

APLICATIVO EMBEBIDO DE ALERTA DE COLISIÓN CON OTROS VEHÍCULOS USANDO 3 CÁMARAS ESTÉREO

FRANKLIN DAVID ATUESTA FLOR

Código: 201922928

Director:
Bladimir Bacca Cortes PhD.
Área informática industrial

Universidad del Valle
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica y Electrónica
Santiago de Cali
Junio 2025

APLICATIVO EMBEBIDO DE ALERTA DE COLISIÓN CON OTROS VEHÍCULOS USANDO 3 CÁMARAS ESTÉREO

FRANKLIN DAVID ATUESTA FLOR

Código: 201922928

Documento presentado al programa de:

Ingeniería Electrónica (3744)

como requisito para optar por el título de:

Ingeniero Electrónico

Director:

Bladimir Bacca Cortes PhD.

Área informática industrial

Universidad del Valle

Facultad de Ingeniería

Escuela de Ingeniería Eléctrica y Electrónica

Santiago de Cali

Junio 2025

Nota de aceptación:



Firma del director del trabajo

Firma del evaluador

Firma del evaluador

RESUMEN

Los sistemas avanzados de asistencia al conductor, conocidos por sus siglas en inglés como ADAS (Advanced Driver Assistance Systems). Son tecnologías que pueden emitir alertas al conductor o intervenir directamente en el control del vehículo para evitar posibles accidentes. En países de bajos ingresos, estas herramientas no son de fácil acceso debido a su alto costo y aunque existen módulos adaptables como cámaras frontales o radares, estos no ofrecen lo necesario para alertar eficazmente al conductor sobre posibles colisiones con otros vehículos. Este proyecto se centra en el desarrollo de un aplicativo de alerta de colisión con otros vehículos usando 3 cámaras estéreo.

El objetivo principal del aplicativo es generar alertas visuales para el conductor, evaluando el riesgo de colisión con otros vehículos. Para ello, se basa en las distancias relativas, las trayectorias de los vehículos cercanos y la velocidad del propio vehículo, utilizando la información captada por cámaras estéreo. El desarrollo incluye la creación de algoritmos para evaluar colisiones y generar sus respectivas alertas, el uso de redes neuronales para detectar los objetos de interés en las escenas y la integración de las tres cámaras para generar una panorámica de la vista frontal del vehículo.

Los resultados de este proyecto demuestran la viabilidad y la efectividad del uso de cámaras estéreo y redes neuronales para generación de alertas, pues logro una precisión del 94,2 % y una exactitud del 86,78%.

Palabras clave: Sistemas ADAS, alerta de colisión, cámaras estéreo, visión por computador.

ABSTRACT

Advanced Driver Assistance Systems, known by their acronym ADAS, are technologies that can issue alerts to the driver or directly intervene in vehicle control to prevent potential accidents. In low-income countries, these tools are not easily accessible due to their high cost. Although adaptable modules like front cameras or radars exist, they don't provide what's necessary to effectively alert the driver about potential collisions with other vehicles. This project focuses on developing a collision alert application with other vehicles using three stereo cameras.

The main objective of the application is to generate visual alerts for the driver by evaluating the risk of collision with other vehicles. To do this, it relies on the relative distances, trajectories of nearby vehicles, and the speed of the vehicle itself, using information captured by stereo cameras. The development includes creating algorithms to evaluate collisions and generate their respective alerts, using neural networks to detect objects of interest in the scenes, and integrating the three cameras to generate a panoramic view of the vehicle's front view.

The results of this project demonstrate the viability and effectiveness of using stereo cameras and neural networks for alert generation, achieving a precision of 94.2% and an accuracy of 86.78%.

Keywords: ADAS systems, collision warning, stereo cameras, computer vision.

TABLA DE CONTENIDO

1. INTRODUCCIÓN.....	1
1.1 PLANTEAMIENTO DEL PROBLEMA	2
1.2 OBJETIVOS	4
1.2.1 Objetivo general	4
1.2.2 Objetivos específicos.....	4
1.3 SOLUCIÓN PROPUESTA	5
1.4 ESTRUCTURA DEL LIBRO	5
2. ANTECEDENTES Y MARCO TEÓRICO	7
2.1 INTRODUCCIÓN	7
2.2 ANTECEDENTES	7
2.3 MARCO CONCEPTUAL.....	14
2.3.1 Advanced Driver Assistance Systems (ADAS).....	14
2.3.2 Visión Estéreo	15
2.3.3 Stitching de imágenes	19
2.3.4 Redes Neuronales Convolucionales (CNN).....	20
2.3.5 Seguimiento de Objetos (Object Tracking).....	21
2.3.6 Filtros Kalman	23
2.3.7 Estimación de colisión con vehículos	24
2.3.8 Metodología RUP.....	25
2.4 OBSERVACIONES FINALES.....	26
3. DESARROLLO DE UN APLICATIVO EMBEBIDO DE ALERTA DE COLISIÓN CON OTROS VEHÍCULOS.....	27
3.1 INTRODUCCIÓN	27
3.2 DISEÑO DEL APLICATIVO EMBEBIDO DE ALERTA DE COLISIÓN CON OTROS VEHÍCULOS.....	27
3.2.1 Requerimiento funcionales y no funcionales.....	27
3.2.2 Diagrama de concepto	29
3.2.3 Diagrama de clases	31
3.2.4 Diagrama de nodos	34
3.3 IMPLEMENTACIÓN APLICATIVO EMBEBIDO DE ALERTA DE COLISIÓN CON OTROS VEHÍCULO.....	37
3.3.2 Módulo de procesamiento	40
3.4 CONCLUSIONES	61
4. RESULTADOS Y ANÁLISIS	63

4.1 INTRODUCCIÓN	63
4.3 PRUEBAS DE INTEGRACIÓN.....	63
4.4 PRUEBAS DE CAMPO	74
4.5 CONCLUSIONES	81
5. CONCLUSIONES Y TRABAJOS FUTUROS	83
5.1 Conclusiones generales	83
5.2 Trabajos futuros	84
6. REFERENCIAS	85
7. ANEXOS.....	90

