente utilizadas en la industria son el método triangular, el método de tamaño conocido y el método de profundidad de enfoque.

Método triangular: Esta técnica utiliza relaciones conocidas como triángulos semejantes para poder realizar aproximaciones en el tamaño del campo de visión de la cámara, para poder aplicar este método se necesita conocer la longitud focal de la cámara y el tamaño de los sensores de visión CCD de la cámara, con estos dos valores se puede generar un triángulo imaginario el cual llega hasta el lente, donde se refleja y se crea un triángulo equivalente el cual se rige por la distancia de trabajo y el campo de visión, tal y como se muestra en la Figura 3. **2**. [41] A partir de ello, se utiliza trigonometría para crear una relación entre todas las variables involucradas en la captura de la imagen y de esta manera se obtiene una ecuación que permite obtener el valor de cualquiera de estas variables presentes.

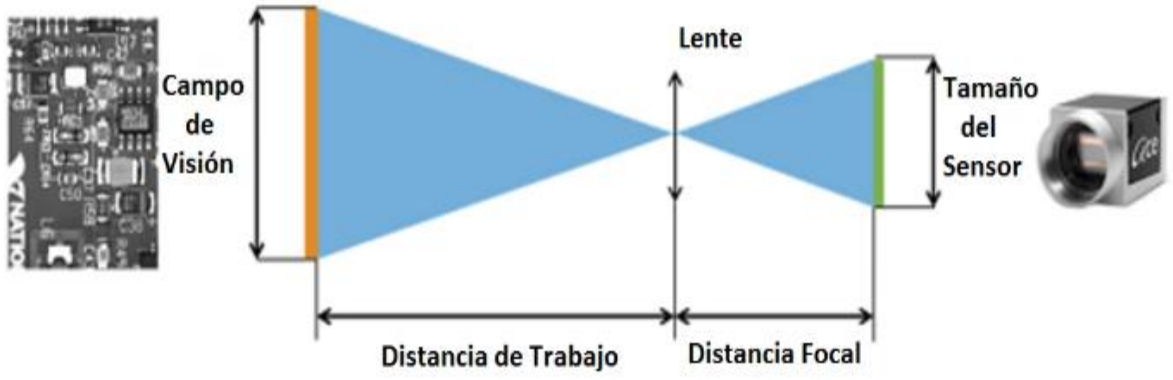


Figura 3. 2 Semejanza de triángulos utilizada en el método triangular. [42]

Método del tamaño conocido: Esta técnica es ampliamente utilizada en aplicaciones de visión por computadora, consiste en tomar una fotografía de un objeto cuyo tamaño es conocido, a partir de este se puede realizar una aproximación según la cantidad de pixeles que abarca el objeto en la imagen, de esta manera es posible establecer una relación para transformar unidades de medición reales a píxeles en una imagen, esta técnica se utiliza ampliamente para aplicaciones en las que la cámara no se mueve, por lo que la distancia de trabajo se mantiene de forma constante y resulta más sencillo aproximar tamaños de objetos. [43]

Método de profundidad de enfoque: es una técnica para estimar la distancia de un objeto desde una cámara mediante el análisis de la nitidez de diferentes áreas de la imagen. Se basa en la suposición de que los objetos a diferentes distancias de la cámara tendrán diferentes niveles de enfoque y que el grado de nitidez en la imagen se puede usar para determinar la distancia. El método implica capturar una serie de imágenes de la misma escena mientras se cambian los ajustes de enfoque de la cámara. Luego se analizan las imágenes para identificar el área de enfoque más nítida para cada imagen. Al comparar la nitidez de diferentes áreas a lo largo de la serie de imágenes, es posible estimar la profundidad de cada punto en la escena.

Sin embargo, debe tenerse en cuenta que la precisión del método de profundidad a partir del enfoque depende de varios factores, como la calidad de la cámara y las condiciones de iluminación de la escena. Además, el método puede no ser adecuado para todos los tipos de imágenes, como aquellas con escenas complejas o bajo contraste.

3.1.2.5 Lenguaje de programación

Dentro de los subproblemas denotados se encuentra el determinar un lenguaje de programación en el cual se va a correr el algoritmo de visión artificial. Se consideraron dos alternativas

principales: Python y C#, C++. Los cuales son los lenguajes más comúnmente utilizados en aplicaciones de detección y clasificación de objetos. La **Tabla 3. 3** presenta las principales diferencias entre ambas opciones. [44]

Tabla 3. 3 Cuadro comparativo entre lenguajes de programación

Python	C++
• Diseño simple para prototipado • Facilita el solucionar problemas • Posee algoritmos pre hechos de detección de objetos • La mayoría de los marcos de trabajo o *frameworks* de C++ tienen una versión compatible con Python	• Tiene un mejor control de los recursos computacionales que se utilizan • Utiliza menor tiempo de procesamiento • Permite una mejor implementación en tarjetas embebidas • No todos los marcos de trabajo o *frameworks* de Python se encuentran adaptados para su uso en C++

3.1.2.6 Protocolos de comunicación

El último subproblema por solucionar es el método de comunicación con el sistema de control, hay varias consideraciones importantes para realizar esta comunicación, debido al alcance del proyecto y las características de este, resaltan tres protocolos de comunicación que se pueden llegar a utilizar:

TCP/IP: El protocolo de control de transmisión/Protocolo de Internet es el protocolo de comunicación más utilizado a nivel global, se destaca por su uso en Internet y la mayoría de las redes informáticas modernas. Se trata de un conjunto de reglas que gobiernan cómo se transmite la información entre dispositivos en una red.

TCP es responsable de asegurar que la información se transmita de manera confiable entre dispositivos. Divide la información en paquetes y agrega información a cada paquete, como la dirección de destino y un número de secuencia. Luego, envía estos paquetes a través de la red, y el dispositivo receptor los reensambla en la información original. **IP** es responsable de enrutar los paquetes de información de un dispositivo a otro a través de la red. Agrega un encabezado a cada

paquete que contiene las direcciones de origen y destino, así como otra información necesaria para dirigir el paquete a su destino.

Cuando un dispositivo desea enviar información a través de la red, primero establece una conexión con el dispositivo receptor utilizando TCP. Esto implica una confirmación en el que los dispositivos intercambian mensajes para establecer una conexión y acordar parámetros, como el tamaño máximo de los paquetes que se pueden enviar. En general, TCP/IP proporciona un medio confiable y eficiente para transmitir información entre dispositivos en una red. [45]

UDP: El protocolo de datagramas de usuario es un protocolo inalámbrico usado en aplicaciones como comunicaciones por voz o transmisión de video en tiempo real, es característicamente más rápido que TCP/IP, no obstante, no posee un control de confiabilidad o control de congestión de datos, por lo que se prefiere en aplicaciones que requieren alta velocidad y enviar pocos datos a la vez. [46]

CANbus: Se trata de un sistema de comunicación utilizado principalmente en sistemas automotrices, robótica y automatización, su enfoque es principalmente a la transmisión de información de manera confiable y a alta velocidad. Utiliza una topología de tipo bus lo que significa que múltiples dispositivos se pueden comunicar por medio de la misma red. [47]

Finalmente, se recopila la información en la **Tabla 3. 4**. La cual tiene la función de ser un resumen para los múltiples subproblemas detectados y las posibles soluciones sobre las cuales se investigó.

Tabla 3. 4 Lista resumen de la búsqueda interna y externa de soluciones

Detectar objetos	Uso de máscaras
	Por medio de colores
	Utilizando algoritmos de IA
Clasificar Objetos	YOLO
	F-RCNN
	SSD
Rastrear Objetos	SORT
	ByteTrack
Determinar distancia entre cámara y objetos	Método triangular
	Método del tamaño conocido
	Método de profundidad del enfoque
Lenguaje de programación	Python
	C#, C++
Comunicación con el sistema de control	TCP/IP
	UDP
	CANbus

3.1.3 Alternativas de solución

La siguiente etapa consta de la combinación y calificación de candidatos a solución, es decir, para cada subproblema se eligen candidatos y se forman combinaciones de soluciones de estos para generar una solución general para el problema planteado. La lista de subproblemas se puede apreciar en la tabla 3. Cada subproblema debe cumplir con métricas definidas para verificar si cumplen con el comportamiento deseado para atacar el problema. Retomando las necesidades representadas en la tabla 2, se establecen las métricas denotadas en la **Tabla 3. 5**.

Tabla 3. 5 Métricas definidas para cumplir con las diferentes necesidades encontradas

Número de métrica	Número de necesidad	Métrica	Importancia	Unidades	Valor marginal	Valor Ideal
1	1, 4	Precisión al detectar objetos en ambientes sencillos	5	Porcentaje	> 80	> 90
2	1,4	Precisión al detectar objetos en ambientes medianamente complejos	5	Porcentaje	>70	>80
3	1,4	Precisión al detectar objetos en ambientes complejos	4	Porcentaje	>30	>40
4	2	Tiempo de procesado de imágenes	5	ms	> 50	> 25
5	3	Clases distintas de objetos en que se puede clasificar	5	Cantidad	2	> 3
6	4	Precisión del rastreo	5	Porcentaje	> 70	> 90
7	5	Tiempo de procesamiento y envío de información a la capa de control	5	ms	> 30	< 30
8	6	Potencia utilizada por el SD	4	W	< 200	< 50
9	9	Precisión del sistema completo	4	Porcentaje	90	100
11	11	Memoria RAM necesaria	3	GB	> 20	< 8

Todas las métricas denotadas son importantes para cumplir con las necesidades planteadas, todas se basan en la información definida en la investigación interna y externa que se realizó previamente, así como información importante por parte del cliente. El cliente definió la importancia de parámetros como el número de clases. Se espera poder utilizar de forma eficaz al menos 2 clases de objetos distintas, con el fin de poder simular casos de posibles accidentes tanto con automóviles como con peatones, es preferible el uso de mínimo una clase más para tener una mayor variedad de escenarios de prueba, así como tomar distintas decisiones que dependen del tipo de objeto que se detecte.

Por otra parte, se define un retraso en el tiempo de reacción en ambientes reales en comparación con la reacción en ambientes simulados debido a que en ambientes reales se generan imágenes más complejas (debido a cambios de iluminación, calidad de cámara, condiciones climáticas, entre otras razones) para el funcionamiento del detector de objetos, lo que genera un mayor tiempo de procesamiento en el sistema diseñado, de igual manera se apunta a cumplir con los límites definidos por las múltiples normativas sobre la velocidad de procesamiento de aplicaciones de visión por computadora en conducción autónoma. Directamente relacionado con este tiempo de procesamiento, se desea optimizar lo más posible el funcionamiento del sistema, por lo que se desea utilizar la menor cantidad de potencia posible (los sistemas embebidos comerciales utilizan entre 20 y 200 W, con algunos modelos incluso superando los 300 W) así como memoria RAM (Las ECU de sistemas ADAS cuentan con varios valores de memoria RAM, cuando se utilizan en aplicaciones de detección en tiempo real pueden utilizar desde 8GB hasta 20 GB de memoria) [48]

La precisión del sistema completo se refiere a la precisión entre la información enviada por el algoritmo de detección de objetos y la respuesta del sistema de control, esta se ve afectada principalmente por la comunicación que exista entre ambos subsistemas además de la eficacia con la que se logre optimizar los subsistemas de detección de objetos y de control de manera individual.

3.1.4 Generación de conceptos

A partir de la definición de métricas y de subproblemas, se plantean múltiples conceptos a evaluar como posibles soluciones las cuales se resumen en las tablas a continuación:

Tabla 3. 6 Concepto A

Subproblema	Solución Seleccionada
Detectar objetos	Utilizando algoritmos de IA
Clasificar Objetos	YOLO
Rastrear Objetos	SORT
Determinar distancia entre cámara y objetos	Método triangular
Lenguaje de programación	Python
Comunicación con el sistema de control	TCP/IP – UDP (Existe la libertad de utilizar ambos)

Tabla 3. 7 Concepto B

Subproblema	Solución Seleccionada
Detectar objetos	Utilizando algoritmos de IA
Clasificar Objetos	F-RCNN
Rastrear Objetos	ByteTrack
Determinar distancia entre cámara y objetos	Método triangular
Lenguaje de programación	Python
Comunicación con el sistema de control	TCP/IP

Tabla 3. 8 Concepto C

Subproblema	Solución Seleccionada
Detectar objetos	Utilizando algoritmos de IA
Clasificar Objetos	SSD
Rastrear Objetos	SORT
Determinar distancia entre cámara y objetos	Método del tamaño conocido
Lenguaje de programación	Python
Comunicación con el sistema de control	TCP/IP

Tabla 3. 9 Concepto D

Subproblema	Solución Seleccionada
Detectar objetos	Uso de máscaras
Clasificar Objetos	F-RCNN
Rastrear Objetos	ByteTrack
Determinar distancia entre cámara y objetos	Método del tamaño conocido
Lenguaje de programación	Python
Comunicación con el sistema de control	CANbus

Tabla 3. 10 Concepto E

Subproblema	Solución Seleccionada
Detectar objetos	Por medio de colores
Clasificar Objetos	YOLO
Rastrear Objetos	SORT
Determinar distancia entre cámara y objetos	Método del tamaño conocido
Lenguaje de programación	Python
Comunicación con el sistema de control	UDP

Tabla 3. 11 Concepto F

Subproblema	Solución Seleccionada
Detectar objetos	Utilizando algoritmos de IA
Clasificar Objetos	SSD
Rastrear Objetos	ByteTrack
Determinar distancia entre cámara y objetos	Método triangular
Lenguaje de programación	Python
Comunicación con el sistema de control	TCP/IP

Una vez con todos los conceptos planteados, es necesario realizar una comparación con un sistema comercial de forma de referencia y así poder ya sea descartar algún concepto, o bien detectar tendencias en los conceptos. Se toma como referencia el algoritmo de detección denominado Yolov5-ADAS, el cual se utiliza en aplicaciones reales de conducción autónoma. Tal y como se explica en [51], la forma correcta de analizar la precisión de distintos casos, los cuales se clasifican en imágenes sencillas, moderadamente complejas y considerablemente complejas, puesto que la aplicación que se desea es muy volátil y efectos externos como el clima pueden llegar a perjudicar su funcionamiento. Para lograr cumplir este experimento, el conjunto de datos denominado KITTI, posee ejemplos de los tres casos señalados con el fin de realizar estudios en base a este, tal y como se muestra en la **Tabla 3. 12**.

Aparte de la precisión en la detección, se necesita poco tiempo de procesamiento, alta efectividad en la comunicación (tanto a nivel velocidad como en certeza de que llegue la información de forma correcta), así como se requiere un alto nivel de precisión en el rastreo ya que cualquier cambio en el entorno puede hacer que el objeto detectado se vea distinto y el algoritmo no sea capaz de rastrearlo, lo que puede afectar al funcionamiento efectivo del SD. En la **Tabla 3. 13** se denota cómo se realizó esta etapa de filtrado entre los conceptos tentativos.

Tabla 3. 12 Resultados de experimentos del modelo comercial Yolov5-ADAS en el conjunto de datos "KITTI" [51]

Referencia	Imagen Sencilla (%)	Imagen Moderada (%)	Imagen Compleja (%)	Promedio (%)	Tiempo de procesamiento promedio (s)
Automóvil	95.92	93.31	85.44	91.45	0.019
Ciclista	82.29	66.33	61.23	69.4	0.025
Peatón	71.49	54.07	47.82	56.96	0.050

Tabla 3. 13 Calificación y filtrado de conceptos

	Conceptos						
Criterios de selección	Yolov5 - ADAS	A	B	C	D	E	F
Precisión en imágenes sencillas	0	0	+	+	+	-	+
Precisión en imágenes moderadamente complejas	0	+	+	+	0	-	+
Precisión en imágenes considerablemente complejas	0	-	0	0	-	-	0
Tiempo de detección promedio	0	+	-	+	-	+	-
Precisión del rastreo	0	+	+	-	+	-	+
Precisión en la distancia estimada entre SD y objeto detectado	0	+	+	-	-	-	+
Efectividad de comunicación	0	+	+	+	0	0	+
Suma +	0	5	5	4	2	1	5
Suma 0	0	1	1	1	2	1	1
Suma -	0	1	1	2	3	5	1
Evaluación neta	0	4	4	2	-1	-4	4
Lugar	**Ref**	**1**	**1**	**2**	**3**	**4**	**1**
¿Continuar?		Sí	Sí	No	No	No	Sí

Luego de la filtración inicial se destacaron tres conceptos ganadores, estos se evaluaron según su capacidad de cumplir con las diferentes necesidades detectadas, tal y como se muestra en la **Tabla 3. 14**, no obstante, la calificación obtenida en el análisis de los tres conceptos obtuvo resultados muy similares, los tres conceptos se complementan en el sentido de que las debilidades de unos complementan las fortalezas de otros, por ende, tomando en cuenta el objetivo 3, se decide desarrollar estos tres conceptos ganadores puesto que son los mejores adaptados para la detección de objetos en tiempo real, además tienen una diferencia importante entre sí con la arquitectura que utilizan para la IA encargada de la clasificación de objetos, por lo que se puede realizar una comparación entre los tres prototipos en software para determinar cuál es el concepto verdaderamente ganador y que debería ser desarrollado si el proyecto llegase a la etapa de validación.

Tabla 3. 14 Selección de conceptos

		Conceptos					
Criterios de selección	**Peso (%)**	**A**		**B**		**F**	
		Calificación	Evaluación ponderada	Calificación	Evaluación ponderada	Calificación	Evaluación ponderada
Precisión en imágenes sencillas	16,67	80	13,33	90	15,00	80	13,33
Precisión en imágenes moderadamente complejas	16,67	80	13,33	80	13,33	70	11,67
Precisión en imágenes considerablemente complejas	13,33	60	8,00	70	9,33	60	8,00
Tiempo de detección promedio	16,67	80	13,33	60	10,00	80	13,33
Precisión del rastreo	10,00	60	6,00	60	6,00	70	7,00
Precisión en la distancia estimada entre SD y objeto detectado	10,00	70	7,00	70	7,00	80	8,00
Efectividad de comunicación	16,67	80	13,33	70	11,67	80	13,33
Total puntos	100,00		74,33		72,33		74,67
Lugar		**2**		**3**		**1**	
¿Continuar?		*Considerar*		*Considerar*		*Sí*	

Una vez se seleccionaron las variaciones a prototipar de las posibles soluciones, se definen las pruebas de validación con las cuales se podrá definir si se cumple con las métricas establecidas previamente, en las tablas más adelante, se describe cada una de las pruebas a realizar y cómo se evalúa esta, además en la **Tabla 3. 20** se describe en forma de resumen el objetivo de cada una de estas pruebas.