Índice de figuras

Figura 1-1. Sistema de reconocimiento de señales	1
Figura 1-2. Vehículo con modulo integrado (coche de pruebas Ford Argo AI en Detroit en 2019).	1
Figura 1-3. Diagrama conceptual de la solución propuesta.	5
Figura 2-1. Ejemplo cámara estéreo. Modelo ZED2 de la empresa StereoLabs (https://store.stereolabs.com/products/zed-2).....	15
Figura 2-2. Par estéreo reajustado, rectificado y con una correspondencia conocida, donde la distancia Z se puede encontrar mediante triángulos semejantes.	16
Figura 2-3. Mapa de disparidad formado a partir de imágenes estéreo rectificadas.	17
Figura 2-4. Parámetros extrínsecos de la calibración estéreo.....	17
Figura 2-5. Patrón de ajedrez.....	18
Figura 2-6. Detección de esquinas para posterior rectificación.....	18
Figura 2-7. Vecindad de pixeles de una imagen en escala de grises.....	19
Figura 2-8. Ejemplo de procesamiento de imágenes por extracción de características.....	19
Figura 2-9. Estructura básica de una red neuronal convolucional.	20
Figura 2-10. Etapas del filtro de Kalman Extendido.....	23
Figura 2-11. Algoritmos basados en la percepción.	25
Figura 2-12. Algoritmos cinemáticos.	25
Figura 3-1. Diagrama de conceptual.	30
Figura 3-2. Diagrama de Clases.	32
Figura 3-3. Diagrama de nodos.	33
Figura 3-4. Imágenes RGB de las cámaras izquierda, central y derecha.....	34
Figura 3-5. Matrices de profundidad de las cámaras izquierda, central y derecha.	34
Figura 3-6. Imágenes en escala de grises para la visualización de la profundidad.	34
Figura 3-7. Panorámica de las matrices de profundidad de cada cámara.....	35
Figura 3-8. Panorámica RGB.	35
Figura 3-9. Panorámica en escala de grises normalizada con falso color.	35
Figura 3-10. Mensaje custom para los objetos detectados.	36
Figura 3-11. Mensaje custom para publicar las distancias de los objetos detectados.....	36

Figura 3-12. Imagen con la colisión evaluada y las distancias de seguridad visibles.....	37
Figura 3-13. Imagen con la proyección de la trayectoria.	37
Figura 3-14. GUI módulo de configuración.	38
Figura 3-15. Homografía 2D.....	39
Figura 3-16. GUI con la información cargada.	40
Figura 3-17. GUI módulo de procesamiento o ejecución.....	41
Figura 3-18. Disposición de las cámaras.	42
Figura 3-19. Par de imágenes a unir.	43
Figura 3-20. Matching de las características.	43
Figura 3- 21. Imágenes resultantes después de aplicar el warping.	44
Figura 3-22. Imágenes resultantes después de aplicar transformaciones de perspectiva.	44
Figura 3-23. Panorámica resultante.	44
Figura 3-24. Matriz de confusión YOLOv5l con Dataset 1.....	48
Figura 3-25. Matriz de confusión YOLOv8m con Dataset 1.	48
Figura 3-26. Matriz de confusión YOLOv5l Dataset 2.	49
Figura 3-27. Matriz de confusión YOLOv8m con Dataset 2.	50
Figura 3-28. ByteTrack workflow Imagen tomada de [36].	52
Figura 3-29. Fotogramas A y B con algoritmo BYTETRACK implementado en video de carretera. Se conserva el ID:2.....	55
Figura 3-30 Fotogramas C y D con algoritmo BYTETRACK implementado en video de carretera. Se conserva el ID:9.....	55
Figura 3-31. Ejemplo de publicación del mensaje que contiene ID y distancia correspondiente.	58
Figura 4-1. Vehículos usados para las pruebas de campo.....	74
Figura 7-1. Estructura Anexos.....	90

Índice de tablas

Tabla 2-1. Trabajos previos clasificados según sus puntos de interés	8
Tabla 3-1. Requerimientos funcionales.....	28
Tabla 3-2. Requerimientos no funcionales.	28
Tabla 3-3. Comparación entre los diferentes modelos YOLO más recientes del 2023 [47].	45
Tabla 3-4. Métricas obtenidas después del entrenamiento con el Dataset 1.	47
Tabla 3-5. Métricas obtenidas después del entrenamiento con el Dataset 2.	49
Tabla 3-6. Comparación de los métodos de última generación bajo el protocolo “detector privado” en el conjunto de pruebas MOT17. Resultados obtenidos de [50].	54
Tabla 3-7. Niveles de alerta de acuerdo con el Tiempo de Colisión.	61
Tabla 4-1. Pruebas de integración Modulo de Configuración.	63
Tabla 4-2. Pruebas de integración Modulo de Ejecución.....	68
Tabla 4-3. Pruebas realizadas.....	75
Tabla 4-4. Parámetros para obtener la matriz de confusión.....	77
Tabla 4-5. Resultados generación de alerta para el vehículo quieto.	78
Tabla 4-6. Resultados generación de alerta para el vehículo en movimiento.	80

1. INTRODUCCIÓN

El crecimiento demográfico ha provocado un notable incremento en el número de vehículos en circulación, lo que a su vez ha resultado en un aumento de los accidentes de tránsito. Con el objetivo de hacer frente a esta situación y mitigar las significativas pérdidas económicas y de vidas humanas que se generan, se ha implementado el uso de diferentes herramientas tecnológicas como los sistemas de frenado automático, sistemas para reconocimiento de señales de tráfico, control del crucero adaptativo, sistemas de alerta de colisión, entre otras [1].



Figura 1-1. Sistema de reconocimiento de señales (Tomado de: [2]).

Para lograr un transporte más seguro y eficiente, las principales marcas automotrices han desarrollado las tecnologías ADAS (por sus siglas en Advanced Driver Assistance Systems), que son sistemas avanzados de asistencia al conductor. Como su nombre sugiere, estas herramientas buscan acompañar y ayudar al conductor mediante alertas en tiempo real o actuando directamente sobre el vehículo para prevenir situaciones de riesgo.