Tabla 3. 15 Prueba de validación 1

Objetivo: Verificar la precisión al detectar objetos de los algoritmos diseñados	Número de prueba: 1
Descripción de la prueba: Se requiere verificar la capacidad de los distintos modelos desarrollados de poder clasificar de forma efectiva. El conjunto de datos "KITTI" posee un subconjunto general de validación en el cual se encuentran las clases de peatón, ciclista y vehículo. Se va a realizar una validación con dicho conjunto de datos para conocer el valor de precisión de cada una de las clases para cada algoritmo, para obtener resultados aceptables, se requiere un valor de al menos 80% en la clase de peatón y 90% en las otras dos clases.	
Naturaleza de la prueba: Prueba de funcionamiento Software: Python	

Tabla 3. 16 Prueba de validación 2

Objetivo: Verificar la precisión para detectar objetos en distintos ambientes	Número de prueba: 2
Descripción de la prueba: Se requiere verificar la capacidad de los distintos modelos desarrollados de poder clasificar de forma efectiva. El conjunto de datos "KITTI" tiene distintos subconjuntos de datos los cuales poseen imágenes clasificadas como "sencillas", "moderadamente complejas" y "complejas". Estas tienen características distintas que pretenden poner a prueba a los algoritmos ante ambientes realistas, donde existen múltiples cambios en el ambiente del automóvil. Se desea sobrepasar los resultados obtenidos por Yolov5-ADAS (Tabla 10)	
Naturaleza de la prueba: Prueba de funcionamiento Software: Python	

Tabla 3. 17 Prueba de validación 3

Objetivo: Verificar la velocidad de procesamiento de los algoritmos y la eficiencia de los métodos de comunicación	Número de prueba: 3
Descripción de la prueba: Para garantizar un funcionamiento adecuado del SD, la capa de percepción debe ser más rápida que el estándar de cámaras de captura de video en autos autónomos, el cual es de 20 fotogramas por segundo o bien 50 milisegundos, por lo que en esta prueba, se van a poner a funcionar a los algoritmos diseñados en varios ambientes de simulación creados en la plataforma "Monodrive", en el cual se va a analizar el tiempo de procesamiento para cada fotograma y se debe cumplir con que en ningún momento sobrepase los 50 milisegundos. Serán escenarios distintos en los cuales se simulen casos comunes para el funcionamiento del SD. Los escenarios para utilizar son los siguientes: • Un automóvil que se encuentra delante del auto autónomo frena de golpe • Un automóvil realiza un cambio de carril frente al auto autónomo • El auto autónomo se acerca a una intersección en la cual un automóvil entra de golpe pese a no tener vía Además, se requiere que la comunicación entre la capa de percepción y las demás capas del SD no acapare un alto nivel de recursos y tiempo de procesamiento. Se realizarán simulaciones con los ambientes descritos anteriormente en que se pongan a prueba múltiples tipos de reacciones y a partir de ahí se va a comparar la diferencia de tiempo de ejecución cuando los algoritmos de detección funcionan con entradas de datos guardados localmente y con datos obtenidos en tiempo real. La comunicación no debe superar el 15% del tiempo de ejecución puesto que, si se aumenta este valor, en una iteración que requiera mayor tiempo de procesamiento, la comunicación puede provocar que se exceda el límite de tiempo de 50 milisegundos.	
Naturaleza de la prueba: Prueba de funcionamiento Software: Python	

Tabla 3. 18 Prueba de validación 4

Objetivo: Verificar la precisión del sistema de rastreo	Número de prueba: 4
Descripción de la prueba: Para el correcto funcionamiento de la capa de control del SD, se necesita cumplir con un rastreo efectivo de objetos en tiempo real. Se realizará una prueba en la cual se utilizará de entrada videos realizados con la ayuda de la plataforma "Monodrive", en los que se simularán escenarios comunes para el funcionamiento del SD. Los escenarios para utilizar son los siguientes: • Un automóvil que se encuentra delante del auto autónomo frena de golpe • Un automóvil realiza un cambio de carril frente al auto autónomo • El auto autónomo se acerca a una intersección en la cual un automóvil entra de golpe pese a no tener vía En estos videos se va a conocer la cantidad de fotogramas en que se encuentran objetos presentes, entonces se correrán los algoritmos con dicha entrada y en la salida se guardarán los diferentes fotogramas con el fin de comparar la cantidad de fotogramas en los que se detectó el objeto y se rastreó de forma efectiva y de esta forma obtener el porcentaje de error del sistema de rastreo.	
Naturaleza de la prueba: Prueba de funcionamiento Software: Python	

Tabla 3. 19 Prueba de validación 5

Objetivo: Verificar la precisión de los cálculos de distancia entre objeto y cámara	Número de prueba: 6
Descripción de la prueba: La capa de percepción necesita ser capaz de aproximar la distancia existente entre el objeto a detectar y la posición del vehículo autónomo. Para esta prueba se van a utilizar imágenes en las cuales se tiene certeza de la distancia real entre el objeto y la cámara que capturó dicha imagen y se comparará con los resultados dados por los algoritmos diseñados, obteniendo así un porcentaje de error que no debe pasar del 5% por motivos de seguridad para la aplicación.	
Naturaleza de la prueba: Prueba de funcionamiento Software: Python	

Tabla 3. 20 Resumen de las pruebas de validación

Número de prueba	Necesidad	Objetivo	Variable de muestra	Criterios	Factores de influencia
1	El SD debe detectar objetos en tiempo real.	El promedio de detección general con el conjunto de datos de validación debe superar el 80%	Precisión durante la detección	Robustez del modelo de detección de objetos	Estructura del modelo de IA, calidad de la imagen, hardware utilizado
2	El SD debe tener un promedio de precisión adecuado	Debe superar las métricas explicadas en la **Tabla 3. 5**	Precisión durante la detección	Robustez del modelo de detección de objetos	Estructura del modelo de IA, calidad de la imagen, hardware utilizado
3	El SD debe cumplir un tiempo de procesado de imágenes menor que 50 ms.	Tiempo de procesado menor que 50 ms	Tiempo de procesamiento/inferencia	Velocidad del modelo de detección de objetos	Estructura del modelo de IA, calidad de la imagen, hardware utilizado
4	El SD debe rastrear objetos que ya detectó anteriormente.	Porcentaje de error en el rastreo debe ser menor a 5%	Cantidad de fotogramas en los que se detecta un objeto conocido	Robustez del modelo de rastreo	Estructura del modelo de IA, claridad de la imagen, hardware utilizado
5	El SD debe poder comunicarse con las demás capas del sistema ADAS de forma eficiente	Retraso en el funcionamiento debido a comunicación debe ser menor al 15%	Tiempo de envío de información	Comunicación efectiva	Hardware utilizado, tipo de comunicación
6	El SD debe ser capaz de calcular la distancia entre el objeto detectado y el vehículo	El porcentaje de error en los cálculos debe ser menor al 5%	Precisión en los cálculos de distancia	Robustez del algoritmo de cálculo de distancias	Hardware utilizado, metodología para calcular distancias

4 PROPUESTA DE DISEÑO

En este capítulo se pretende abordar de manera detallada el desarrollo del diseño ingenieril llevado a cabo para la alternativa seleccionada en el capítulo anterior, el actual capítulo se divide en cinco secciones según sea el tipo de diseño realizado, higiénico y seguridad, estructura de secadora, mecánico, automatización o del SCADA. El presente incluye diagramas, cálculos matemáticos, equipo o componentes seleccionados y criteriosutilizados a lo largo del proceso de diseño, en fin, el paso a paso de la solución presentada.

4.1 Entrenamiento de algoritmos de IA para los sistemas de detección

El proceso de diseño inició con el entrenamiento de varios modelos de IA con la finalidad de clasificar imágenes en tiempo real, siguiendo lo visto en la investigación previa, se utilizaron las tres arquitecturas definidas en los conceptos considerados como posibles ganadores (YOLO, SSD y F-RCNN). Para poder entrenar de manera efectiva estos modelos de clasificación se requieren datos, por ende, se utiliza el set de datos conocido como KITTI, el cual ha sido aprobado por el Instituto Tecnológico de Karlsruhe y el Instituto Tecnológico de Toyota de Chicago. Este conjunto de datos consiste en 17000 imágenes en las que se presentan autos de todo tipo de modelos, peatones, ciclistas, trenes y en general todo tipo de objetos que se pueden encontrar en carreteras. El conjunto de datos se compone de tres subdivisiones, imágenes sencillas, imágenes moderadamente complejas e imágenes complejas, las cuales se dividen dependiendo de la dificultad que representa para la mayoría de los algoritmos de detección de objetos el poder clasificar correctamente dichos objetos. [52]

La primera solución utiliza la arquitectura de YOLOv8, para entrenar un modelo de YOLOv8, se realizó un estudio exhaustivo en el cual se pusieron a prueba múltiples hiperparámetros con el fin de obtener la mejor optimización posible para el modelo de inteligencia artificial. En la **Tabla 4. 1** se definen los hiperparámetros a poner a prueba en el estudio exhaustivo, para la cantidad de épocas se evaluaron los valores de 64, 128 y 256 por ser números potencias de 2, está demostrado según [53] que esta es una práctica que reduce el tiempo de entrenamiento, así como recursos computacionales.

Además de las épocas, los principales hiperparámetros por estudiar son la tasa de aprendizaje, tamaño del lote de entrenamiento y el modelo base, para el tamaño de lote se escogen valores que sean potencias de 2 por razones similares a la cantidad de épocas, mientras que en la tasa de aprendizaje los valores dependen de la función de activación que se utilice, los modelos YOLO utilizan ADAM, por lo que se utilizaron los

valores recomendados para dicha función de activación. Finalmente, para evitar una asignación de características erróneas, se utilizan de modelos base los creados por los desarrolladores de YOLOv8, los modelos *nano, small y large* (Yolov8-n, Yolov8-s y Yolov8-l respectivamente), se diferencian entre la capacidad para tener una alta precisión a la detección a coste de velocidad, el modelo Yolov8-n es el más rápido en procesar datos, pero tiene peor precisión promedio, mientras que el modelo Yolov8-l destaca por su elevada precisión, pero decae en velocidad. El modelo Yolov8-s es un punto medio entre ambos modelos anteriormente descritos.

Tabla 4. 1 Hiperparámetros utilizados para el entrenamiento de la primera solución

Épocas	Tasa de aprendizaje	Tamaño del lote de entrenamiento	Modelo base
64	0.001	4	Yolov8-n
128	0.01	8	Yolov8-s
256	0.1	16	Yolov8-l

El segundo modelo por generar utiliza la arquitectura conocida como F-RCNN, se desarrolló por medio de la estructura de nombre "Detectron2", desarrollada por Meta, esta arquitectura destaca por tener una precisión muy elevada sacrificando parte de la velocidad de procesamiento que puede llegar a ofrecer su competencia. De igual manera se realiza un estudio exhaustivo con el fin de obtener la mejor combinación de hiperparámetros posible para la aplicación. En la **Tabla 4. 2** se especifican los valores utilizados para dicho estudio. En este caso las épocas aumentan considerablemente debido a que en los primeros ensayos se determinó que esta arquitectura para funcionar correctamente con los demás hiperparámetros establecidos requiere una cantidad de iteraciones considerablemente mayor para no comprometer la calidad de la precisión del modelo. [54] Como la arquitectura de F-RCNN se diferencia considerablemente con la arquitectura Yolo, se utilizarán de modelos base, modelos que se adaptan de forma correcta a la estructura de estos algoritmos, se utilizan los modelos desarrollados por el equipo desarrollador de *Pytorch.* [55]

Tabla 4. 2 Hiperparámetros utilizados para el entrenamiento de la segunda solución

Épocas	Tasa de aprendizaje	Tamaño del lote de entrenamiento	Modelo base
512	0.001	4	R-50-C4
1024	0.01	8	HRNetV2p-W48
2048	0.1	16	fasterrcnn_x_101

Finalmente, la tercera solución planteada utiliza la arquitectura conocida como SSD, a diferencia del modelo anterior, este destaca por requerir menos épocas durante el entrenamiento para lograr tener resultados satisfactorios, no obstante, existe la limitante del *hardware* utilizado, el cual tiene límites de memoria los cuales pueden ser alcanzados si la combinación de los hiperparámetros utilizados forman una combinación muy demandante computacionalmente. En la **Tabla 4. 3** se observan los hiperparámetros estudiados, nuevamente los modelos base cambian debido a la arquitectura del modelo de clasificación.

Tabla 4. 3 Hiperparámetros utilizados para el entrenamiento de la tercera solución

Épocas	Tasa de aprendizaje	Tamaño del lote de entrenamiento	Modelo base
32	0.001	4	SSD NASNet
64	0.01	8	SSD ResNet
128	0.1	16	SSD512

En los tres modelos distintos de detección de objetos, se define un valor de umbral de precisión mínimo para dar resultados de 30%, esto debido a que posteriormente se requiere implementar una función de rastreo y para que este funcione de manera correcta se requiere que constantemente se detecten objetos a través de los distintos fotogramas, según [56] para aplicaciones que funcionen en tiempo real es óptimo asignar un valor de umbral bajo para evitar fallas constantes debido a ligeros cambios en el ambiente.

4.2 Diseño del rastreo de objetos

Una vez se obtienen los diferentes modelos de IA que se utilizarán para la aplicación deseada, se debe generar una estrategia para que el sistema diseñado sea capaz de reconocer cuando a través de múltiples fotogramas se observa el mismo objeto. Anteriormente en la sección de estado del arte se describieron dos algoritmos de rastreo que fueron contempladas para esta etapa del diseño: SORT y ByteTrack. Ambos se encuentran considerados dentro de los conceptos ganadores de los diferentes filtrados, por lo que se procede a desarrollar una versión para cada uno de estos rastreadores.

El rastreador SORT, funciona a través de vectores de movimiento que intentan predecir cuál será la siguiente posición del objeto y la compara con lo obtenido por la capa de detección, si un automóvil es

detectado y el algoritmo es capaz de definir que éste se está acercando, va a comparar constantemente las posibles posiciones futuras del objeto y compararlas con los resultados que se van obteniendo de forma constante, de esta manera, si la predicción coincide con lo obtenido, se le asigna una identificación al objeto, la cual se mantiene siempre que se detecte el mismo objeto en pantalla, la mayor desventaja de éste método es que si el objeto detectado no se encuentra en varios fotogramas consecutivos, la aplicación reinicia los valores de identificación y entonces descarta que se encuentra rastreando al objeto, por lo tanto, si se vuelve a detectar este objeto, será clasificado como uno completamente nuevo.

Para lograr desarrollar un rastreador SORT, se requiere de las salidas de los algoritmos de detección, principalmente el vector de posición en el que se encuentran los objetos detectados, esta salida se encuentra en formato de una lista de datos en la cual se encuentran dos coordenadas en el eje x y dos coordenadas en eje y, con estas se generan 2 puntos a partir de los cuales se dibujan los cuadros delimitadores que segmentan el objeto de la imagen. A partir de estos 2 puntos y de la posición general de estos en la imagen de entrada, se utiliza la fórmula de distancia euclidiana para aproximar la distancia entre los puntos y a partir de ella, calcular cuál sería su posible posición en el futuro, la ecuación 4.1 describe como se obtiene dicho valor de distancia.

$$Distancia = \sqrt{(x2 - x1)^2 (y2 - y1)^2} \tag{4.1}$$

Donde x1, x2, y1 y y2 se refieren a las coordenadas de los puntos descritos anteriormente.

El algoritmo entonces se encarga de calcular esta distancia entre los puntos de un fotograma y la compara con los puntos del fotograma siguiente, se asigna un valor límite de 30% de precisión entre ambos valores con el fin de tener consistencia con los resultados obtenidos de los algoritmos de detección de objetos, si este valor es similar al de la aproximación calculada para una iteración futura (es decir, hay una varianza menor a 30% entre ambos valores), se define que se está reconociendo el mismo objeto y entonces el identificador se mantiene igual. Para obtener esta predicción de la precisión se utiliza el método de filtros de Kalman. [57]

Para lograr a la ecuación de predicción de estado, el filtro de Kalman debe llevar a cabo un desarrollo matemático que se describe a continuación:

- Modelo de transición del estado:

El filtro de Kalman se basa en la idea de que el estado de un sistema puede ser descrito por un conjunto de variables y que este estado evoluciona con el tiempo. El modelo de transición del estado define cómo el estado evoluciona de un instante de tiempo a otro. Matemáticamente, se puede representar por medio de la ecuación 4.2:

$$x(t) = F \cdot x(t-1) + B \cdot u(t) + w(t) \quad (4.2)$$

Donde:

x(t) es el estado en el tiempo t.

F es la matriz de transición del estado que describe cómo evoluciona el estado.

B es la matriz de control que modela la influencia del control externo u(t) en el estado.

u(t) es el control externo en el tiempo t.

w(t) es el ruido del proceso que representa las incertidumbres y perturbaciones en el modelo de transición.

- Modelo de medición:

El filtro de Kalman también utiliza mediciones para actualizar y refinar la estimación del estado. El modelo de medición define cómo se relacionan las mediciones observadas con el estado del sistema. Matemáticamente, se puede representar mediante la ecuación 4.3:

$$z(t) = H \cdot x(t) + v(t) \quad (4.3)$$

Donde:

z(t) es la medición observada en el tiempo t.

H es la matriz de observación que relaciona el estado con las mediciones observadas.

v(t) es el ruido de medición que representa las incertidumbres y errores en las mediciones.

- Predicción del estado:

El filtro de Kalman realiza una predicción del estado futuro y su incertidumbre utilizando el modelo de transición del estado. La predicción se realiza mediante las ecuaciones 4.4 y 4.5:

$$x_{pred}(t) = F \cdot x(t-1) + B \cdot u(t) \quad (4.4)$$

$$P_{pred}(t) = F \cdot P(t-1) \cdot F^T + Q \quad (4.5)$$

Donde:

$x_{pred}(t)$ es la predicción del estado en el tiempo t.

$P_{pred}(t)$es la matriz de covarianza del estado que representa la incertidumbre de la predicción.

P(t-1) es la matriz de covarianza del estado en el tiempo anterior.

Q es la matriz de covarianza del ruido del proceso.

- Actualización del estado:

  Después de recibir una medición, el filtro de Kalman realiza una actualización del estado y su incertidumbre utilizando el modelo de medición. La actualización se realiza mediante las ecuaciones, la ecuación 4.6, 4.7 y 4.8:

$$K(t) = P_{pred(t)} \cdot H^T \cdot \left(H \cdot P_{pred(t)} \cdot H^T + R\right)^{-1} \cdot P_{pred(t)} \quad (4.6)$$

$$x(t) = x_{pred(t)} + K(t) \cdot \left(z(t) - H \cdot x_{pred(t)}\right) \quad (4.7)$$

$$P(t) = (I - K(t) \cdot H) \cdot P_{pred(t)} \quad (4.8)$$

  Donde:

  K(t) es la matriz de ganancia de Kalman que equilibra la confianza en la predicción y en la medición.

  R es la matriz de covarianza del ruido de medición.