No obstante, muchas de estas tecnologías no han logrado reducir de manera significativa la accidentalidad vial, principalmente porque no son accesibles para la mayoría de los conductores. Esta limitación resulta aún más crítica si se tiene en cuenta que cerca del 93% de los accidentes de tránsito ocurren en países de ingresos bajos y medianos [3]. Una posible solución al problema es usar módulos adaptables que realicen funciones específicas de los sistemas ADAS. Por ejemplo, en la figura 1-2 se puede ver que el vehículo tiene en la parte superior un módulo integrado, este puede contener LIDAR, cámaras, Infrarrojos o unidades de cómputo, lo cual es fácilmente adaptable a cualquier vehículo en comparación a un sistema de frenado automático o un sistema de conducción autónomo.



Figura 1-2. Vehículo con modulo integrado (coche de pruebas Ford Argo AI en Detroit en 2019).

Los sistemas de alerta de colisión con otros vehículos pueden implementarse de dicha forma, pues los elementos que lo componen se pueden integrar a diferentes vehículos. Ya que solo necesitan de un sensor o sensores para obtener información del entorno, un módulo de procesamiento de los datos y una interfaz de usuario para las alertas generadas. Para este tipo de sistemas es posible utilizar cámaras estéreo, ya que permiten obtener una gran cantidad de información del entorno, la cual resulta fundamental para la detección y el seguimiento de vehículos.

Esta propuesta de trabajo de grado está orientada a el diseño e implementación de un aplicativo embebido de alerta de colisión con otros vehículos usando 3 cámaras estéreo, el objetivo principal es proporcionar asistencia a los conductores al alertarles sobre posibles colisiones con otros vehículos, buscando así reducir los accidentes de tránsito.

Este trabajo de grado surge del interés en contribuir al bienestar de las personas mediante la implementación de soluciones, tanto en software como en hardware, con el objetivo de mejorar su calidad de vida. El desarrollo de un aplicativo embebido para la alerta de colisiones vehiculares, con un enfoque modular, representa un proyecto con potencial para convertirse en una herramienta replicable y comercializable en la industria. Esto abriría la posibilidad de que más individuos accedan a herramientas que les asistan durante la conducción.

Además, esta iniciativa brinda la oportunidad de ampliar los conocimientos en diversas áreas de interés para la comunidad de la Facultad de Ingeniería. Esto incluye la exploración de nuevas tecnologías ampliamente utilizadas en la industria global, como las cámaras estéreo de Stereo Labs y el sistema embebido Nvidia Jetson AGX Orin de Nvidia Corporation, comúnmente empleado en aplicaciones de inteligencia artificial y procesamiento de imágenes. Asimismo, implica el uso de frameworks reconocidos a nivel mundial en el campo de la robótica, como ROS2.

1.1 PLANTEAMIENTO DEL PROBLEMA

La seguridad vial es una preocupación prioritaria a nivel global ya que según datos de la Organización Mundial de la Salud cada año mueren aproximadamente 1.3 millones de personas por accidentes de tráfico. Además, se estima que anualmente se producen entre 20 y 50 millones de traumatismos no mortales por dichos accidentes y que en el 2030 será la quinta causa de muerte a nivel mundial [3]. Las lesiones derivadas de accidentes de tránsito resultan en considerables pérdidas económicas para las personas afectadas, sus familias y para los países en general. Las colisiones relacionadas con el tráfico generan costos que, en promedio, representan alrededor del 3% del PIB de la mayoría de los países [3].

En el caso específico de Colombia, las muertes por colisiones relacionadas con el tráfico representaron el 62.2% del total nacional de fallecidos para el año 2022 [4]. Estas colisiones son de muchos tipos, pero se pueden identificar 3: colisión por invasión de carril, colisión en intersecciones y colisión en la parte trasera del vehículo. En el año 2012 en Cali ocurrieron 1320 accidentes debido a la invasión de carril, 981 por cruces en intersecciones y 379 por colisión en la parte trasera [5].

Es en este contexto que los sistemas de alerta de colisiones se muestran como una solución viable para disminuir la siniestralidad en las carreteras, pues estos son una tecnología diseñada para prevenir accidentes de tráfico. Su principal función es alertar al conductor de un posible choque con otro vehículo, objeto o peatón. Esto se realiza mediante el uso de sensores, cámaras y algoritmos que permiten realizar el seguimiento de los objetos en la carretera, a fin de avisar al conductor la situación de peligro.

Este tipo de herramienta hace parte de un conjunto más grande de tecnologías conocidas como los sistemas avanzados de asistencia al conductor, también llamados sistemas ADAS por sus siglas en inglés (Advanced Driver Assistance System). El objetivo principal de estos es proporcionar un entorno más seguro para el conductor y contribuir a la seguridad del tráfico en general. Para lograr esta labor ofrecen diferentes soluciones como: la conducción automatizada en autopistas, la detección de peatones, el reconocimiento de señales de tráfico, el estacionamiento automatizado, la detección de fatiga del conductor entre otras [6].

Sin embargo, la implementación de estos sistemas de alerta puede ser variada, pues existen diversos métodos para la identificación y seguimiento de objetos, entre ellos destacan los basados en características, segmentación, estimaciones y aprendizaje. Cada uno con diferentes niveles de robustez, adaptabilidad y procesamiento [7].

Los métodos clásicos de seguimiento de objetos suelen ser los métodos basados en extracción de características, donde se extraen propiedades como color, textura, flujo óptico (patrón de movimiento aparente), etc. Estas características extraídas deben ser únicas para que los objetos puedan distinguirse fácilmente en el campo de visión del sensor. También está la extracción basada en estimación de movimiento, donde el objeto está representado por un vector de estado. El vector de estado describe el comportamiento dinámico de un sistema, como la posición y la velocidad de un objeto. En este grupo se encuentran los filtros Kalman y los filtros de partículas [7].

Por otro lado, las implementaciones actuales han incorporado métodos basados en aprendizaje, pues se ha demostrado que son altamente efectivos en la extracción de características y clasificación de objetos, lo que sugiere que el uso de técnicas de aprendizaje profundo puede mejorar el rendimiento del seguimiento [7].

Por ejemplo: algunas alertas de colisiones han implementado redes neuronales convolucionales para calcular distancias entre vehículos u objetos y evitar colisiones en la parte trasera del vehículo, tomando datos desde una cámara ubicada en esa misma zona [8]. Otros hacen uso de la API de detección de objetos de Google Tensor Flow (gran plataforma para construir y entrenar redes neuronales) con 3 funciones básicas: detección frontal del automóvil, cálculo del factor de seguridad y estimación de la distancia frontal del automóvil y por último discriminación del estado de conducción. Donde la información la toma de una cámara, ubicada en el frente del vehículo [9].

Ahora bien, el problema radica en que estas tecnologías en su gran mayoría vienen integradas a vehículos de alta gama, por lo que en Colombia no es una opción viable considerando que el 63,8% de las personas no ganan más de un salario mínimo [10]. Más

aún cuando tampoco existen muchas empresas que comercialicen esta tecnología como módulos que se puedan adaptar o integrar a diferentes vehículos. Además, como se puede notar en los desarrollos mencionados anteriormente, muchas de las alertas anticollisiones son para la parte delantera y trasera del vehículo, dejando de lado las colisiones que se pueden dar en las partes laterales del vehículo.

En esta propuesta se contempla emplear tres cámaras estéreo (ya que con esta cantidad se puede abarcar un área visual más amplia). Estas se conectarían a un sistema embebido encargado de procesar las imágenes y, mediante un algoritmo definido usando alguno de los métodos de extracción por aprendizaje, ser capaz de seguir los objetos y emitir las alertas oportunas. Entonces, teniendo en cuenta lo mencionado anteriormente se plantea la siguiente pregunta: ¿Cómo se puede desarrollar un aplicativo embebido que use cámaras estéreo para generar alertas (auditivas o visuales) de colisión con otros vehículos?

1.2 OBJETIVOS

1.2.1 Objetivo general

Desarrollar un aplicativo embebido de alerta de colisión con otros vehículos usando 3 cámaras estéreo y técnicas de visión por computador.

1.2.2 Objetivos específicos

1. Obtener los requerimientos funcionales y no funcionales del aplicativo embebido, considerando la revisión bibliográfica sobre los algoritmos de seguimiento y detección de vehículos utilizando visión estéreo.
2. Diseñar el aplicativo embebido de alerta de colisión con otros vehículos usando visión estéreo.
3. Implementar el aplicativo embebido de alerta de colisión con otros vehículos empleando visión estéreo.
4. Realizar pruebas de integración y de campo del aplicativo embebido de alerta de colisión con otros vehículos.

1.3 SOLUCIÓN PROPUESTA

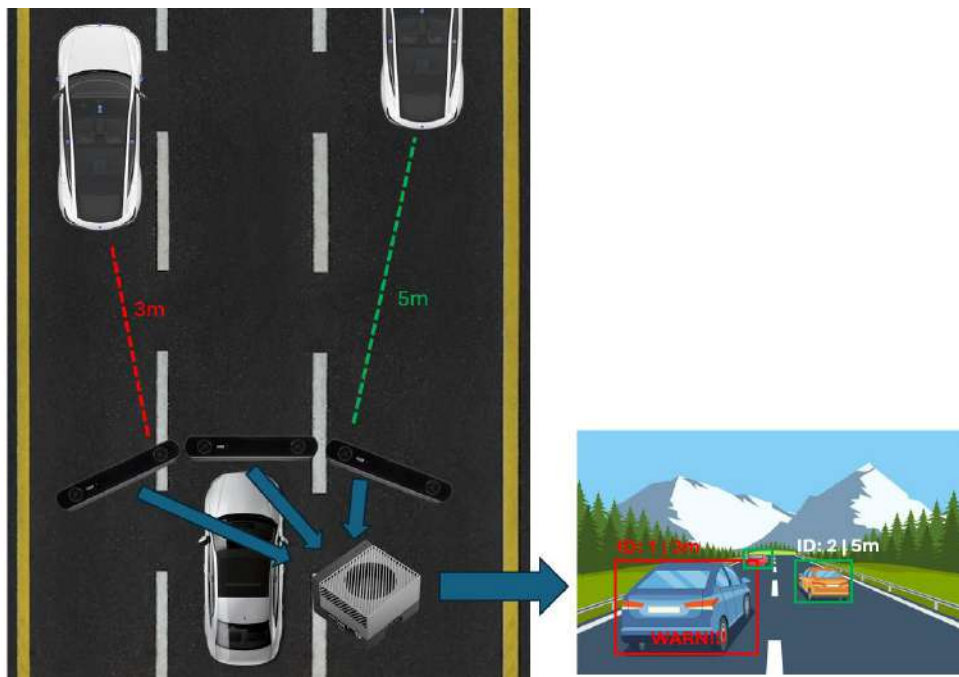


Figura 1-3. Ilustración de la solución propuesta.

El diagrama presentado en la Figura 1-3 ilustra el funcionamiento de la solución propuesta. Este sistema incluye una aplicación embebida en ROS2, tres cámaras estéreo y una interfaz gráfica. El proceso comienza con la adquisición de imágenes por parte de las tres cámaras, las cuales se utilizan para generar una imagen panorámica. Esta imagen panorámica es luego procesada por una red neuronal, que identifica las regiones donde se encuentran los vehículos, determinando tanto el recuadro donde se encuentra el vehículo, como el centroide de cada vehículo detectado.

Luego, se lleva a cabo el seguimiento de cada vehículo detectado y sus centroides calculados. Utilizando el mapa de disparidad de la imagen panorámica, se estiman las distancias relativas al centroide de cada vehículo. Con estas distancias estimadas, se calcula la velocidad relativa de los vehículos. Posteriormente, mediante filtros de Kalman, se predice la trayectoria y se genera un porcentaje de probabilidad de colisión. Esta información se muestra a través de la interfaz gráfica del aplicativo, emitiendo una alerta en caso de riesgo elevado.

1.4 ESTRUCTURA DEL LIBRO

En el transcurso de este documento se detallan los capítulos que abordan los objetivos específicos planteados y se describe el proceso a través del cual se alcanzaron dichos objetivos.