Por otra parte, el rastreador de tipo ByteTrack funciona de una forma distinta, este se enfoca en el funcionamiento de la capa de detección de reconocer objetos, este utiliza patrones detectados en una iteración de imagen de los objetos y los guarda en memoria, luego en la imagen siguiente compara dichos patrones detectados en la imagen anterior y decide si se encuentra dentro de un margen de confiabilidad establecido previamente para definir si en ambas imágenes se observa el mismo objeto. El principal problema de este método es que requiere un mayor peso computacional, para combatir esto, se suele definir un límite de fotogramas en los cuales no se detecta el mismo objeto y cuando este límite es sobrepasado, se borra la información adquirida sobre dicho objeto.

Para asociar los datos obtenidos de la detección y definir si se está rastreando un objeto definido, se utiliza la misma técnica de filtros de Kalman, a partir de esto se realiza una predicción de movimiento similar al rastreador SORT, solo que este se utiliza como un factor a considerar para definir si el mismo objeto del fotograma anterior se encuentra en pantalla, este método termina funcionando como una confirmación de los resultados al comparar los datos en memoria, ambos pasos tienen un peso similar en la toma de decisión para el rastreo de objetos.

El filtro de Kalman utilizado para ByteTrack posee la misma estructura anteriormente descrita mientras que el manejo de patrones por reconocer se toma directamente de la salida de los algoritmos de detección, de estos se toman las coordenadas del objeto detectado, la información de los píxeles en dichas coordenadas y el valor de precisión obtenido, para compararlo con el mínimo límite establecido (nuevamente 30%) para definir el rastreo de forma correcta.

Para ambos rastreadores se define que si a través de los distintos fotogramas, el área abarcada por los cuadros delimitadores aumenta, significa que el objeto detectado se acerca y de manera contraria, si esta área disminuye, significa que el objeto se aleja, a partir de esta información se genera una salida de tipo binaria que define si los objetos detectados se acercan o alejan con respecto al tiempo, con esta salida es posible definir el cálculo para obtener aproximaciones de distancias reales entre el automóvil autónomo y los objetos detectados.

4.3 Cálculo de distancias en las imágenes

Una vez diseñado el sistema para el rastreo de objetos, se deben preparar los datos faltantes para poder proceder a enviar señales a la capa de toma de decisiones del simulador de ECU, gracias a los algoritmos de detección de objetos, se pueden detectar los diferentes obstáculos y definir el espacio que requieren, por otra parte, el rastreo permite definir si a través de múltiples fotogramas, se encuentran los mismos objetos y, a partir de ello, definir si el obstáculo se acerca o se aleja del automóvil autónomo, el siguiente reto es obtener un aproximado de la distancia existente entre el vehículo y el obstáculo, para que el sistema de control sea capaz de actuar de forma adecuada a la realidad que se esté percibiendo.

Los conceptos seleccionados para desarrollar todos coinciden en utilizar el “Método Triangular”, para definir distancias dentro de la imagen, esta decisión se toma a partir de que en el estudio previamente realizado para el filtrado de conceptos, se concluyó que este método es el que aporta menor tiempo de procesamiento al sistema, lo que permite que la capa de toma de decisiones funcione adecuadamente en términos de velocidad, según el estudio realizado en [58], este método es el más apto para buscar eficiencia

computacional entre las opciones investigadas, además este funciona de forma adecuada para la aplicación deseada.

Para poder desarrollar el algoritmo encargado de utilizar el Método Triangular, se necesita de algún tipo de referencia de objetos de tamaño conocido, como los algoritmos de detección diseñados detectan objetos de tipo automóvil, ciclista y peatón, se requiere una referencia de tamaño promedio de cada uno de estos para a partir de esto desarrollar las aproximaciones de distancia. Como referencia, se tomaron fotografías de cada uno de los tres objetos por detectar en escenarios que se conoce la distancia entre la cámara y el objeto, en cada uno de estos casos se realizó una medición de la altura del objeto.

En la Figura 4. 1 se observan las imágenes utilizadas de referencia para la clase de automóvil, se conoce que la altura del modelo de vehículo presente es de 1.5 ± 0.05 metros y se capturaron las imágenes a distancias medidas por una cinta métrica de 0.05 mm de incertidumbre. De la misma manera, en la Figura 4. 2, se presenta la referencia para ciclista la cual tiene una altura de 1.85 ± 0.05 metros y en la Figura 4. 3 para peatón el cual posee una altura de 1.71 ± 0.05 metros, por efectos de privacidad, las referencias fueron censuradas. Como ya se conoce el alto de los objetos se puede hacer una estimación de cuántos píxeles equivalen a una unidad de distancia, de esta forma se obtendría la variable h_{imagen}.

Seguidamente, por medio de semejanza de triángulos, se establece una relación entre h_{imagen}, la altura real del objeto (h_{real}), la distancia real entre la cámara y el objeto (d_{real}) y la distancia obtenida en píxeles de la imagen (d_{imagen}) la cual se aprecia en la ecuación 4.9.

$$\frac{h_{real}}{d_{real}} = \frac{h_{imagen}}{d_{imagen}} \tag{4.9}$$

Se busca que el algoritmo sea capaz de obtener el valor de d_{real} a través de la información obtenida en la imagen, por lo que se puede reacomodar como se aprecia en la ecuación 4.10:

$$d_{real} = \frac{h\_real \cdot d\_imagen}{h\_imagen} \tag{4.10}$$

En las referencias se conoce el valor exacto de d_{real} por lo que se puede corroborar la calibración exitosa de la cámara a utilizar si el porcentaje de error entre el valor obtenido de forma calculada y el real es menor a 5% según se detalla en [58]. Si la calibración es correcta, a partir de la bandera dada por la sección del rastreo de objetos, se inicializa el cálculo de distancia entre el automóvil y los obstáculos y de esta manera se obtiene un valor de distancia, puesto que, la cámara envía fotogramas cada 50 milisegundos, se puede

conocer cuánta distancia se acerca un objeto al vehículo en 50 ms, con esta información es posible aproximar la velocidad con la que este objeto se acerca (cambio de distancia entre fotogramas / tiempo) y enviar esta información al sistema de control para que este se encargue de tomar decisiones de acuerdo a la velocidad del propio vehículo autónomo y a la velocidad con la que se aproxima el obstáculo.

a)

b)

c)

Figura 4. 1 Imágenes de referencia para la clase de automóvil (a) a 5 metros de distancia, b) a 3 metros de distancia y c) a 1 metro)

a)

b)

c)

Figura 4. 2 Imágenes de referencia para la clase de ciclista (a) a 5 metros de distancia, b) a 3 metros de distancia y c) a 1 metro)

a)

b)

c)

Figura 4. 3 Imágenes de referencia para la clase de automóvil (a) a 5 metros de distancia, b) a 3 metros de distancia y c) a 1 metro)

4.4 Generación del Entorno virtual para validación

Debido a la naturaleza del funcionamiento de la aplicación por diseñar, se requiere de un entorno especial para realizar pruebas de validación, la colaboración con NICR permitió el uso del *software* especializado denominado "NI Monodrive" por medio del pago de su respectiva licencia de uso, por lo que se optó por utilizar esta herramienta para abaratar costos del desarrollo del proyecto. Como se describió en el Marco Teórico, este programa se utiliza con el fin de crear entornos realistas y poder probar el comportamiento de los algoritmos a utilizar sin poner vidas ni *hardware* valioso en peligro, además de que permite realizar prácticas de validación y de mejora continua como SIL o HIL también descritas en el Marco Teórico.

Para garantizar un óptimo ambiente de pruebas, se requiere utilizar la herramienta de diseño de escenarios, la cual permite crear todo tipo de ambiente realista por el cual circularía un vehículo. En este editor, el cual se puede apreciar en la Figura 4. 4, se puede simular el uso de un vehículo propio o "*ego vehicle*" que realiza la función de un auto autónomo, el cual seguirá las instrucciones que le envíe el sistema de control que se le llegue a conectar, además se pueden agregar objetos adicionales como otros vehículos, peatones, ciclistas, entre otros. Y se puede dictar el comportamiento de estos, lo que permite realizar ambientes que simulan situaciones específicas que se necesita garantizar el funcionamiento óptimo de los sistemas de seguridad, pero pueden ser pruebas peligrosas de realizar en campo.

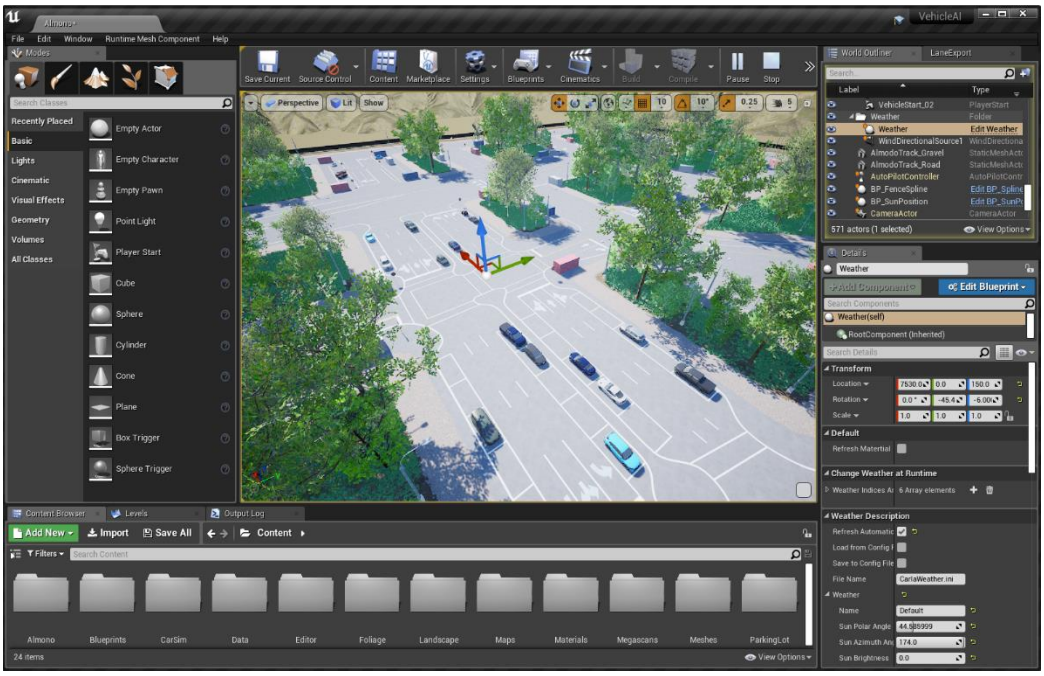


Figura 4. 4 Editor de escenarios de Monodrive

Monodrive también tiene la ventaja de dar acceso a la información captada por los distintos sensores presentes en el auto autónomo, como las cámaras, el tacómetro, radares, lo que permite realizar ensayos más precisos puesto que en todo momento se conocen los factores que afectan al comportamiento mecánico del automóvil.

Con la ayuda de esta herramienta, se preparan tres escenarios claves para la puesta a funcionamiento del simulador de sistema de frenado automático de emergencia, los cuales se pueden visualizar brevemente en el anexo A.4. Los escenarios fueron descritos en la **Tabla 3. 17**y fueron seleccionados debido a que son los escenarios más utilizados por EURO NCAP para las pruebas de algoritmos de frenado de emergencia, se tratan además de un caso sencillo de asimilar (Escenario 1) un escenario moderadamente complejo (Escenario 2) y un escenario complejo (Escenario 3).

Para poder utilizar esta herramienta es necesario realizar una comunicación entre la capa de percepción y la capa de control del simulador de sistema ADAS con respecto al *software* de simulación, puesto que ambas capas requieren de información captada por los sensores del vehículo autónomo, por ende, se desarrolla una comunicación por medio de TCP/IP y otra por medio de UDP con el fin de definir la comunicación más eficaz para el simulador de ECU que se desea desarrollar.

4.5 Integración con capa de control

Una etapa fundamental para el cumplimiento de los objetivos del proyecto es el realizar una comunicación e integración efectiva de la capa de percepción con la capa de control, para ello se consideraron dos protocolos de comunicación que se adaptan de forma más efectiva, TCP/IP y UDP. Estas fueron seleccionadas en vez de las otras investigadas por varias ventajas que ambas poseen a diferencia de las otras opciones que fueron investigadas: Compatibilidad tanto con Python (la herramienta principal utilizada para el desarrollo de los conceptos) como con Monodrive. La comunicación es basada por puertos, lo que permite realizar una programación paralela por medio de procesos que puede llegar a recortar tiempo de operación del algoritmo puesto que como no funcionaría de forma secuencial, el algoritmo no se ve obligado a esperar respuestas del sistema de control para empezar a analizar nuevas entradas. Ambas poseen conectividad por medio de internet y protocolos robustos de seguridad de datos lo que permite un desarrollo continúo en múltiples máquinas sin la necesidad de tomar turnos para utilizar el equipo de cómputo, en una máquina se utiliza el sistema de simulación y la capa de control mientras que en otra máquina se utiliza la capa de percepción.

La capa de control posee las siguientes entradas que son necesarias de parte de la capa de percepción para su correcto funcionamiento:

- Localización en píxeles de los objetos detectados: esta entrada se trata de una lista de datos que conllevan los cuatro puntos que generan las esquinas de los cuadros delimitadores.
- Clase del objeto detectado: la capa de control requiere tomar decisiones distintas según el tipo de objeto que se haya detectado, por ejemplo, si se detecta un peatón, se debe reaccionar de forma más precavida que si se detecta un automóvil.
- Identificador del rastreador y velocidad aproximada: el rastreador de objetos es capaz de calcular la velocidad con la que se acerca el objeto detectado como fue descrito anteriormente, la capa de control requiere asegurarse de que el rastreo del objeto es efectivo (el identificador se mantiene a través de múltiples fotogramas) y necesita el valor de velocidad para tomar una decisión dependiendo del entorno.
- Bandera de peligro: basándose en el tiempo de reacción humana descrita en [61], la capa de percepción es capaz de comparar la distancia entre la cámara y el objeto con la velocidad a la que

se acerca y definir si existe un riesgo de colisión si no se frena lo antes posible, para obtener la distancia de frenado ideal se utiliza el estándar definido en [62] en el cual se describe a cuánta distancia se debe frenar para evitar accidentes a x distancia. Cuando el valor de distancia tiene una diferencia menor a 10% de error entre con respecto a la distancia máxima segura definida por la distancia de frenado ideal, se activa esta bandera de peligro que fuerza a la capa de control a priorizar el frenado antes de cualquier otra función que posea.

Todas estas corresponden a salidas de la capa de percepción y deben de ser comunicadas de forma efectiva a la mayor velocidad posible.

4.6 Optimizaciones

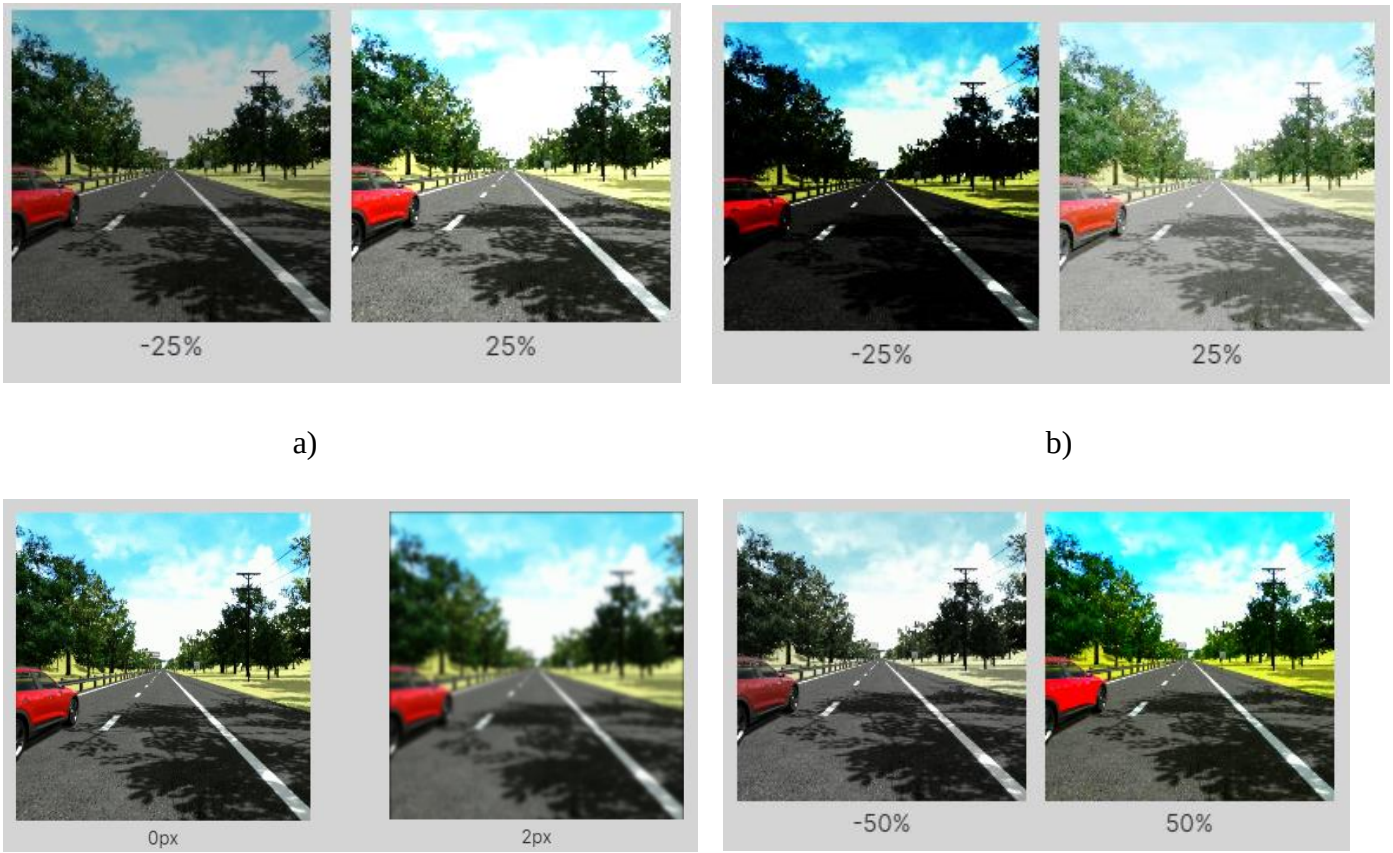
Al haber finalizado las demás etapas descritas a lo largo del capítulo 4, se hallaron múltiples posibles mejoras en las distintas secciones del sistema, estas optimizaciones permiten el generar mejores resultados en las diferentes pruebas de validación y en general, desarrollar mejores diseños para el sistema.