- En el primer capítulo del libro se aborda la introducción detallada donde se establece el problema a abordar, los objetivos del proyecto, la solución propuesta, así como una breve descripción de la estructura del libro.
- En el capítulo 2, siguiendo lo establecido con el objetivo 1, se presenta la revisión bibliográfica que presenta las definiciones y fundamentos teóricos que son la base para este trabajo, dado que se abordan de forma detallada los conceptos que integran el proyecto, como lo es la visión estéreo, los filtros Kalman, las redes neuronales convolucionales y los métodos de estimación de colisión con otros vehículos. Así mismo se aborda y discute sobre los trabajos previos relacionados al proyecto, aquí se presentan trabajos que buscaban implementar este tipo de sistemas de alerta de colisión usando diferentes métodos para alertar y procesar las imágenes.
- El capítulo 3 llamado “Desarrollo de un aplicativo embebido de alerta de colisión con otros vehículos” detalla cómo se implementó la solución, empezando desde el diseño teniendo en cuenta la metodología RUP, se presentan los requerimientos funcionales y no funcionales, así como sus diagramas de concepto, clases y nodos. De igual forma se presenta la implementación del aplicativo en su módulo de configuración y su módulo de procesamiento.
- Para el capítulo 4 se relata y describe los resultados obtenidos de las pruebas de integración y de campo del aplicativo embebido. Del mismo modo se presentan las métricas de evaluación y los análisis de los resultados junto con sus respectivas conclusiones. Por último, el capítulo 5 plantea las conclusiones del trabajo, plantea perspectivas para trabajos futuros y observaciones.

2. ANTECEDENTES Y MARCO TEÓRICO

2.1 INTRODUCCIÓN

En este capítulo se abordarán los distintos trabajos relacionados al área de asistencia al conductor, especialmente en la alerta de colisión, estos se organizaron en una tabla donde se comparan los sensores usados, los métodos de detección, el sistema embebido donde se prueba, el tipo de detección, el porcentaje de éxito en la prevención de colisiones, entre otras. Así mismo se discuten más a profundidad cada uno de los trabajos.

Además, en este capítulo se introducen los conceptos necesarios para entender lo que se quiere abordar en los capítulos posteriores, esto con el fin de que el lector pueda seguir correctamente el desarrollo del trabajo, entendiendo con claridad lo que se aborda en cada sección. El marco teórico trata aspectos teóricos fundamentales involucrados en el proyecto, como lo es la visión estero, las redes neuronales convolucionales, los filtros Kalman, la estimación de colisión con otro vehículo, entre otras.

2.2 ANTECEDENTES

Los sistemas de alerta de colisiones son fundamentales en la navegación segura del conductor en la carretera. Debido a esto es importante que cumplan con ciertas características significativas para ser efectivos, como lo es la capacidad de respuesta en tiempo real, el desempeño satisfactorio tanto de día como de noche, la detección de objetos estáticos y dinámicos, la capacidad de seguimiento de objetos, la capacidad de predicción en los algoritmos, el tiempo de procesamiento de las imágenes, entre otras

Teniendo en cuenta las necesidades mencionadas anteriormente se analizaron trabajos relevantes que se presentan en la tabla 2-1. Las columnas de dicha tabla incluyen información sobre el sistema embebido, el tipo de detección, el sensor usado, el método de detección, software, realimentación al usuario y porcentaje de éxito de la alerta ante posibles colisiones.

Tabla 2-1. Trabajos previos clasificados según sus puntos de interés

REFERENCIA	SISTEMA EMBEBIDO	TIPO DE DETECCIÓN	SENSOR	MÉTODO DE DETECCIÓN	SOFTWARE USADO	REALIMENTACIÓN AL USUARIO	PORCENTAJE DE ÉXITO
[8]	N/A	Colisiones en la parte trasera.	Cámara mono.	Aprendizaje profundo y filtrado. Red neuronal convolucional	N/A	Alerta de sonido	77,75%
[9]	N/A	Colisiones en la parte frontal.	Cámara mono.	Aprendizaje profundo. API Google Tensor Flow Object Detection.	Programado en Python	Alerta visual	N/A
[11]	N/A	Colisión en la parte frontal	Cámara black box integrada al vehículo	Extracción de características y filtrado	N/A	N/A	87.5%
[12]	Nvidia Jetson TX2	Detección de invasión de carril.	Cámara mono.	Aprendizaje profundo. YOLO V3 (You Only Look Once)	Programado en Python	Alerta visual	80.4%
[13]	Raspberry pi 3	Colisiones en la parte frontal.	Cámara Raspberry pi.	Segmentación y filtrado	N/A	Alerta de sonido	86.8%
[14]	PC móvil (i5 CPU de 2.1Ghz)	Colisiones en la parte trasera	Cámara Go pro clip on.	Extracción de características (flujo óptico)	Matlab	Alerta visual y de sonido	73.3%
[15]	PC (I7 CPU de 3.5Ghz, GPU NVIDIA GTX650 Y 4GB RAM)	Colisión en la parte frontal	Cámara de visión estéreo.	Aprendizaje profundo, segmentación y filtrado. Red neuronal convolucional	N/A	Alerta visual	N/A
[16]	CPU Intel i7 de 4.0 GHz y GPU Nvidia GTX 1080	Colisión en la parte frontal y adelantamiento de vehículos	Cámaras DVR	Aprendizaje profundo, extracción de características y filtrado. Red neuronal convolucional	N/A	Alerta visual	96,3%

[17]	Nvidia MX 450, 2G memoria Intel(R) Core i7- 1165G7 CPU 2.8GHz, 16Gb memoria ram, Windows10	Colisión frontal	Cámara frontal mono	Aprendizaje profundo YOLO v3	Programado en Python en conjunto con OpenCV (usaron base de datos KITTI y una privada)	Alerta visual	94.04% KITTI 92.42% Base de datos privada
[18]	NVIDIA Jetson AGX with 8 CPU cores, 512 GPU cores and 32 GB of RAM	Colisión frontal	Cámara HDR 2MP Starlight -Radar ARS 404-21	Aprendizaje profundo YOLO v3	Programado en Matlab para luego usar C	Alerta visual y auditiva	N/A
[19]	Celular Android	Colisión frontal	Cámara del teléfono	Aprendizaje profundo MobileNet basado en depthwise separable convolution	N/A	Alerta visual	70.6%
[20]	Raspberry Pi 4 Modelo B con 8 GB de RAM	Accidentes de T-bone, colisiones traseras y frontales	Cámara Raspberry Pi NoIR V2	Aprendizaje profundo MobileNet DLA	Programado en Python	Alerta visual	N/A
[21]	Computadora portátil Intel Core i7 de 3,4 GHz, 4 GB de RAM tarjeta gráfica NVIDIA GeForce GTX 560	Colisiones traseras	Cámara estéreo ojo de pez	Coincidencia estéreo basada en HCT	N/A	Alerta visual	95%

[22]	Nvidia Tesla C1060	Colisiones frontales	Cámara estero	Mapa de disparidad densa, usando coincidencia de bloques y segmentación basado en etiquetado de píxeles conectados	API Nvidia CUDA	Alerta visual	N/A
[23]	GPU TITAN Xp	Colisiones frontales	Cámara mono	Aprendizaje profundo (YOLO) y filtros Kalman	N/A	Alerta visual	N/A

Los trabajos revisados fueron extraídos de las bases de datos IEEE Xplorer y Scopus, reconocidas por su amplio contenido en ingeniería. Cabe mencionar que no se encontraron antecedentes del tema en Colombia. Durante la revisión, los sistemas de alerta de colisiones se clasificaron en dos grupos principales según el tipo de colisión que previenen: frontal y trasera, siendo estas las implementaciones más comunes. Las diferencias entre los estudios radican en los métodos de detección y los sistemas embebidos utilizados.

En los trabajos [8] y [15], se emplean sistemas basados en redes neuronales convolucionales CNNs, (por sus siglas en inglés). No obstante, difieren en su objetivo de aplicación: mientras que en [8] se utilizan para estimar la distancia entre nuestro vehículo y el vehículo precedente para mantener una distancia segura, en [15] se emplean para clasificar los carriles y detectar la presencia de vehículos que puedan incorporarse en el carril del vehículo principal. Cuando se realizaron las pruebas de validación en [8], se logró obtener una precisión promedio de 77.75% en la medición de las distancias entre los vehículos.

Para este último se utilizó una cámara estéreo para mejorar la percepción del entorno. A partir de las imágenes captadas, se desarrolló un método robusto de detección y clasificación de carriles, resistente a baches y cambios en la vía, mediante regiones de interés específicas. Además, se empleó una red neuronal convolucional para identificar las marcas viales, determinar el área de conducción y estimar la distancia de colisión frontal en entornos estructurados. Cabe señalar que el software se montó en un PC, lo que puede afectar a la portabilidad del sistema. En cada proyecto se usó una base de datos diferente, para [8] crearon su propia base de datos y para [15] usaron la base de datos públicos KITTY [15] y una propia que ellos desarrollaron para realizar comparaciones entre los resultados de ambas.

Algunos trabajos como [16] también integran métodos clásicos de extracción de características y filtrado con métodos de aprendizaje profundo como las redes neuronales convolucionales. En este se busca prevenir accidentes por colisiones frontales y adelantamientos, mediante detección de cambio de carril, advertencia de colisión e identificación de vehículos que adelantan. Propone un método de detección frontal basado en simetría y función de densidad acumulativa (CDF), y usa flujo óptico junto con redes neuronales convolucionales para mejorar la detección precisa de adelantamientos[16]. En sus resultados lograron una precisión del 96.3% en la detección de adelantamiento.

En otras implementaciones, como las presentadas en [9] y [12], han utilizado herramientas de machine learning. En el primer caso se ha empleado la API de Google Tensor Flow, mientras que en el segundo se utilizó YOLO V3-Tiny. Para la API de Google Tensor Flow se ha llevado a cabo un entrenamiento con el conjunto de datos Microsoft Common Objects in Context (COCO) [9], utilizando el modelo "ssd_mobilenet_v1_coco" por su eficiencia. En esta implementación, el objetivo es evitar colisiones frontales, estableciendo una distancia umbral que alerte al usuario cuando se acerca demasiado al vehículo delantero.

En [12] se utilizó YOLO entrenado con el dataset "COCO" para detectar automóviles y reducir el tiempo de procesamiento. La estimación de profundidad se basa en las dimensiones del objeto detectado y se usa regresión múltiple para calcular la distancia. Posteriormente, se aplican algoritmos de asignación y seguimiento de carriles, logrando un 80,4 % de precisión en la estimación de distancia y un 93 % en la detección de carriles.

Por otro lado, los trabajos [11], [13] y [14] emplean herramientas convencionales para el procesamiento de imágenes y la visión por computadora, como técnicas de filtrado, segmentación y extracción de características, prescindiendo del uso de la inteligencia artificial.

En [11], se presenta un método que utiliza una única cámara de caja negra instalada en el vehículo para detectar el entorno y estimar su distancia con otros vehículos. Esto lo logran mediante el algoritmo AdaBoost basado en ACF [11] para detectar la región del vehículo y estimar la distancia utilizando la transformación de perspectiva inversa de forma simétrica y así extraer las características más importantes de las imágenes. Las pruebas se realizaron en entornos de carretera para los cuales se estimó que la precisión de la detección del vehículo era del 87,5%, la precisión de la estimación de la distancia era del 92,8% y el tiempo necesario para el procesamiento era de 0,76s por fotograma, aunque el sistema presenta problemas cuando la sombra de árboles, señales de tránsito o farolas se solapan con los vehículos en circulación.

En [13] el sistema se implementó en una tarjeta Raspberry pi 3, usando una cámara Raspberry pi. El proceso de alerta consta de cinco pasos: captura de imágenes con el vehículo en movimiento, segmentación de la región de interés, detección de vehículos considerando sombras en el día y luces traseras en la noche, y generación de la alerta. El sistema alcanzó una precisión promedio del 86,8 %, demostrando su fiabilidad para advertir colisiones.

En [14] desarrollan un algoritmo que utiliza la combinación de técnicas de segmentación de peatones y vehículos a partir del fondo en movimiento, aprovechando el valor de flujo óptico local. Además, utiliza un método adaptativo de escala que se basa en el modelo de pieza deformable para detectar objetos a diferentes distancias. En las pruebas el método desarrollado por el grupo logro una precisión del 73.33% con un tiempo de procesamiento de 0.63s, logrando identificar personas y vehículos. Todo esto desarrollado en Matlab e implementado en un PC portátil.