Un problema detectado en la fase inicial de prueba de conceptos es que el porcentaje de precisión no es lo suficientemente alta para que las soluciones sean viables y como efecto secundario, como la detección no es ideal, tanto el rastreo como la aproximación de distancia tiende a fallar. Anteriormente se definieron los hiperparámetros que definen el correcto funcionamiento de aplicaciones de visión artificial, por ende, se genera la estrategia de, además de intentar mejorar los algoritmos de IA por medio de mejoramiento continuo (iterar nuevas versiones constantemente variando los distintos hiperparámetros), se decidió expandir el set de datos a utilizar.

Si bien KITTI es aceptado de forma general para aplicaciones de detección de objetos en la industria de los autos autónomos, no se adapta de forma perfecta a todo tipo de escenarios, por ello se diseñó una ampliación para el set de datos en la cual se utilizan métodos de aumentación de datos, entre ellos se encuentra el modificar múltiples imágenes existentes en el conjunto de datos, para lograr adaptar a los modelos de detección para que funcionen de mejor manera para la aplicación deseada.

En la Figura 4. **5** se aprecian cuatro técnicas utilizadas para realizar la aumentación de datos, la primera corresponde a la exposición a la luz presente en la cámara, la cual puede provocar efectos bastante relevantes para la calidad de la imagen y por ende, afectar al sistema de detección, cambios de brillo los cuales son comunes en ambientes reales, durante el día a día el ambiente posee diferente iluminación y esta

se ve reflejada por medio del brillo en la imagen, seguidamente se agrega ruido a las imágenes con el fin de aumentar la robustez del modelo de IA diseñado ante ambientes complejos o bien que se genera alguna interferencia que afecte la calidad de las imágenes tomadas por la cámara y finalmente cambios de saturación en la imagen, esta última tiene el fin de facilitar que los algoritmos sean capaces de detectar de forma eficiente a pesar de que existan tonos de color similares en toda la imagen.



a) b)

c) d)

Figura 4. 5 Ejemplos de las técnicas de aumentación de datos utilizadas (a) Exposición a la luz, b) Cambios de brillo, c) Agregar ruido en la imagen y d) Saturación)

Al agregar este conjunto de datos extra dentro del original, se notó una mejoría importante en las distintas métricas que se obtuvieron en las medidas iniciales previas al proceso de validación, dentro de ellas se destacan: mejor precisión al detectar objetos, mejor calidad de rastreo a través de múltiples fotogramas, mayor velocidad de procesamiento y mejor calidad de aproximaciones a tiempo real, por lo que esta optimización resulto en un éxito completo en la tarea de mejorar el funcionamiento de los modelos de IA utilizados.

4.7 Proceso de validación

Para el proceso de validación se utilizarán las pruebas de validación detalladas en la sección 3.1, para ello se configura un sistema de pruebas *Software in the Loop* en el cual se configura un sistema a lazo cerrado que consiste en lo siguiente: En el ambiente de simulación se generan las simulación del auto autónomo en carretera, además de brindar las funcionalidades de las cámaras frontales presentes en este, las imágenes captadas por dicha cámara serán los insumos para la aplicación de detección el cuál utilizará el sistema de comunicación implementado para que la capa de control tome las decisiones necesarias para que el vehículo autónomo defina un curso de acción.

A partir de ello, por medio del mismo sistema de comunicación, el sistema de control envía sus señales al entorno de simulación para que este accione. Debido a los cambios en el ambiente debido a estas acciones, la información recibida por los sensores va a cambiar, por lo que se pueden generar entornos de simulación que se retroalimentan de forma indefinida y permiten un análisis a profundidad del sistema puesto en un funcionamiento propenso a cambios.

5 RESULTADOS Y ANÁLISIS

5.1 Análisis de Resultados:

El objetivo de este capítulo es recapitular los resultados obtenidos hasta este punto y hacer comparación de ellos con aquello que de alguna manera valida o bien demuestra lejanía con lo que refleja un adecuado resultado, esta comparación se realiza de distintas maneras y se pretende tanto identificar cómo analizar las potenciales mejoras al diseño de manera que el refinamiento sea posible, mientras que se da cabida al surgimiento de conclusiones y recomendaciones importantes y coherentes con lo que respecta al diseño y al proyecto desarrollado.

5.1.1 Prueba de Validación 1:

Como se describió en la Tabla 3. 15, la primera prueba de validación consiste en poner a prueba los modelos de inteligencia artificial con el set de datos de validación (Equivalente a 20% de las imágenes del conjunto de datos, lo que equivale a 2800 imágenes aproximadamente) con el fin de determinar la precisión promedio de dichos modelos obtenida durante el entrenamiento, para ello se utilizan dos herramientas principales, los gráficos de pérdidas y precisión con respecto a la cantidad de iteraciones, así como las

matrices de confusión que permiten visualizar de manera más efectiva la capacidad de clasificar información de los diferentes modelos, así como apreciar el porcentaje de falsos positivos, las métricas meta a cumplir en este ejercicio es tener gráficas que tengan un comportamiento asintótico a un valor así como tener valores de 80% de precisión en la detección de la clase de peatón y un 90 % para las clases de automóviles y de ciclistas.

Se reitera que cada uno de los modelos entrenados tienen una arquitectura de detección de objetos distinta, la primera solución utiliza Yolov8, la segunda utiliza Detectron2 y la tercera utiliza SSD-NAS, este experimento permite generar un análisis más profundo sobre estas arquitecturas de detección de objetos para la aplicación deseada y con respecto al hardware utilizado.

5.1.1.1 Resultados del estudio exhaustivo de los modelos de inteligencia artificial:

En la **Tabla 5. 1** se aprecian los resultados obtenidos para las mejores combinaciones de hiperparámetros obtenidas en cada una de las soluciones propuestas, este fue un proceso iterativo en el cual luego de probar todas las posibles combinaciones de hiperparámetros propuestos, se seleccionaron los mejores resultados, los cuales serán detallados a continuación:

Tabla 5. 1 Mejores combinaciones de parámetros para los entrenamientos de modelos de IA

	Tasa de entrenamiento	Tamaño del lote de entrenamiento	Iteraciones	Modelo base
Solución 1 (Yolov8)	0.01	16	256	Yolov8n
Solución 2 (Detectron2)	0.001	4	1024	fasterrcnn_x_101
Solución 3 (SSD-NAS)	0.001	4	64	SSD_NAS

5.1.1.2 Resultados del entrenamiento del primer prototipo (Yolov8):

En la Figura 5. 1, la Figura 5. 2 y la Figura 5. 3 se aprecian las herramientas que se utilizan en la literatura respectiva de inteligencia artificial para evaluar el entrenamiento de un modelo de detección de objetos, las gráficas poseen un comportamiento cercano al deseado, en el cual se ve un comportamiento asintótico al final, lo que significa que la cantidad de iteraciones seleccionada es adecuada. Además, se nota en la matriz de confusión de la Figura 5. 3 el cómo se cumplen con las métricas, obteniendo 82% en la clase

de peatones, 90 y 96 por ciento en ciclistas y automóviles respectivamente, por lo que esta solución es viable en cuanto a precisión durante el entrenamiento se refiere.

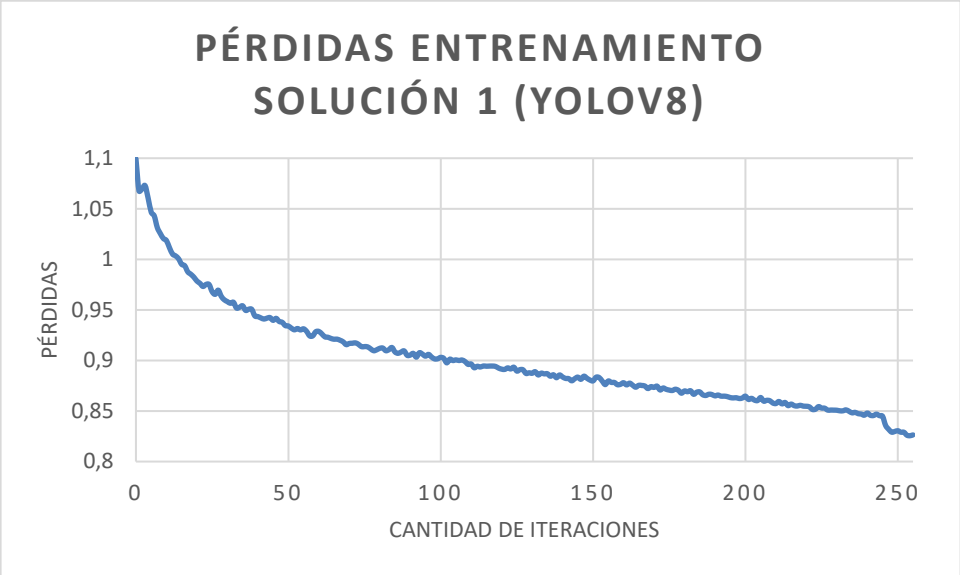


Figura 5. 1 Pérdidas con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la primera solución.

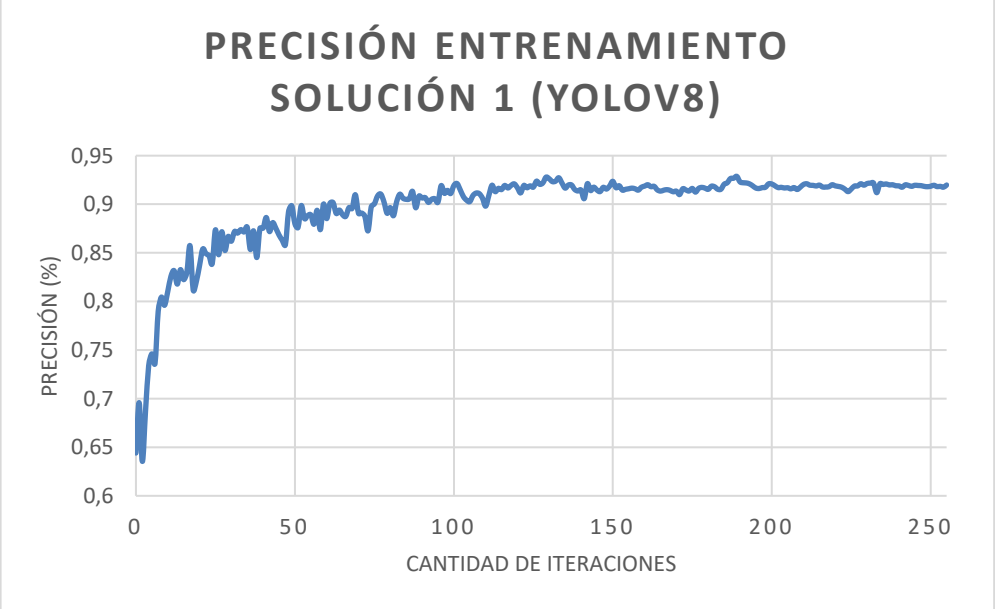


Figura 5. 2 Precisión con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la primera solución.

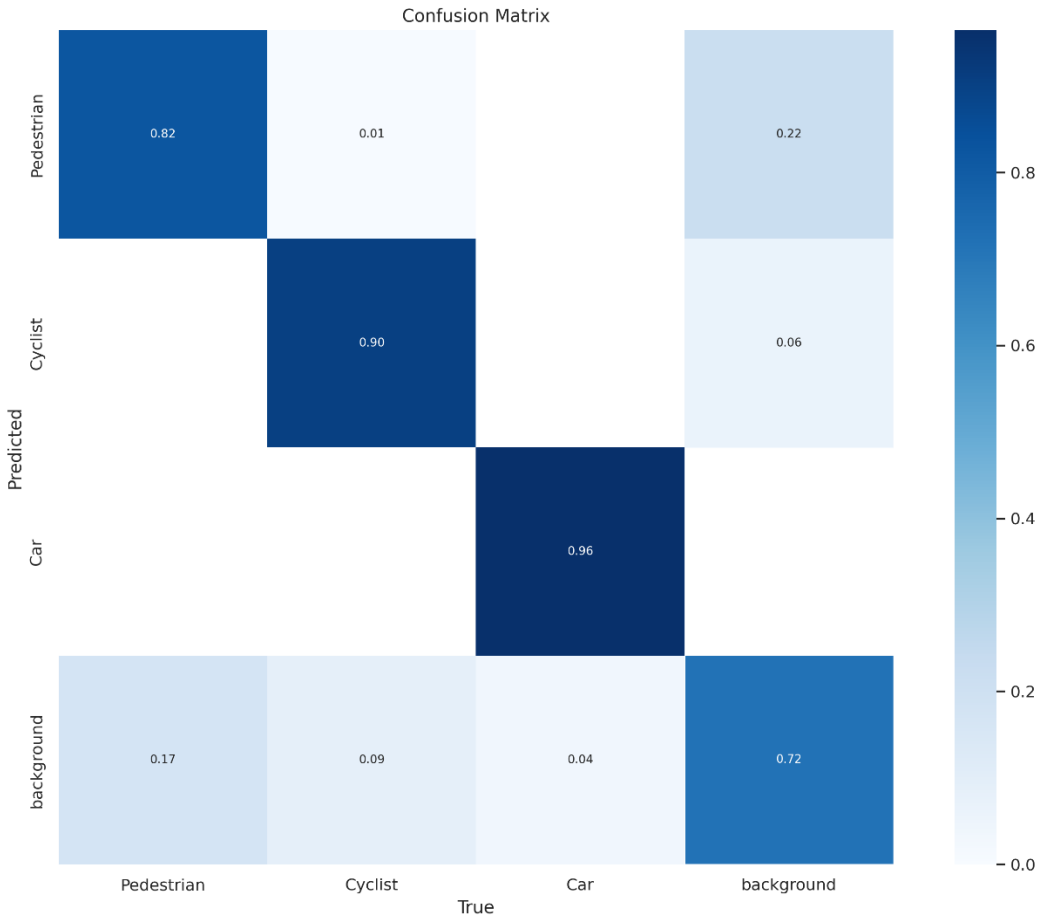


Figura 5. 3 Matriz de confusión para el entrenamiento de la mejor combinación de hiperparámetros para la primera solución.

5.1.1.3 Resultados del entrenamiento del segundo prototipo (Detectron2):

Seguidamente, la Figura 5. 4, la Figura 5. 5 y la Figura 5. 6 muestran los resultados obtenidos para la evaluación de la segunda solución, nuevamente la cantidad de épocas seleccionada parece ser adecuada debido a la forma de las gráficas tanto de pérdidas como de precisión, por otra parte, la matriz de confusión permite confirmar que, en términos de precisión en las distintas clases, cumple con las métricas establecidas, obteniendo un 96 % de precisión para automóviles, 94% para ciclistas y 87% para peatones

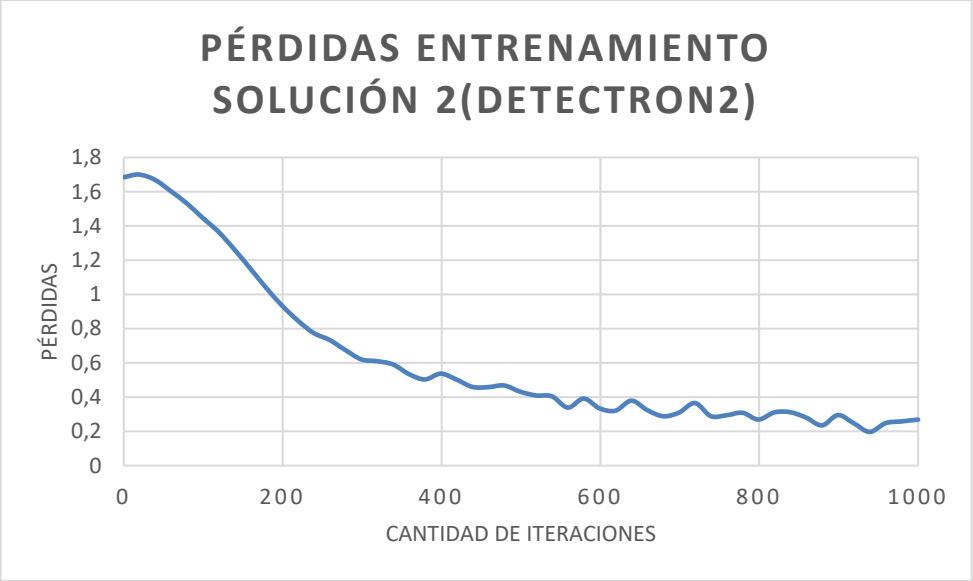


Figura 5. 4 Pérdidas con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la segunda solución.

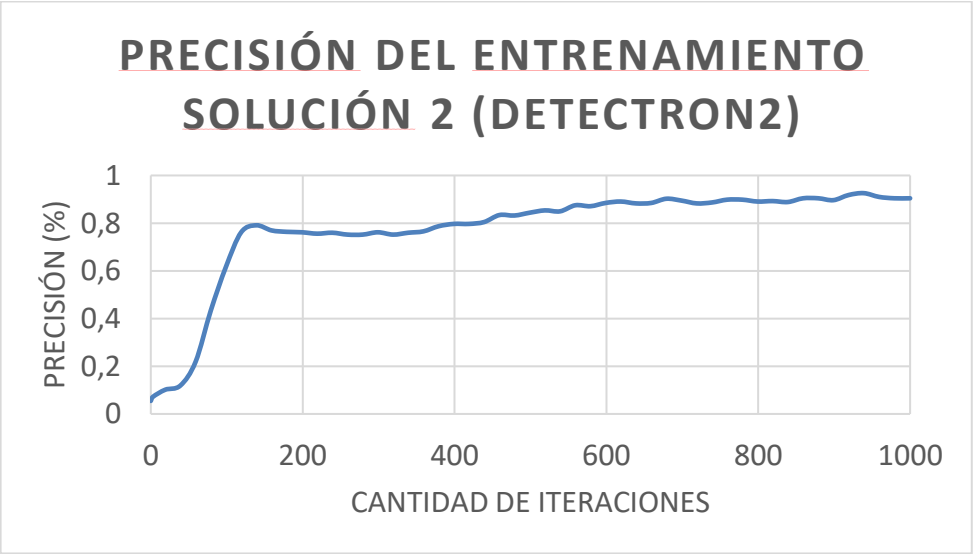


Figura 5. 5 Precisión con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la segunda solución.

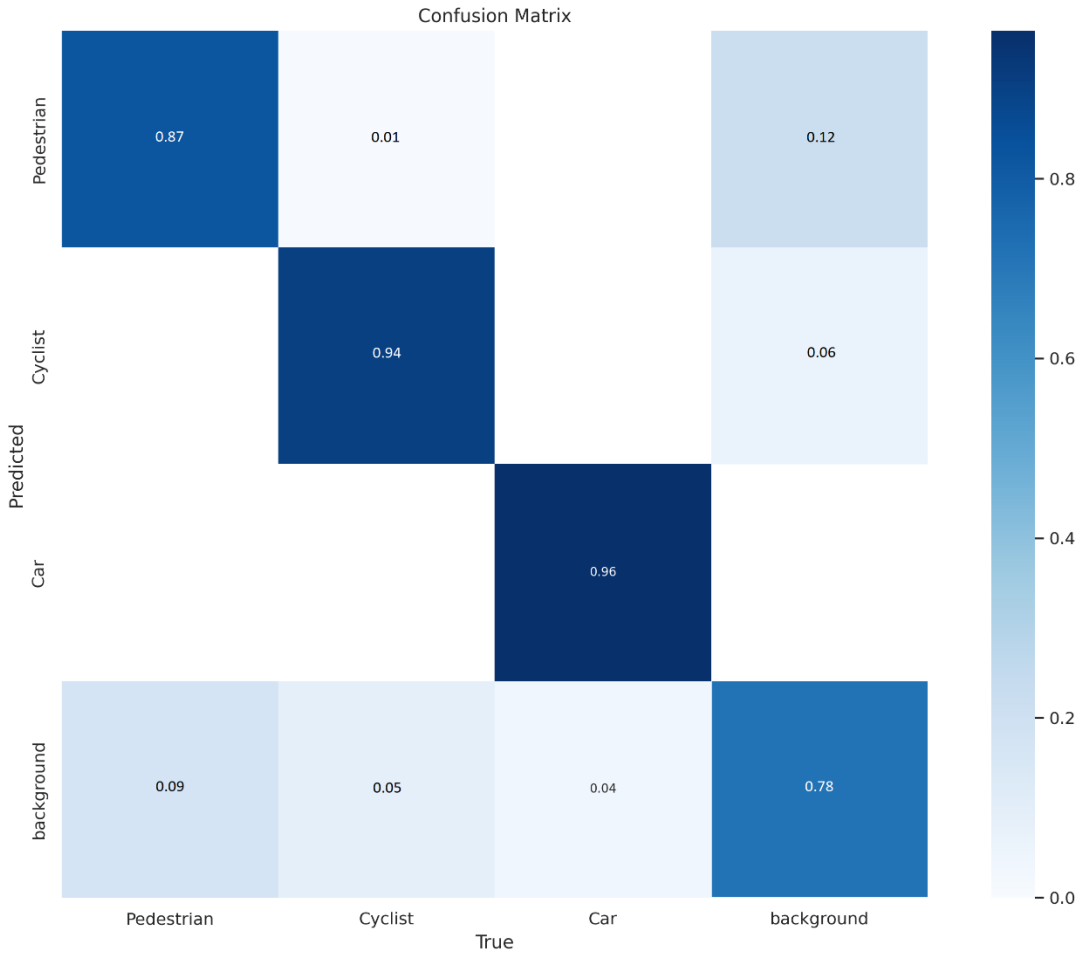


Figura 5. 6 Matriz de confusión para el entrenamiento de la mejor combinación de hiperparámetros para la segunda solución.

5.1.1.4 Resultados del entrenamiento del tercer prototipo (SSD-NAS):

Finalmente, la tercera solución presenta sus resultados en la Figura 5. 7, la Figura 5. 8 y la Figura 5. 9, en este caso las gráficas de pérdidas y precisión destacan un comportamiento que sugiere que deberían utilizarse más iteraciones durante este entrenamiento, no obstante, debido a limitaciones de *hardware*, todas las combinaciones de hiperparámetros que utilizaban mayor cantidad de memoria que la combinación seleccionada llegaban a fallar debido a limitaciones de poder computacional. Añadido a esto, se aprecia en la matriz de confusión consecuencias de este problema, puesto que la precisión de la detección de peatones no alcanza las métricas deseadas, llegando únicamente a 74 por ciento.

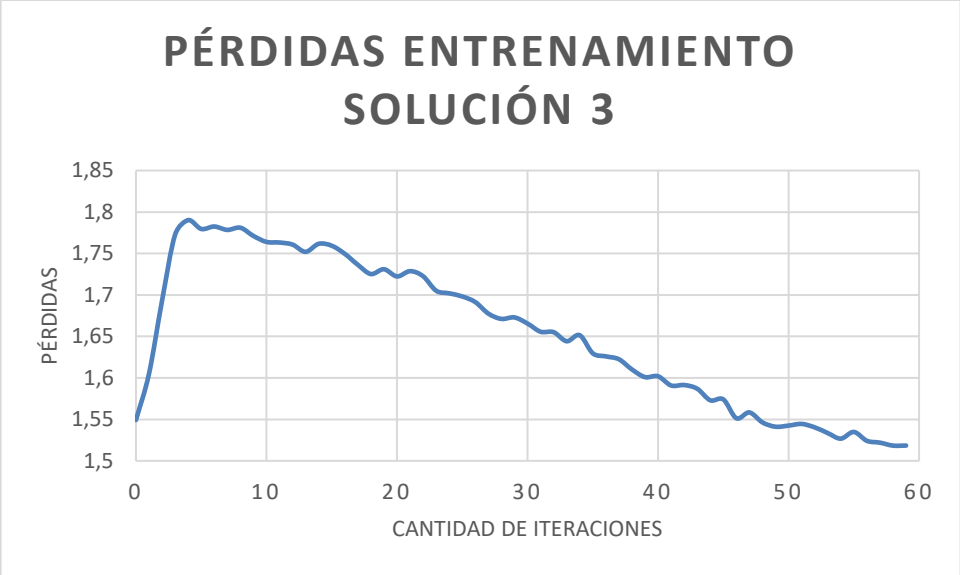


Figura 5. 7 Pérdidas con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la tercera solución.

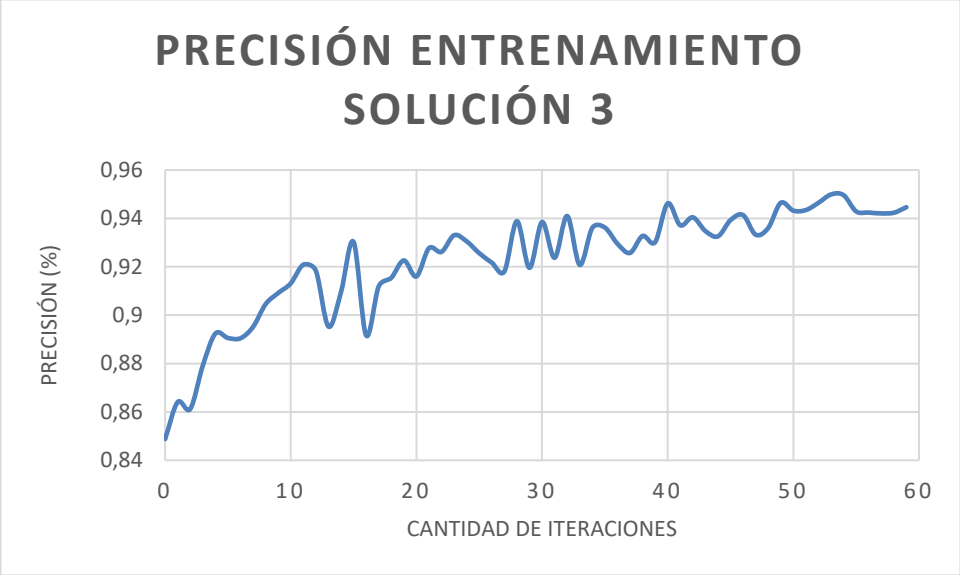


Figura 5. 8 Figura 16. Precisión con respecto al número de iteraciones para el entrenamiento de la mejor combinación de hiperparámetros para la tercera solución.

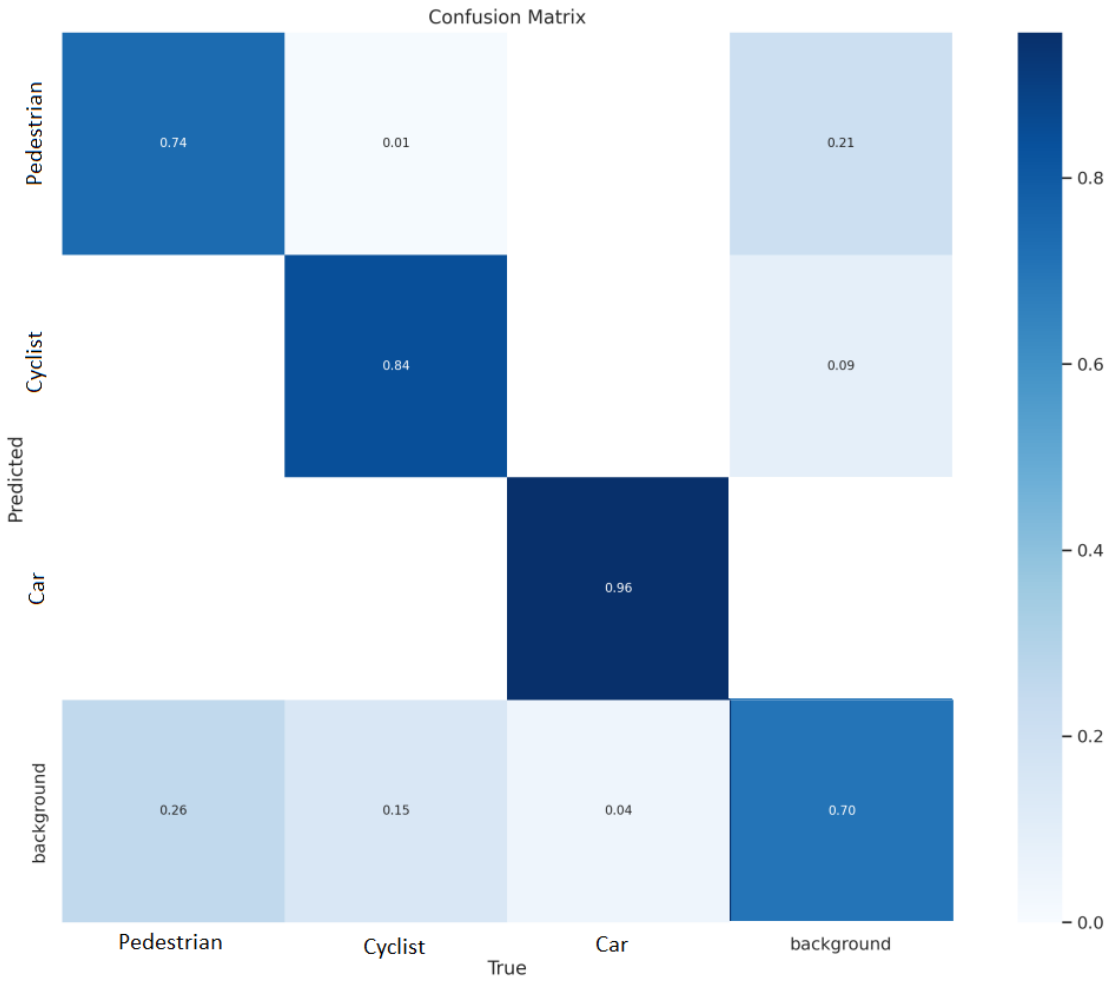


Figura 5. 9 Matriz de confusión para el entrenamiento de la mejor combinación de hiperparámetros para la tercera solución.

5.1.1.5 Resumen de los resultados del experimento 1

Finalmente, la **Tabla 5. 2** denota en forma de resumen los resultados obtenidos por medio de esta primera prueba de validación, donde se aprecia que la única solución incapaz de cumplir con todas las métricas deseadas es SSD-NAS, debido a que esta demostró tener un porcentaje de precisión menor para la clase de peatones, por lo que es la única solución planteada que no supera las métricas establecidas. La segunda solución demuestra ser la más apta en términos de precisión general con respecto a su competencia

Tabla 5. 2 Resumen de resultados del análisis de precisión durante el entrenamiento de los modelos de IA diseñados

Solución	Automóviles (%)	Peatones (%)	Ciclistas (%)
Métricas referencia	90	80	90
Yolov8	94	87	96
Detectron2	96	87	94
SSD-NAS	84	74	96

5.1.2 Prueba de validación 2

La segunda prueba de validación consiste en evaluar las tres soluciones desarrolladas en ambientes distintos, el set de datos KITTI permite el uso de sets menores divididos en tres dificultades: imágenes sencillas, imágenes moderadamente complejas e imágenes complejas. En la Figura 5. 10, la Figura 5. 11 y la Figura 5. 12 se encuentran ejemplos de imágenes tomadas de cada uno de estos subconjuntos de datos.

Figura 5. 10 Ejemplo de imagen sencilla

Figura 5. 11 Ejemplo de imagen moderadamente compleja

Figura 5. 12 Ejemplo de imagen compleja

Cada uno de los algoritmos diseñados será evaluado utilizando los tres algoritmos de pruebas y serán comparados con las métricas dadas por Yolov5-ADAS denotadas en la **Tabla 3. 12**, esas definen las métricas a superar puesto que son las métricas alcanzadas por un modelo que se utiliza en la industria de la conducción autónoma. Se evaluará principalmente por medio de matrices de confusión obtenidas al correr los algoritmos en modo de validación

5.1.2.1 Resultados del primer prototipo:

La Figura 5. 13, la Figura 5. 14 y la Figura 5. 15 representan los resultados obtenidos en la primera solución (Yolov8) tras el proceso de validación con los diferentes subconjuntos de datos

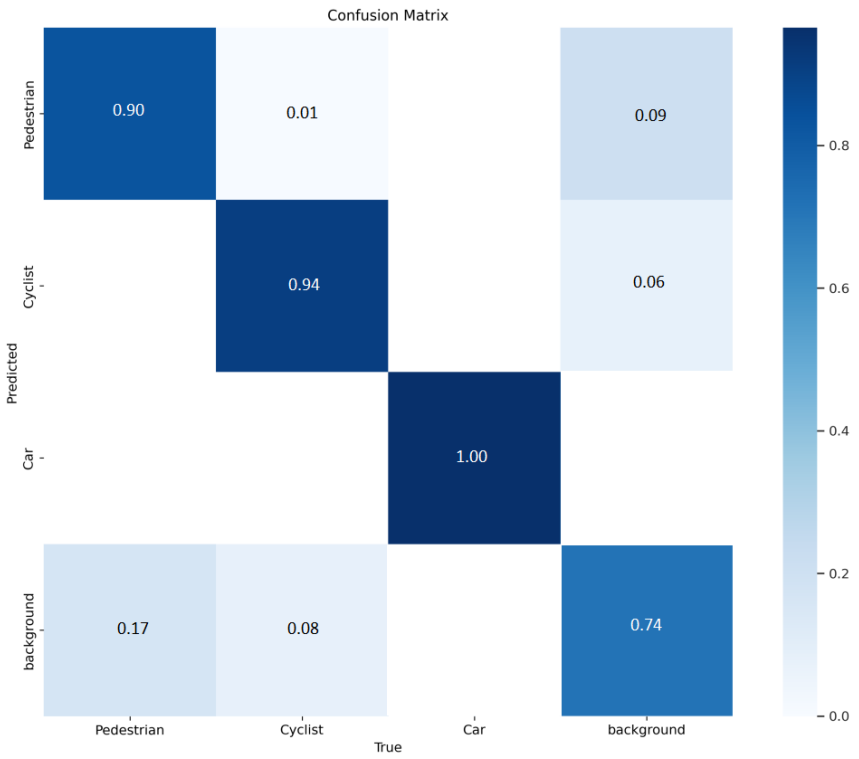


Figura 5. 13 Matriz de confusión para la primera solución utilizando el conjunto de datos de imágenes sencillas

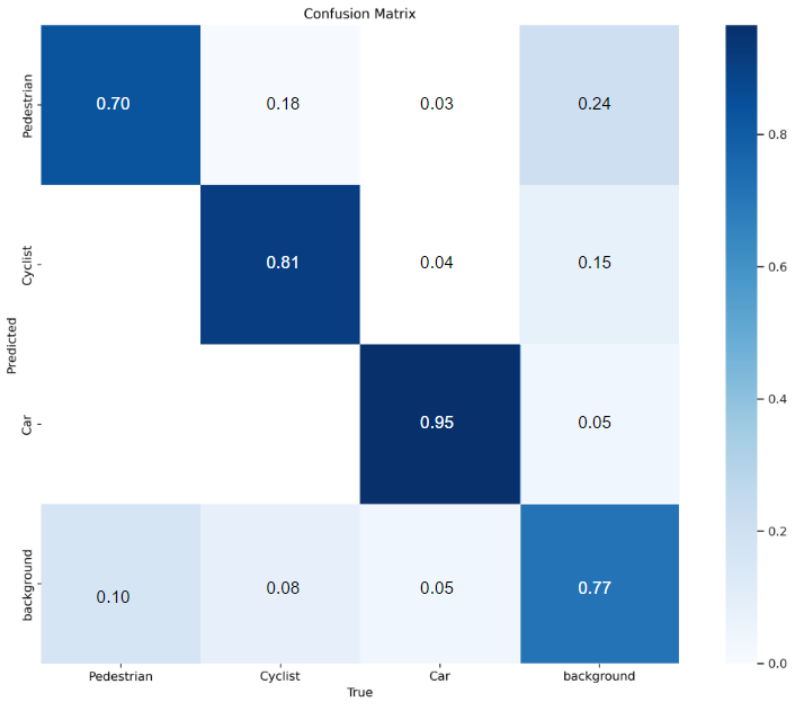


Figura 5. 14 Matriz de confusión para la primera solución utilizando el conjunto de datos de imágenes moderadamente complejas.

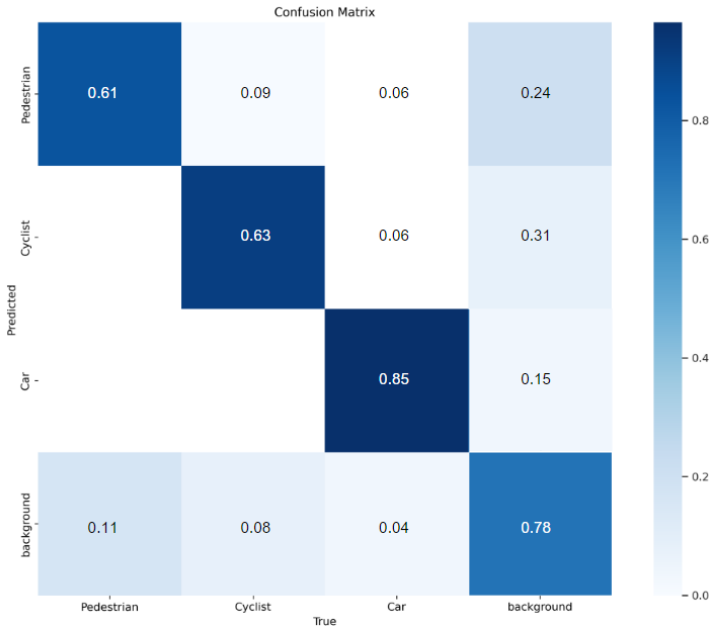


Figura 5. 15 Matriz de confusión para la primera solución utilizando el conjunto de datos de imágenes complejas.