En [17] usan métodos de aprendizaje profundo para detectar los vehículos a partir de las imágenes que se obtienen de la cámara frontal, determinar la situación de riesgo y finalmente activar los frenos del vehículo. Para el sistema usan dos modelos de red neuronal, uno para detección de los objetos y otro para estimar distancias; YOLOv3 (basado en Darknet) para detección de objetos y DISNET para estimar distancias; que es un esquema de aprendizaje profundo distribuido micro dividido para IoT. Ya con las distancias obtenidas lo comparan con un umbral definido que permite decidir si se está o no en una situación de riesgo. El sistema logro una precisión en la detección del 94.4% con la base de datos KITTI y un 92.42% para la base de datos local.

En [18] se usa también la red neuronal YOLOv3 para la detección de los vehículos, esta red neuronal suele ser muy usada ya que su procesamiento es veloz y bastante preciso, se diferencia de [17] en el hecho de que usa el algoritmo Global Nearest Neighbor (GNN) con un banco de filtros lineales de Kalman para realizar el seguimiento a los vehículos. Así mismo debido a que pueden ocurrir situaciones de conflicto, se debe definir una función de costos y resolver un problema de optimización en cada paso de tiempo, el problema de optimización se resuelve utilizando el algoritmo de Munkres.

Un modelo diferente se propone en [19] donde no usan un hardware demasiado potente, en su lugar usan un celular para mejorar la portabilidad y facilidad de instalación. Como el sistema se desea implementar en un celular, se propone utilizar **MobileNets**, un modelo de aprendizaje profundo optimizado y ligero que emplea *depth-wise separable convolutions*. Esta arquitectura incluye dos hiperparámetros que permiten ajustar el equilibrio entre latencia y precisión, facilitando la elección del modelo adecuado según las limitaciones del problema. Con este enfoque, se logra detectar vehículos y estimar su distancia.

En [20] se propone un sistema de prevención de colisiones tipo T-bone, traseras y frontales, utilizando solo una cámara Raspberry Pi NoIR V2 y una Raspberry Pi 4 modelo B sin hacer uso de sensores adicionales. El procesamiento se lleva a cabo en la Raspberry Pi 4, por lo que se emplean los modelos “MobileNet” y SSD (Single Shot MultiBox Detector), debido a su buen equilibrio entre precisión y velocidad. Para estimar distancias relativas, el sistema considera dos posiciones y las variaciones en la altura del mismo objeto, teniendo en cuenta que estas cambian a medida que la cámara se aproxima o se aleja.

En [21] se utilizan técnicas de procesamiento de imágenes estéreo, proponiendo el uso de la coincidencia estéreo basada en la Transformación Jerárquica del Censo (HCT). Antes de aplicar este método, se realiza un preprocesamiento para optimizar el rendimiento. Las cámaras estéreo ojo de pez se calibran con tablas de búsqueda (LUTs) y las imágenes se transforman en vistas estenopeicas y de vista superior. Luego, se identifica la región candidata a obstáculo (OCR) a partir de los bordes verticales en la imagen en color, se eliminan las texturas de la carretera, y finalmente se aplica la coincidencia HCT solo sobre la OCR, reduciendo así el tiempo de procesamiento y el uso de memoria en la estimación del mapa de profundidad.

Así mismo en [22] se usan métodos de procesamiento de imágenes estéreo, para este caso el procesamiento viene de la siguiente manera primero se calcula el mapa de disparidad densa utilizando el algoritmo de coincidencia de bloques. Luego dicho mapa se segmenta utilizando un método basado en etiquetado de píxeles conectados, ya con los objetos segmentados y etiquetados se comprueba si están a una distancia segura o de riesgo. El sistema se probó en una Nvidia Tesla C1060 programada con la API Nvidia CUDA, logrando un tiempo de procesamiento de 58,3 ms.

En [23] se combina el aprendizaje profundo con los filtros Kalman para estabilizar las mediciones, ya que se utiliza una cámara monocular, la información proporcionada será limitada. En el sistema propuesto YOLO se encarga de detectar los vehículos, y la distancia se estima a partir del ancho detectado, aplicando un filtro Kalman para estabilizar este valor. Además, se usa un segundo filtro Kalman para estabilizar la velocidad relativa, y con estos datos se calcula el tiempo de colisión.

Tras revisar los desarrollos en el área, se destacan tres aspectos clave en los sistemas de alerta de colisión: la calidad del sensor, que define la información del entorno; el método de extracción y seguimiento, que puede mejorar la precisión al combinarse con otras técnicas; y la necesidad de un sistema embebido potente, que permita generar respuestas con una latencia mínima, de modo que las alertas se emitan en el momento preciso.

2.3 MARCO CONCEPTUAL

En esta sección se presentarán algunos conceptos necesarios para comprender de manera correcta el funcionamiento, los tipos y las características de los sistemas tipo ADAS y automatización de los vehículos. Además, se presentará lo que es la visión estéreo. las técnicas más utilizadas para el procesamiento de las imágenes estereoscópicas, junto con el método más usado para la calibración de cámaras estéreo, así como lo que son las redes neuronales convolucionales

2.3.1 Advanced Driver Assistance Systems (ADAS)

Son tecnologías diseñadas para ayudar al conductor a obtener la información que necesita, automatizar ciertas tareas y mejorar la seguridad general del vehículo. Estas innovaciones mejoran la experiencia de conducción y mejoran la seguridad general en la carretera [24]. Funcionan con una amplia gama de tecnologías como cámaras, radares y sensores láser con los cuales se percibe el entorno que rodea al vehículo para así identificar y evitar posibles peligros, alertando al conductor o accionando automáticamente un elemento del vehículo.

2.3.1.1 Tipos de sistemas ADAS

En el estudio [25], se establece que los sistemas de Asistencia Avanzada al Conductor (ADAS) se dividen en dos categorías principales: pasivos y activos. Estas categorías se determinan en función de cómo responden ante situaciones de riesgo en el entorno del vehículo. Los sistemas ADAS pueden tomar medidas directas para controlar el vehículo o, alternativamente, solo enviar alertas al conductor.

- **Sistema activo ADAS:** Interviene en el vehículo para evitar riesgos, como frenar automáticamente o realizar estacionamiento asistido.
- **Sistema pasivo ADAS:** Solo alerta al conductor sobre posibles peligros sin controlar el vehículo, mediante señales sonoras, visuales o táctiles.