5.1.2.2 Resultados del segundo prototipo:

La Figura 5. 16, la Figura 5. 17 y la Figura 5. 18 representan los resultados obtenidos en la segunda solución (Detectron2) tras el proceso de validación con los diferentes subconjuntos de datos

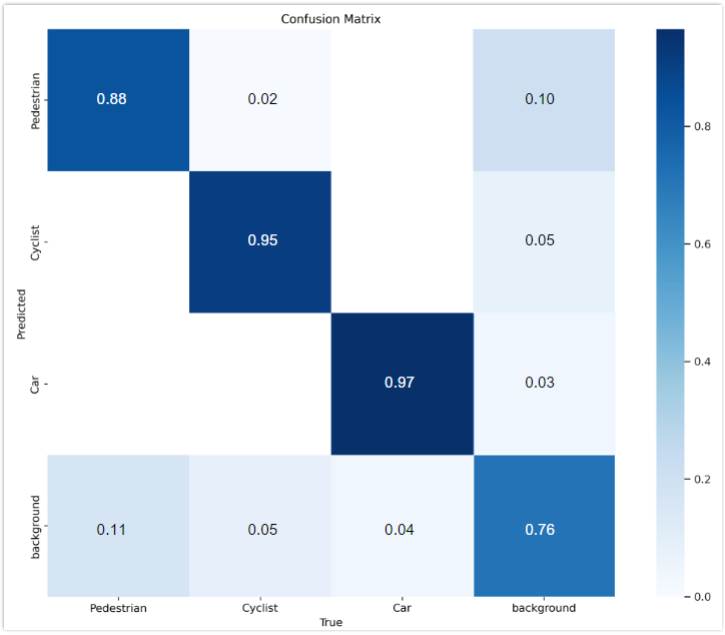


Figura 5. 16 Matriz de confusión para la segunda solución utilizando el conjunto de datos de imágenes sencillas

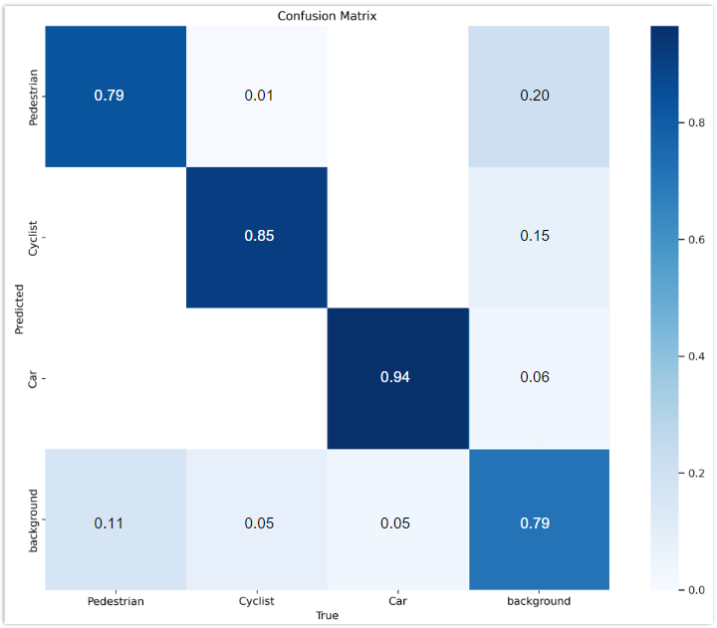


Figura 5. 17 Matriz de confusión para la segunda solución utilizando el conjunto de datos de imágenes moderadamente complejas

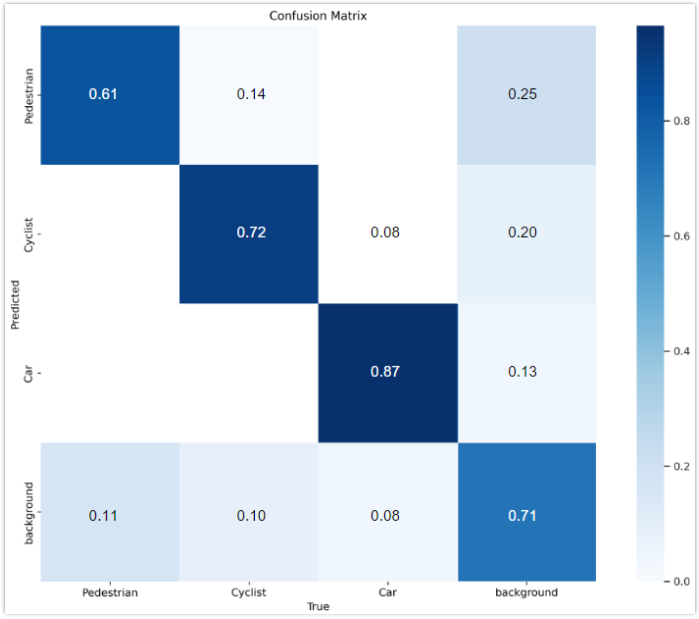


Figura 5. 18 Matriz de confusión para la segunda solución utilizando el conjunto de datos de imágenes complejas

5.1.2.3 Resultados del tercer prototipo:

La Figura 5. 19, la Figura 5. 20 y la Figura 5. 21, representan los resultados obtenidos en la tercera solución (SSD-NAS) tras el proceso de validación con los diferentes subconjuntos de datos.

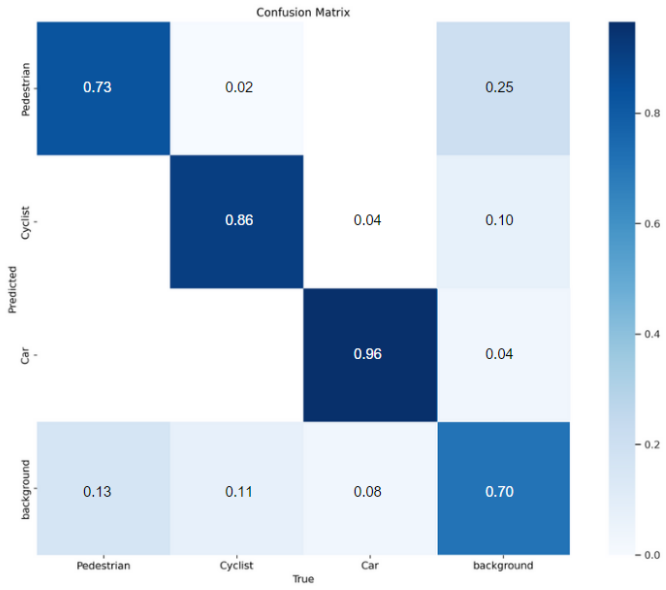


Figura 5. 19 Matriz de confusión para la tercera solución utilizando el conjunto de datos de imágenes sencillas

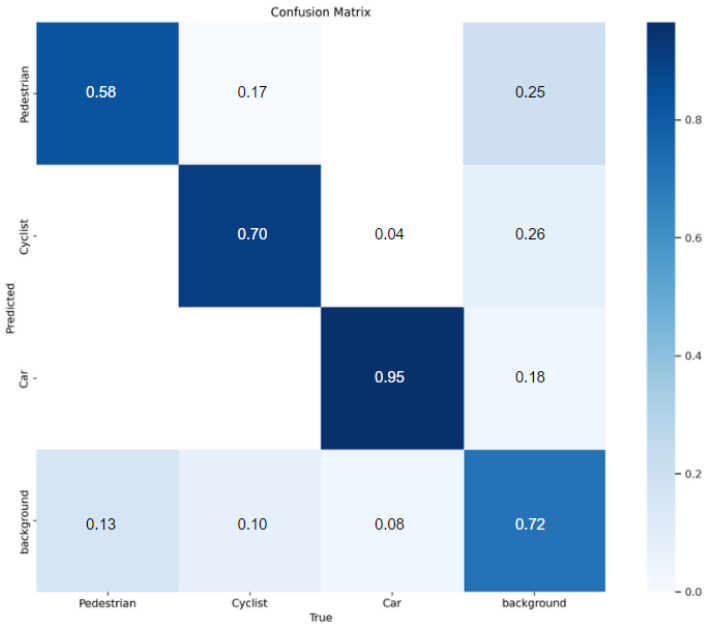


Figura 5. 20 Matriz de confusión para la tercera solución utilizando el subconjunto de datos de imágenes moderadamente complejas

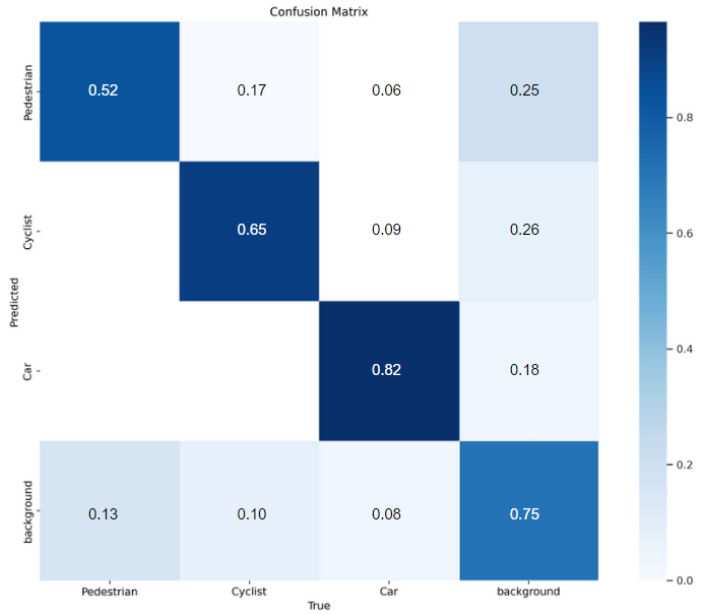


Figura 5. 21 Matriz de confusión para la tercera solución utilizando el subconjunto de datos de imágenes complejas

5.1.2.4 Resumen de resultados de la prueba de validación:

Luego de realizar todos los experimentos necesarios, se reúnen los datos relevantes en la tabla resumen comprendida en la **Tabla 5. 3**, se denota que la única solución que no llega a cumplir con todas las métricas establecidas es la solución de SSD-NAS, el cual falla en la precisión de imágenes sencillas y de imágenes complejas para la clase de automóvil, debido a que ambos casos fallan, en el promedio general se puede apreciar que no se cumple la métrica para esta clase, no obstante, todos los demás casos son capaces de superar las métricas esperadas.

Tabla 5. 3 Resumen de resultados del análisis de precisión en sets de datos de distintas dificultades.

Solución	Referencia	Imagen Sencilla (%)	Imagen Moderada (%)	Imagen Compleja (%)	Promedio
YOLOv8	Automóvil	99.63	94.93	84.82	93.13
	Ciclista	94.24	81.26	63.14	74.54
	Peatón	90.02	69.86	61.32	73.73
Detectron2	Automóvil	97.34	94.15	87.13	92.87
	Ciclista	94.95	84.68	72.16	83.93
	Peatón	88.31	79.26	60.87	76.15
SSD NAS	Automóvil	95.90	95.31	81.95	91.05
	Ciclista	85.84	69.64	65.11	73.53
	Peatón	72.65	57.91	51.92	60.83

5.1.3 Prueba de validación 3

La tercera prueba de validación consiste en verificar que se cumple con una velocidad de procesamiento de imágenes menor a 50 milisegundos, para la prueba se desarrollaron 3 escenarios en el simulador de Monodrive en los cuales se pone a prueba el comportamiento del sistema diseñado a través de múltiples situaciones las cuales fueron brevemente descritas en la descripción de la prueba que se encuentra en la **Tabla 3. 17**. En el anexo A.3 se encuentran breves videos que ejemplifican los distintos escenarios.

Las tres soluciones desarrolladas se pusieron a prueba frente a los 3 escenarios, cada uno tiene 100, 60 y 150 fotogramas respectivamente se ponen a correr y se registran los resultados de tiempo de procesamiento para cada fotograma, inicialmente se realiza una corrida en el escenario 1 para caracterizar el sistema de

detección con respecto a la desviación estándar en general de todo el experimento, destacando además el tiempo máximo alcanzado por alguna iteración. A partir de esta corrida se utiliza el teorema 9.2 presente en [59] en el cual se dicta cuántos experimentos se deben realizar para tener un 95% de confianza y que, a su vez no exceda un error mayor al 10% (Este error se definió anteriormente que es suficiente para evitar problemas de comunicación debido al tiempo que demora el algoritmo en detectar y rastrear objetos). Como se explica en la literatura, al no conocer la varianza de la población de datos, se puede utilizar la desviación de la muestra.

"En términos estrictos, la fórmula del teorema 9.2 solo será aplicable si se conoce la varianza de la población de la cual se seleccionó la muestra. Si no contamos con esa información, podríamos tomar una muestra preliminar de tamaño n ≥ 30 para proporcionar una estimación de σ. Después, usando S como aproximación para σ (desviación estándar) en el teorema 9.2, podemos determinar aproximadamente cuantas observaciones necesitamos para brindar el grado de precisión deseado." [59]

Para estas pruebas de validación, se tiene que tomar en cuenta todas las variaciones que se le han realizado a los distintos prototipos, por ende, se realizan mediciones para cada uno de los tres modelos de IA para la detección de objetos, así como se pone a prueba los dos sistemas de comunicación para la primera solución.

En el anexo A4 se encuentran los resultados del experimento inicial para cada uno de los prototipos, la primera prueba realizada para el primer prototipo tiene una desviación estándar de 0.09261 y un tiempo promedio de 17.9822 ms por lo que se puede utilizar, según el teorema 9.2, $Z_{\frac{a}{2}}$ es un valor estadístico que se define en [59], para un valor de confianza del 95 %, su valor es de 1.96, y e es el error que se desea no exceder, el cual es 0.10 para un valor de 10%. En la ecuación 5.1 se demuestra el cálculo de la cantidad de experimentos que se requieren para obtener una muestra con los hiperparámetros deseados. De la misma manera, para los prototipos 2 y 3 se obtiene el valor de muestras ideal para cumplir con dichos hiperparámetros a partir de sus valores de desviación estándar (0.1634 y 0.1172 respectivamente) los cuales se desarrollan en las ecuaciones 5.2 y 5.3.

$$n_1 = \left(\frac{\frac{Z_{\frac{a}{2}} \cdot \sigma}{2}}{e}\right)^2 = \left(\frac{1.96 \cdot 0.09261}{0.10}\right)^2 = 3.29 \quad (5.1)$$

$$n_2 = \left(\frac{1.96 \cdot 0.1634}{0.10}\right)^2 = 10.26 \quad (5.2)$$

$$n_3 = (\frac{1.96 \cdot 0.1172}{0.10})^2 = 5.27 \quad (5.3)$$

Según estos resultados, se realizaron 3 experimentos para el prototipo número 1, 10 experimentos para el segundo prototipo y finalmente 5 experimentos para el tercer prototipo

5.1.3.1 Resultados del primer escenario en el primer prototipo:

El primer prototipo tiene múltiples pruebas a tomar en cuenta, la primera se realiza utilizando de entrada un vídeo guardado de manera local en el ordenador en el que se utiliza el sistema diseñado (**Tabla 5. 4**), luego de realizar el mismo experimento con una entrada de imágenes generadas por el simulador Monodrive, pero utilizando la comunicación por medio de UDP (**Tabla 5. 5**) y luego por TCP/IP (**Tabla 5. 6**).

Tabla 5. 4 Resultados del primer escenario para el prototipo 1, utilizando una entrada de vídeo local

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	15.7551	0.8542	35.6428
2	16.4110	0.9194	35.4185
3	18.1845	1.0092	36.4962
Promedio	16.7835		

Tabla 5. 5 Resultados del primer escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por UDP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	95.6233	4.2335	130.5616
2	94.9453	4.5434	129.6512
3	97.8372	4.2652	131.2845
Promedio	96.1353		

Tabla 5. 6 Resultados del primer escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx.(ms)
1	19.2991	1.4584	26.5612
2	19.5708	1.5435	32.5150
3	19.5263	1.4594	26.0005
Promedio	19.4654		

5.1.3.2 Resultados del primer escenario en el segundo prototipo:

El segundo prototipo, como no tiene compatibilidad con UDP, se reducen las variantes para realizar los experimentos, únicamente se tienen las entradas en vídeo local (**Tabla 5. 7**) y por medio de comunicación TCP/IP (**Tabla 5. 8**), se reitera que para estas pruebas se realizan 10 experimentos debido a los resultados del teorema 9.2 [59]

Tabla 5. 7 Resultados del primer escenario para el prototipo 2, utilizando una entrada de vídeo local

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	101.0597	0.1634	259.2430
2	102.0451	0.1766	271.4977
3	102.9034	0.1745	263.8332
4	101.9904	0.1698	263.8643
5	101.6829	0.1726	260.4568
6	101.7735	0.1702	266.2801
7	101.9832	0.1733	262.9072
8	102.3327	0.1709	267.2029
9	102.4896	0.1721	267.3204
10	101.9108	0.1713	264.9381
Promedio	101.5524		

Tabla 5. 8 Resultados del primer escenario para el prototipo 2, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	118.8490	0.1945	303.5302
2	119.7103	0.2176	314.6020
3	120.0172	0.2125	311.2895
4	119.2024	0.2068	310.4028
5	120.1921	0.2076	307.7895
6	119.9485	0.2091	310.0852
7	118.9128	0.2079	312.6920
8	120.0367	0.2086	311.7324
9	119.3266	0.2094	308.9701
10	119.5293	0.2072	310.6717
Promedio	119,5724		

5.1.3.3 Resultados del primer escenario en el tercer prototipo:

De forma similar al prototipo 2, el prototipo 3 tiene únicamente 2 casos distintos (**Tabla 5. 9** y **Tabla 5. 10**) a los cuales se detallan los resultados a continuación:

Tabla 5. 9 Resultados del primer escenario para el prototipo 3, utilizando una entrada de vídeo local

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	4.0166	0.1029	6.8116
2	4.5263	0.1178	7.0796
3	4.5735	0.1213	8.6160
4	4.3156	0.1065	7.0941
5	4.4912	0.1157	8.1302
Promedio	4.3846		

Tabla 5. 10 Resultados del primer escenario para el prototipo 3, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	7.5305	0.0989	10.9992
2	7.5319	0.1023	10.9980
3	7.6926	0.1284	10.9985
4	7.5162	0.1151	10.9951
5	7.6138	0.0996	11.0017
Promedio	7.5770		

5.1.3.4 Resultados del segundo escenario en el primer prototipo:

Tabla 5. 11 Resultados del segundo escenario para el prototipo 1, utilizando una entrada de vídeo local

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	10.4691	0.5838	36.5517
2	10.2867	0.6026	38.2926
3	10.1323	0.6023	37.4284
Promedio	10.2960		

Tabla 5. 12 Resultados del segundo escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por UDP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	89.1238	0.5334	121.9422
2	88.1209	0.5454	118.1652
3	90.4539	0.5574	120.6153
Promedio	89.2329		

Tabla 5. 13 Resultados del segundo escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	19.5837	0.0154	25.0518
2	19.9870	0.0156	23.9990
3	19.4398	0.0177	26.1049
Promedio	19.6702		

5.1.3.5 Resultados del segundo escenario en el segundo prototipo:

Tabla 5. 14 Resultados del segundo escenario para el prototipo 2, utilizando una entrada de vídeo local

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	103.3412	0.1554	263.5314
2	103.6560	0.1576	268.0590
3	103.2681	0.1554	261.6160
4	103.3693	0.1559	263.4382
5	103.5067	0.1560	265.9756
6	103.3801	0.1562	262.8217
7	103.4462	0.1558	265.9075
8	103.4037	0.1562	265.0634
9	103.4398	0.1559	263.8610
10	103.4525	0.1558	265.3353
Promedio	103.4264		

Tabla 5. 15 Resultados del segundo escenario para el prototipo 2, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	119.1859	0.2057	298.7754
2	117.3480	0.1935	304.8083
3	118.2569	0.1890	302.2686
4	118.6564	0.1943	302.3991
5	119.3487	0.1972	301.9075
6	119.0882	0.1969	301.8622
7	118.8749	0.1963	302.0437
8	118.9715	0.1959	303.6069
9	118.8548	0.1958	301.1787
10	118.9926	0.1970	301.3633
Promedio	118.7578		

5.1.3.6 Resultados del segundo escenario en el tercer prototipo:

Tabla 5. 16 Resultados del segundo escenario para el prototipo 3, utilizando una entrada de vídeo local

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	4.5311	0.1313	9.1510
2	4.5072	0.1206	7.6888
3	4.6586	0.1264	9.4748
4	4.5326	0.1235	9.1262
5	4.6176	0.1292	8.9852
Promedio	4.5574		

Tabla 5. 17 Resultados del segundo escenario para el prototipo 3, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	7.5923	0.0936	12.0012
2	7.4565	0.0953	10.0014
3	7.6242	0.1049	10.9992
4	7.5732	0.0940	11.0039
5	7.4349	0.1057	10.9938
Promedio	7.5616		

5.1.3.7 Resultados del tercer escenario en el primer prototipo:

Tabla 5. 18 Resultados del tercer escenario para el prototipo 1, utilizando una entrada de vídeo local

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	9.9238	0.054437	28.6396
2	9.2962	0.046432	29.6100
3	9.6067	0.047548	28.3999
Promedio	9.6089		

Tabla 5. 19 Resultados del tercer escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por UDP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	97.4572	0.06234	130.5150
2	99.2321	0.06435	128.5651
3	98.1034	0.06367	131.1523
Promedio	98.5976		

Tabla 5. 20 Resultados del tercer escenario para el prototipo 1, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	15.3459	0.01032	18.9970
2	15.5269	0.01364	19.0020
3	15.3419	0.01254	18.9986
Promedio	15.4049		

5.1.3.8 Resultados del tercer escenario en el segundo prototipo:

Tabla 5. 21 Resultados del tercer escenario para el prototipo 2, utilizando una entrada de vídeo local

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	104.2577	0.2183	259.7539
2	104.8595	0.2196	262.2814
3	105.1704	0.2235	263.0031
4	104.6233	0.2253	262.4701
5	104.9507	0.2214	259.7313
6	104.8229	0.2206	260.7582
7	104.7491	0.2256	262.2564
8	104.8422	0.2194	261.2175
9	104.7278	0.2249	262.9481
10	104.7956	0.2115	260.8587
Promedio	104.7799		

Tabla 5. 22 Resultados del tercer escenario para el prototipo 2, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	122.6643	0.2644	308.4743
2	122.2185	0.2595	303.7220
3	121.4295	0.2537	302.0016
4	121.8702	0.2589	303.6541
5	122.4827	0.2601	302.4912
6	121.9536	0.2594	307.8356
7	122.0669	0.2586	307.6283
8	121.9542	0.2598	303.0194
9	122.4087	0.2595	309.1268
10	121.7368	0.2631	304.2107
Promedio	122.0785		

5.1.3.9 Resultados del tercer escenario en el tercer prototipo:

Tabla 5. 23 Resultados del tercer escenario para el prototipo 3, utilizando una entrada de vídeo local

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	4.8076	0.1117	7.1201
2	4.7993	0.1304	8.8450
3	4.8868	0.1216	8.4212
4	4.8220	0.1175	8.6474
5	4.7999	0.1244	7.9832
Promedio	4.8231		

Tabla 5. 24 Resultados del tercer escenario para el prototipo 3, utilizando una entrada de vídeo generada por Monodrive y una comunicación por TCP/IP

Experimento	Tiempo Promedio (ms)	Desv. Est. (ms)	Tiempo Máx. (ms)
1	7.6633	0.09244	10.9999
2	7.6026	0.09674	10.0009
3	7.7588	0.10487	11.0021
4	7.7489	0.0943	10.9924
5	7.6173	0.1034	11.0035
Promedio	7.6782		

5.1.3.10 Resumen de los resultados del experimento:

En la **Tabla 5. 25**, se denota en forma de resumen los resultados obtenidos a través de todos los experimentos realizados que se detallaron anteriormente, de esta manera es capaz de descartar las soluciones que no son viables según la métrica de tiempo de procesamiento, además se puede observar como el utilizar la comunicación de tipo UDP se aumenta considerablemente el tiempo de procesamiento, por lo que se descarta del grupo de soluciones viables, por otra parte, el segundo prototipo, debido a la arquitectura del modelo utilizado (F-RCNN), tiende a tener un tiempo de procesamiento significativamente mayor a los demás, por lo que no llega a cumplir con los hiperparámetros deseados.

Por otra parte, tanto el primer modelo utilizando TCP/IP y el tercer prototipo cumplen con el valor de velocidad de procesamiento deseado por un margen significativo, inclusive en las ocasiones que definieron el tiempo máximo en los experimentos realizados, se encuentran dentro del margen de error (10 por ciento de los 50 ms planteados), por lo que se definen soluciones viables.

Además, se denota que al enviar datos por medio de una conexión de tipo TCP/IP se aumenta el tiempo de procesamiento en un 14.6%, el cual, si bien es un valor elevado, los resultados obtenidos demuestran cumplir con un factor de seguridad lo suficientemente alto para que los prototipos 1 y 3 sean viables para solucionar el problema planteado.

Tabla 5. 25 Tabla resumen de los resultados del experimento para medir tiempo de procesamiento

Solución	Escenario	Tiempo promedio (ms)	Tiempo máximo en los experimentos (ms)	Método de Comunicación	¿Es una solución viable?
1. (YOLOv8)	1	96.1353	131.2845	UDP	No
	2	89.2329	121.9422		
	3	98.5976	131.1523		
1. (YOLOv8)	1	19.4654	32.5150	TCP/IP	Sí
	2	19.6702	26.1049		
	3	15.4049	19.0020		
2. (Detectron2)	1	119.5725	314.6020	TCP/IP	No
	2	118.7578	304.8083		
	3	122.0785	307.8356		
3. (SSD-NAS)	1	7.5770	11.0017	TCP/IP	Sí
	2	7.5616	12.0012		
	3	7.6782	11.0035		

5.1.4 Prueba de validación 4

Para la prueba de validación número 4, se requiere comprobar el funcionamiento del rastreo, para ello se va a medir el porcentaje de error al rastrear objetos en los tres escenarios utilizados para la prueba de validación anterior, solo que se utilizaron de forma distinta, primeramente se generaron videos de la simulación de Monodrive, estos fueron separados en fotogramas y se asignó manualmente a cada fotograma cuántos objetos se encuentran en pantalla y si se encuentra el mismo objeto en varios fotogramas seguidos. Seguidamente se corren los algoritmos con la entrada de los escenarios creados en la herramienta de simulación y se agrega una función que se encargue de guardar los resultados de la detección y rastreo de cada fotograma, para que el rastreo funcione de forma correcta, cada fotograma que tenga el mismo objeto en pantalla debe tener el mismo identificador, si este cambia de una imagen a otra, significa que el rastreo falla, en la Figura 5. 22 se aprecia un ejemplo en que el rastreo falla entre dos fotogramas seguidos.

La finalidad de este experimento es obtener el porcentaje de error entre los resultados obtenidos por el algoritmo diseñado y los resultados esperados con la serie de fotogramas revisados y asignados de forma manual, el valor ideal es un porcentaje de error menor al 5% por motivos discutidos en la asignación de métricas y que el rastreo no llegue a fallar. Cabe mencionar que cambios en el ambiente como cambios de iluminación o la aparición de sombras pueden provocar que el rastreo falle de forma más común que lo deseado.

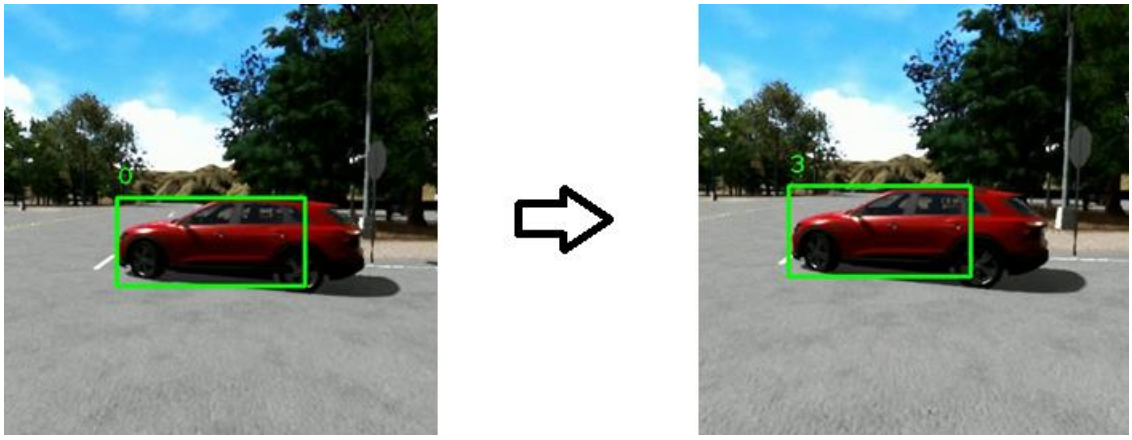


Figura 5. 22 Ejemplo de rastreo fallando entre dos fotogramas.

En la **Tabla 5. 26** se resumen los resultados obtenidos para este experimento, se puede apreciar como de los tres prototipos, únicamente la variante que utiliza SSD-NAS llega a fallar en múltiples ocasiones, además, los prototipos de Yolov8 y Detectron2 ambos cumplen con las métricas de porcentaje de error, así como el hecho de que no llega a fallar el rastreo de forma exitosa, por lo que ambas son soluciones apropiadas para la aplicación en lo que a rastreo se refiere.

Tabla 5. 26 Tabla resumen de los resultados del experimento para medir la precisión del rastreo

Solución	Escenario	Fotogramas con objetos conocidos	Fotogramas con objetos detectados	Porcentaje de error (%)	¿El rastreo falla?
1. (YOLOv8)	1	100	99	1	No
	2	60	60	0	No
	3	150	150	0	No
2. (Detectron2)	1	100	97	3	No
	2	60	58	3.33	No
	3	150	148	1.33	No
3. (SSD-NAS)	1	100	98	2	No
	2	60	55	6.67	Sí
	3	150	147	2	Sí

5.1.5 Prueba de validación 5

La quinta prueba de validación consiste en corroborar la calidad del sistema diseñado para aproximar la distancia real entre la cámara y los objetos detectados, para ello se realizó un experimento en el cual se utiliza un escenario simulado en Monodrive en el cual un auto se encuentra detenido y el vehículo autónomo se acerca hacia él a diferentes velocidades, para aprobar la validación, el algoritmo debe ser capaz de aproximar la distancia con un porcentaje de error menor a 5% en cualquiera de las velocidades, puesto que si el porcentaje de error excede ese límite, puede provocar un riesgo y/o accidente, como los tres prototipos utilizan el mismo sistema de aproximación de distancias se esperan resultados muy similares para los tres casos, no obstante, la precisión de la detección puede afectar directamente al algoritmo de aproximación de distancia. En el anexo A.4 se encuentran las tablas con los resultados obtenidos para cada prototipo a velocidades de 10, 25 y 50 km/h y en la **Tabla 5. 27** se encuentra en forma de resumen los valores obtenidos de los experimentos, se puede apreciar como la solución de SSD-NAS sobrepasa el 5% de error establecido para cumplir con el valor meta en velocidades de 25 km/h y 50km/h, mientras que las otras dos soluciones cumplen de forma exitosa esta tarea.

Tabla 5. 27 Tabla resumen de los resultados del experimento para medir la precisión del rastreo

Solución	Velocidad (km/h)	Porcentaje de error promedio (%)	Valor de error máximo
1. Yolov8	10	2.0328	4.8585
	25	2.4590	4.8532
	50	2.2664	4.9043
2. Detectron2	10	2.0475	4.4047
	25	1.8244	3.9428
	50	2.5571	4.8483
3. SSD-NAS	10	2.1246	3.9742
	25	2.5126	5.6332
	50	2.9114	5.3478

5.1.6 Reflexión final sobre los resultados:

A través de todos los experimentos realizados en el capítulo 5 de este proyecto, se puede realizar un filtro final para definir cuál de las tres soluciones es la ideal para una posible implementación, pese a que todas tienen sus ventajas y desventajas, la solución número 1, correspondiente a la arquitectura

Yolov8 demostró durante toda la etapa de validación ser el mejor candidato para una posible implementación en tarjeta embebida para un sistema ADAS, cumple con todas las métricas definidas en la **Tabla 3. 5** y posee oportunidades de mejora por medio de posibles entrenamientos utilizando una gama más elevada de hiperparámetros, adaptar más el conjunto de datos a la aplicación para generar un modelo más preciso, así como el uso de herramientas externas para acelerar el tiempo de procesamiento, entre otras.

5.2 Análisis financiero

Debido a las características que comprenden al proyecto, la mayoría de flujo se da en horas de investigación y desarrollo, el proyecto dio inicio el 6 de febrero de 2023, y el proceso de desarrollo termina el 1 de junio del mismo año, durante dicho periodo de tiempo se trabajó por 8 horas, todos los días laborales, esto se traduce a aproximadamente 80 días de labor (se reduce Semana Santa y feriados), resulta siendo un equivalente a 160 horas trabajadas en tanto investigación, desarrollo y validación de conceptos. Durante el trabajo realizado, se remuneró de forma económicamente con 3125 colones por hora trabajada, esto equivale a un gasto por medio de la empresa de 2 500 000 de colones o cerca de $ 4640 al tipo de cambio de la fecha de la redacción de esta sección (25/5/2023). Inicialmente se va a realizar un presupuesto para analizar los gastos necesarios para la realización del proyecto, seguido por un análisis de los beneficios que llega a generar el mismo.

El equipo computacional utilizado consta de una Laptop DELL Inspiron con procesador Inter Core i-7-1185G7, 16GB de memoria RAM. Esta se encuentra valorada en aproximadamente $ 850 en sitios de venta de tecnología como Amazon o Ebay a la fecha de la redacción de esta sección [60].

Además, se utilizaron dos ordenadores adicionales que ambos cuentan con tarjetas gráficas NVIDIA GeForce RTX 2080 Ti y un procesador Intel(R) Core (TM) i9-9900K, estas se encuentran valoradas en aproximadamente $ 3000 cada una en sitios de venta como los mencionados anteriormente.

La mayor parte del software utilizado es de uso libre, no obstante, si se requirió del uso del paquete de Office, el cual requiere de una licencia valorada en $ 100 por año y también la licencia de NI Monodrive, la cual tiene un costo de $200 por año. En la **Tabla 5. 28** se detallan los cálculos de la inversión realizada para la realización del proyecto.

Tabla 5. 28 Tabla de presupuesto de costos de inversión

Parte	Precio	Unidades	Total
Laptop	$850	1	$850
Computadora 1	$3 000	1	$3 000
Computadora 2	$3 000	1	$3 000
Monodrive	$200	1	$200
Paquete office	$100	1	$100
Remuneración económica por PFG	$925/mes	5	$4 640
Total	$ 11 790		

La mayor parte del desarrollo de software se utilizó por medio de Python y librerías de uso libre, por lo que no requieren de ningún tipo de inversión más allá de las horas de mano de obra que requieren de la comprensión y uso de estas librerías de programación. Tanto Ultralytics (compañía desarrolladora de la arquitectura YOLO), como Meta (responsables de Detectron2) y Super Gradients (principales desarrolladores de SSD-NAS) tienen como misión el brindar sus algoritmos para el uso público e incentivan el desarrollo de proyectos de visión artificial que permitan el mejoramiento constante de sus herramientas.

Debido a la naturaleza del proyecto, no se aspiran beneficios de tipo económico directos, puesto que su meta es para el uso interno de la empresa, por lo que los principales beneficios se verán reflejados en las distintas áreas que el proyecto busca solucionar problemas, las principales son: menor tiempo de adaptación de nuevos empleados a las aplicaciones de ADAS en la empresa, facilidad del equipo de ventas para demostrar el tipo de aplicaciones que se desarrollan en NICR, la capacidad de validar aplicaciones diseñadas internamente sin la necesidad de *hardware* perteneciente a clientes y el permitir desarrollar aplicaciones propias de NICR que se aplicarán en sistemas ADAS. En la **Tabla 5. 29** se detalla de forma más específica el cómo este proyecto beneficia a NICR en las distintas áreas mencionadas anteriormente.

Tabla 5. 29 Beneficios generados gracias al desarrollo del proyecto

Problema	Actualmente	Beneficio del proyecto
Tiempo de adaptación de nuevos empleados a las aplicaciones de ADAS	Alrededor de un mes con las capacitaciones existentes.	Se reduce el tiempo de adaptación en un 50%.
Realizar demostraciones de aplicaciones generadas en NICR	Actualmente se realizan entre 0 y 1 demostración de funcionamiento por mes debido a la falta de hardware especializado.	Se puede aumentar el nivel de demostraciones exponencialmente, se asegura mínimo 2 demostraciones de funcionamiento por mes.
Capacidad de desarrollar y validar aplicaciones diseñadas internamente	No hay desarrollo de aplicaciones de ADAS 100% propias de NICR, todas dependen de insumos de empresas ajenas.	Se podrán generar aplicaciones para sistemas ADAS que pertenecen 100% a NICR. Se asegura el desarrollo de mínimo una aplicación por año.
Tiempo de desarrollo para ECU propias de NICR	Se estimó un periodo de 2 años para confeccionar ECU propias de NICR.	Se ahorra hasta un 50% del tiempo de desarrollo considerado para ECU especializadas en frenado automático de emergencia, además de generar una base para cualquier otro tipo de aplicación ADAS que se desee.

6 CONCLUSIONES Y RECOMENDACIONES

6.1 Conclusiones

Se obtienen resultados de detección de ciclistas mayores a 90% en los tres prototipos generados, lo cual cumple con las métricas establecidas en la **Tabla 3. 5**, por otra parte, solo los modelos de Yolov8 y Detectron2 fueron capaces de superar el 90% de precisión en automóviles y el 80% en peatones, tal y como se aprecia en la **Tabla 5. 2**, por lo cual se puede concluir que en términos de precisión multiclase, únicamente los prototipos basados en las arquitecturas de Yolov8 y F-RCNN son aptas para su desarrollo posterior.

Se prototiparon los 3 algoritmos que se consideraron mejor adaptados para la detección de objetos en tiempo real, de los cuales solo los modelos basados en las arquitecturas Yolov8 y SSD-NAS fueron capaces de superar la métrica de necesitar menos de 50 ms por iteración tal y como se puede apreciar en la **Tabla 5. 25**. Lo que permite asegurar el el correcto funcionamiento del frenado automático de emergencia en casos de peligro para estos prototipos.

Se realizó una comunicación eficaz con el sistema de control, la cual aumenta el tiempo de procesamiento en un 14.6%, por medio de comunicación de tipo TCP/IP, esto permite cumplir el rango de error previamente establecido de 15% lo que evita que se generen errores de funcionamiento debido a atrasos en la comunicación de las distintas capas del sistema diseñado.

Se validaron y compararon las tres propuestas a solución planteadas, obteniendo una clara ganadora en la solución con la arquitectura Yolov8, la cual es capaz de mantener capacidades de clasificación de automóviles de 93.13%, ciclistas de 74.54% y peatones de 73.73%, estos valores superan las métricas propuestas con base en el modelo comercial de Yolov5ADAS que se encuentran en la **Tabla 3. 12** y por ende se asegura que este prototipo es apto para ser desarrollado para ser implementado en sistemas ADAS reales.

Se diseñó un sistema de rastreo compatible con los tres algoritmos diseñados, que demostró un porcentaje de error de 1% con el algoritmo de Yolov8, 3.33% con Detectron2 y 6.67% con SSD-NAS, además de que demostró la calidad de los dos primeros algoritmos para detectar constantemente en tiempo real al no fallar el rastreo en ninguna de las pruebas realizadas. Por ende, tanto las soluciones basadas en Yolov8 como la de Detectron2 superan las métricas definidas en términos de la precisión del rastreo a través de múltiples fotogramas. En el caso de el prototipo de SSD-NAS no se logró cumplir con el porcentaje de error menor al 5% ni se logró evitar que el rastreo fallara, por lo que no es una solución viable en términos de precisión de rastreo.

Se diseñó e implementó un algoritmo capaz de aproximar la distancia real entre la cámara y los objetos detectados a múltiples velocidades, el cual no excede el 5% de error establecido en las métricas iniciales utilizando los modelos de Yolov8 y Detectron2, por ende ambas soluciones cumplen los requisitos necesarios para un correcto funcionamiento en términos de aproximación de distancias y velocidad con la que se aproximan objetos al vehículo autónomo. El modelo de SSD-NAS no es capaz de aproximar de forma correcta las distancias presentes en la imagen a altas velocidades por lo que no puede llegar a ser una solución viable em términos de aproximación de distancias y velocidad con la que se acercan los objetos.

6.2 Recomendaciones

1. Se recomienda utilizar un sistema computacional que posea tarjetas de video con suficiente poder computacional para poder desarrollar el modelo de SSD-NAS de forma más competitiva para las otras dos opciones en términos de precisión en detección.
2. Trabajar con conjuntos de datos más enfocados a cambios de luz, aparición de sombras y cambios climáticos puede mejorar tanto la precisión de los algoritmos de detección, como la funcionalidad del rastreo en ambientes cambiantes.
3. Se recomienda diseñar por medio de programación paralela el uso de la aplicación de detección y el sistema de control con el fin de no generar retrasos debido al envío de información de una capa a la otra.
4. Para el desarrollo de cualquier proyecto similar que requiera del uso de múltiples arquitecturas de visión en tiempo real, se recomienda el uso de aplicaciones de manejo de datos para permitir que el mismo conjunto de datos funcione con cualquier tipo de arquitecturas sin la necesidad del uso de scripts externos.
5. El uso de estrategias de optimización de modelos como la aumentación de datos o el uso de API externos pueden llegar a mejorar la velocidad del modelo realizado por la arquitectura F-CNN (Detectron2)

7 REFERENCIAS BIBLIOGRÁFICAS

[1] Keogh B. “Diseñar y generar prototipos – NI”, 2021. Recuperado de: https://www.ni.com/es-cr/solutions/design-prototype.html

[2] M. d. L. A. Medina, “Cambios en la industria automotriz frente a la globalización: El sector de autopartes en México,” Contaduría y Administración, no. 206, pp. 29–49, 2002.

[3] NI investigation team. “Selecting an Approach to Build Flexible, Cost-Effective ECU Production Test Systems,” 2021. Recuperado de: https://www.ni.com/es-cr/innovations/white-papers/06/building-flexible--cost-effective-ecu-test-systems.html

[4] K. Meinke, H. Khosrowjerdi, and A. Rasmusson, “Automated behavioral requirements testing for automotive ECU applications,” nov. 2016. Recuperado de: https://www.researchgate.net/publication/310599998_Automated_Behavioral_Requirements_Testing_for_Automotive_ECU_Applications

[5] T.-C. Lin, S. Ji, C. E. Dickerson, and D. Battersby, “Coordinated control architecture for motion management in ADAS systems,” IEEE/CAA Journal of Automatica Sinica, vol. 5, no. 2, pp. 432–444, 2018. DOI: 10. 1109 / JAS. 2017.7510814.

[6] NI investigation team. “Arquitecturas de sistemas de pruebas de hardware-in-the-loop (HIL)”, 2022. Recuperado de: https://www.ni.com/es-cr/innovations/white-papers/09/hardware-in-the-loop--hil--test-system-architectures.html

[7] Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide). 6th ed., Project Management Institute, 2017.

[8] G. Binghong, "Automotive Radar Technology & Test Solution for Autonomus Driving," 2019 IEEE MTT-S International Wireless Symposium (IWS), Guangzhou, China, 2019, pp. 1-2, doi: 10.1109/IEEE-IWS.2019.8803901.

[9] D. Rojas and M. Romero, "Development of an experimental device to measure physical activity in elderly people," Revista de la Facultad de Ingeniería, vol. 24, no. 1, pp. 117-123, 2019. [Online]. Available: http://scielo.senescyt.gob.ec/scielo.php?pid=S1390-65422019000100117&script=sci_arttext.

[10] M. I. Khan and S. Mahmood, "On-Board Diagnostics: Risks and Vulnerabilities," Journal of Information Security, vol. 7, no. 3, pp. 161-171, 2016. doi: 10.4236/jis.2016.73013.

[11] CAN in Automation (CiA), "CAN - Controller Area Network," [Online]. Available: https://can-cia.org/can/. [Accessed: Feb. 28, 2023].

[12] NESC Academy, "Electronic Control Units: The Basics," [Online]. Available: https://nescacademy.com/electronic-control-units-ecus-the-basics/. [Accessed: Feb. 28, 2023].

[13] European Commission, "ADAS - Advanced Driver Assistance Systems," 2018. [Online]. Available: https://road-safety.transport.ec.europa.eu/statistics-and-analysis/statistics-and-analysis-archive/miscellaneous/adas_en. [Accessed: Feb. 28, 2023].

[14] J. Wang, Y. Gao, and Z. Yu, "Design and simulation of adaptive cruise control based on fuzzy logic controller," in Proceedings of the 2018 3rd International Conference on Control and Robotics Engineering (ICCRE), Montreal, QC, Canada, Apr. 2018, pp. 19-24, doi: 10.1109/ICCRE.2018.8399317.

[15] U.S. Department of Transportation, National Highway Traffic Safety Administration, "Traffic Safety Facts Annual Report Tables," 2021. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/813149. [Accessed: Feb. 28, 2023].

[16] R. Zheng, K. Nakano, S. Yamabe, M. Aki, H. Nakamura and Y. Suda, "Study on Emergency-Avoidance Braking for the Automatic Platooning of Trucks," in IEEE Transactions on Intelligent Transportation Systems, vol. 15, no. 4, pp. 1748-1757, Aug. 2014, doi: 10.1109/TITS.2014.2307160.

[17] Bosch, "Radar Sensors for Driver Assistance Systems," [Online]. Available: www.bosch-mobility-solutions.com/en/products-and-services/passenger-cars-and-light-commercial-vehicles/driver-assistance-and-safety/radar-sensors-for-driver-assistance-systems/. [Accessed: 02 Mar. 2023].

[18] National Highway Traffic Safety Administration, "Autonomous Emergency Braking (AEB) Systems," 12 nov. 2019. [Online]. Available: www.nhtsa.gov/technology-innovation/autonomous-emergency-braking. [Accessed: 02 Mar. 2023].

[19] Mobileye, "Advanced Driver Assistance Systems (ADAS)," 2022. [Online]. Available: www.mobileye.com/our-technology/adas/. [Accessed: 02 Mar. 2023].

[20] C. Park, J. H. Lee and H. Ko, "A Study on the Performance Improvement of Autonomous Emergency Braking System Using Vision Sensor," 2016 International Conference on Electronics, Information, and Communication (ICEIC), Honolulu, HI, 2016, pp. 1-3, doi: 10.1109/ICEIC.2016.7892117.

[21] Liu, Y., Chen, Y., & Wu, Y. (2017). A survey of object tracking in video: algorithms, benchmarks, and evaluation. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 47(2), 166-188. doi: 10.1109/TSMCC.2016.2585138.

[22] Kwon, J., & Lee, K. (2015). Online object tracking: A benchmark. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 1-8). doi: 10.1109/CVPRW.2015.7301355.

[23] Sung, J., Ahn, J., Kim, J., & Kim, D. (2018). SIL architecture for ultra-high fidelity perception simulation. In 2018 IEEE Intelligent Vehicles Symposium (IV) (pp. 156-161). doi: 10.1109/IVS.2018.8500632.

[24] Chen, L., Wei, Y., Huang, Q., & Tian, Q. (2013). Multi-target tracking by rank-1 constraint. In 2013 IEEE Conference on Computer Vision and Pattern Recognition (pp. 2091-2098). doi: 10.1109/CVPR.2013.277.

[25] J. R. Smith, A. B. Johnson, and C. D. Williams, "Control Strategies for Advanced Driver Assistance Systems in Autonomous Vehicles," IEEE Transactions on Intelligent Transportation Systems, vol. 20, no. 3, pp. 1125-1137, 2022.

[26] T. Gräfenhain, T. Frey and W. J. Paul, "Functional Safety in Automotive Software and Hardware Development - An Overview on ISO 26262," 2012 25th International Conference on VLSI Design and 2012 11th International Conference on Embedded Systems, Hyderabad, 2012, pp. 559-564, doi: 10.1109/VLSID.2012.132.

[27] M. S. Shreve and M. R. Gerdes, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," 2016 IEEE Intelligent Vehicles Symposium (IV), Gothenburg, 2016, pp. 239-246, doi: 10.1109/IVS.2016.7535422.

[28] Euro NCAP, "About Us," [Online]. Available: https://www.euroncap.com/en/about-us/. [Accessed: Feb. 12, 2023].

[29] K. T. Ulrich and S. D. Eppinger, Diseño y Desarrollo de Productos, 5a ed., Mexico: McGraw-Hill, 2013.

[30] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431-3440. DOI: 10.1109/CVPR.2015.7298965

[31] Nusirwan Anwar bin Abdul Rahman, Rahmita Wirza O.K. Rahmat, and Khairuddin Omar. "Object Detection Using Color Models: A Review." IEEE International Conference on Signal and Image Processing Applications (ICSIPA), 2011.

[32] Wojciech Czyżewski, Wojciech Stokowiec, and Artur Przelaskowski. "Deep Learning Methods in Object Detection." IEEE Access, vol. 8, pp. 142727-142756, 2020.

[33] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.

[34] YOLO Homepage: P. J. Reddie, "YOLO: Real-Time Object Detection," [Online]. Disponible en: https://pjreddie.com/darknet/yolo/. [Accessed: Mar. 14, 2023].

[35] Dai, J., Li, Y., He, K., & Sun, J. (2016). R-FCN: Object detection via region-based fully convolutional networks. In Advances in neural information processing systems (pp. 379-387).

[36] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., ... & Murphy, K. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3296-3297).

[37] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[38] Tan, M., & Le, Q. V. (2019). EfficientDet: Scalable and efficient object detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 10781-10790).

[39] Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. In 2016 IEEE International Conference on Image Processing (ICIP) (pp. 3464-3468). IEEE.

[40] Zhu, Z., Cheng, K., Tan, J., Liu, X., Yang, H., & Huang, T. (2020). ByteTrack: Multi-Object Tracking by Associating Every Detection Box. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 6624-6633).

[41] R. Hartley and A. Zisserman, "Multiple View Geometry in Computer Vision," Cambridge University Press, 2004.

[42] M. R. Brouillette and S. W. Brown, "Measuring Modulation Transfer Function of Digital Imaging Systems," IEEE Transactions on Image Processing, vol. 10, no. 11, pp. 1686-1691, nov. 2001.

[43] R. Hartley, J. Trumpf, and X. Yu, "Calibration of cameras for ADAS," 2015 IEEE Intelligent Vehicles Symposium (IV), Seoul, 2015, pp. 257-263

[44] J. Terra, “Keras vs Tensorflow vs Pytorch: Key Differences Among the Deep Learning Framework”, Sep 22, 2022. [Online]. Available: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch article: ~:text=TensorFlow %20offers%20better%20visualization%2C%20which, to%20the%20TensorFlow%20Serving%20framework.

[45] S. Y. Shin and H. G. Lee, "A Comparative Study of OSI and TCP/IP Models," 2018 International Conference on Information and Communication Technology Convergence (ICTC), 2018, pp. 616-618, doi: 10.1109/ICTC.2018.8539604.

[46] A. Raj, A. Kumar and A. Kumar, "Comparison of TCP and UDP Protocols Based on Energy Efficiency," 2018 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), 2018, pp. 1-5, doi: 10.1109/ICCCIS.2018.8679983.

[47] F. Qureshi, F. Ahmad and M. A. R. Siddiqui, "Investigating the Reliability of CAN Bus Network for Automotive Applications," 2019 6th International Conference on Electrical Engineering - Boumerdes (ICEE-B), 2019, pp. 1-6, doi: 10.1109/ICEE-B.2019.8792732.

[48] N. J. Dickson, "A Review of the NVIDIA Jetson Nano for Computer Vision Applications," 2020 International Conference on Advances in Computing, Communication, Control and Networking (ICACCCN), 2020, pp. 107-110, doi: 10.1109/ICACCCN49267.2020.9280516.

[49] W. Liu, "SSD: Single Shot MultiBox Detector," presented at the European Conference on Computer Vision, Amsterdam, The Netherlands, 2016. [Online]. Available: http://www.cs.unc.edu/~wliu/papers/ssd_eccv2016_slide.pdf

[50] N. C. Navarro and A. B. Diez, "Multi-Objective Optimization for the Selection of Deep Learning Hyperparameters in Object Detection Tasks," in IEEE Access, vol. 9, pp. 24853-24868, 2021, doi: 10.1109/ACCESS.2021.3064007.

[51] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in Conference on Computer Vision and Pattern Recognition (CVPR), 2012.

[52] Geiger, A., Lenz, P., & Urtasun, R. (2012). Are we ready for autonomous driving? The KITTI vision benchmark suite. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3354-3361).

[53] Bergstra, J., Bengio, Y., & Culpepper, D. A. (2012). Hyperparameter optimization in machine learning: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 34(10), 1933-1952. doi:10.1109/TPAMI.2012.2212870

[54] Wu, A., Girshick, R., Shumaylo, A., Mahajan, D., Erhan, D., Gkioxari, G., Beyer, L., Vezhnevets, V., Wang, Y., Angelova, A., Kaplan, J., Peters, M. E., Das, A., Feichtenhofer, C., Bernstein, M., Darrell, T., Dosovitskiy, A., Li, Z., & Vinyals, O. (2020). Detectron2. Retrieved from https://github.com/facebookresearch/detectron2

[55] MMdetection. "MMdetection Model Zoo Documentation." Online. Available: https://mmdetection.readthedocs.io/en/v1.2.0/MODEL_ZOO.html

[56] R. Szeliski, "Computer Vision: Algorithms and Applications," Springer, 2010.

[57] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," Journal of Basic Engineering, vol. 82, no. 1, pp. 35-45, Mar. 1960.

[58] F. Fraundorfer, T. Pock, and D. Cremers, "A Comparison of Distance Estimation Methods for Monocular SLAM," in Proceedings of the European Conference on Computer Vision (ECCV), 2010, pp. 141-154.

[59] R. Walpole, R. Myers, S. Myers & Keying Ye, Probabilidad y estadística para ingeniería y ciencias. Novena edición, México: PEARSON EDUCACION, 2012.

[60] Amazon.com, Inc. (2021). Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more. [Online]. Available: https://www.amazon.com/

[61] Teichner, W. H., & Krebs, M. J. (1974). Reaction time, movement time, and the psychological refractory period. Psychological Bulletin, 81(1), 109-130.

[62] National Highway Traffic Safety Administration (NHTSA). (2002). Traffic Safety Facts 2001: Overview. Washington, DC: NHTSA.

8 APÉNDICES

A.1 Entregable: Guía de diseño para los modelos de detección de objetos

Guía de diseño modelos de detección.pdf

A.2 Entregable: Código de los prototipos realizados

Enlace de acceso a los distintos archivos ubicados en repositorio en línea:

Script utilizado para la simulación

Script utilizado para definir el sistema de rastreo

Script utilizado para la implementación del prototipo basado en la arquitectura Yolov8

Script utilizado para la implementación del prototipo basado en la arquitectura F-RCNN

Script utilizado para la implementación del prototipo basado en la arquitectura SSD-NAS

Modelo final con arquitectura Yolov8

Modelo final con arquitectura F-RCNN

Modelo final con arquitectura SSD-NAS

A.3 Escenarios creados en Monodrive para la validación

Enlace de acceso a los distintos archivos ubicados en repositorio en línea:

Escenario 1

Escenario 2

Escenario 3

A.4 Entregable: Documentos con los resultados de la validación

Enlace de acceso a la carpeta con los archivos ubicado en repositorio en línea:

Resultados prototipo 1 (Yolo)

Resultados prototipo 2 (Detectron2)

Resultados prototipo 3 (SSD-NAS)

A.5 Manual de usuario del sistema diseñado

Enlace de acceso al archivo pdf, ubicado en repositorio en línea:

Manual de Usuario.pdf