2.3.1.2 Niveles de automatización

El grupo NHTSA (National Highway Traffic Safety Administration) señala que, en la actualidad, incluso los vehículos con el nivel más avanzado de automatización disponibles para los consumidores requieren de la atención total de los conductores. A pesar de los considerables esfuerzos realizados en pruebas de seguridad, desarrollo y validación de estos sistemas automatizados de conducción [1].

Nivel 0: El conductor humano se encarga completamente de la tarea de conducir.

Nivel 1: El sistema avanzado de asistencia al conductor (ADAS) puede brindar una asistencia en el frenado/aceleración o la dirección, pero no puede hacerlo de forma simultánea.

Nivel 2: Control combinado de dirección y aceleración/frenado en ciertas situaciones, con supervisión constante del conductor.

Nivel 3: Conducción automatizada con conductor disponible para intervenir si es necesario.

Nivel 4: Conducción totalmente automatizada en zonas limitadas, sin necesidad de conductor activo.

Nivel 5: Conducción totalmente automatizada sin restricciones de zona, sin conductor humano necesario.

2.3.2 Visión Estéreo

La visión por computador busca emular la percepción visual humana, permitiendo que dispositivos capturen imágenes para interpretarlas de modo que puedan analizarlas y dar sentido a dichas imágenes [26]. La percepción de profundidad en humanos, llamada estereopsis, se basa en la diferencia de imágenes entre ambos ojos para determinar la posición de objetos [27]. Este es un proceso natural para el ser humano pero complejo para las máquinas.

En la visión por computadora, específicamente la visión estéreo se busca capturar la escena del mundo 3D por dos cámaras que están posicionadas a una distancia horizontal paralela conocida, obteniendo una imagen 2D en cada cámara y a partir de las cuales se reconstruye la escena 3D [26].

La visión estereoscópica artificial se basa en el modelo biológico de la visión estereoscópica. En estos sistemas, el desplazamiento de los ojos permite determinar la profundidad de los objetos mediante la triangulación de las dos imágenes de la misma escena 3D captadas por cada ojo. Esto es posible porque, al estar los ojos separados, las imágenes de los objetos se ven desplazadas dependiendo de su distancia a los ojos [26] Así mismo se puede lograr esto usando dos cámaras unidas cuyo desplazamiento mutuo es conocido, véase la Figura 2-1:



Figura 2-1. Ejemplo cámara estéreo. Modelo ZED2 de la empresa StereoLabs (<https://store.stereolabs.com/products/zed-2>).

La visión estéreo en computadoras consiste en encontrar puntos coincidentes entre dos cámaras para calcular la posición 3D de esos puntos, usando la distancia entre cámaras. Sin información sobre la geometría del sistema, esta búsqueda es costosa computacionalmente, pero conocer la configuración de las cámaras permite reducir el proceso y hacerlo más eficiente. Por lo que, en la práctica, la visión estéreo, suele englobar los siguientes pasos [27]:

1. Corregir matemáticamente la distorsión de la lente para que las imágenes reflejen correctamente las líneas rectas del mundo real.
2. Ajustar la orientación y posición de las cámaras mediante la rectificación, alineando las imágenes (Figura 2-2).
3. Identificar puntos coincidentes en ambas imágenes (correspondencia) y calcular la disparidad, es decir, la diferencia en la posición horizontal de esos puntos. El resultado de este proceso son las disparidades de los puntos de interés, donde disparidad significa la diferencia de la coordenada x de un punto de interés en las imágenes de la cámara izquierda y derecha: $x_l - x_r$ (Figura 2-2).
4. Usar la geometría de las cámaras para calcular la distancia y posición 3D de los puntos mediante triangulación (re-proyección) (Figura 2-2).

El obtener imágenes rectificadas es importante porque la rectificación consigue que las filas de las imágenes de la cámara izquierda y derecha estén alineadas, por lo que un punto de interés en una fila de la imagen izquierda estará en la misma fila de la imagen derecha [27].

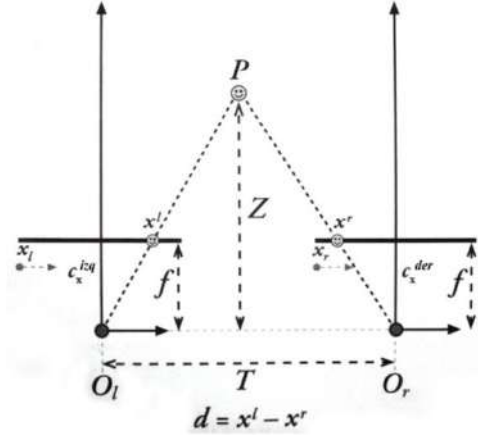


Figura 2-2. Par estéreo reajustado, rectificado y con una correspondencia conocida, donde la distancia Z se puede encontrar mediante triángulos semejantes.

Asumiendo que las imágenes están correctamente reajustadas, rectificadas y la distancia T entre las cámaras es conocida, como se ve en la Figura 2-2. Tenemos entonces dos cámaras cuyos planos de las imágenes son perfectamente coplanares, sus ejes ópticos paralelos y a una distancia conocida, con longitudes focales iguales $f_r = f_l$. Si también se asume que los puntos principales c_x^{izq} y c_x^{der} (los puntos en donde los rayos principales intersectan con los planos de imagen) han sido calibrados para tener las mismas coordenadas en píxeles tanto en la imagen izquierda como en la derecha y que se puede encontrar un punto P en el mundo físico con proyecciones p_l y p_r en las imágenes, cuyas coordenadas en el eje horizontal son respectivamente x_l y x_r . Entonces, se puede obtener la distancia Z como se muestra en la ecuación 2.1 [27]:

$$\frac{T - (x_l - x_r)}{Z - f} = \frac{T}{Z} \rightarrow Z = \frac{fT}{x_l - x_r} \quad (2.1)$$

Donde el valor $d = x_l - x_r$ es conocido como disparidad.

En el mismo escenario, se define la matriz de re-proyección Q mostrada en la ecuación 2.2

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c'_x)/T_x \end{bmatrix} \quad (2.2)$$

Donde, se hace referencia a los parámetros de la imagen izquierda, incluyendo c_x y c_y que representan el punto principal, f que es la longitud focal de ambas cámaras (asumiendo que es la misma en ambas) y T_x que indica la distancia horizontal entre los centros de las cámaras. Sin embargo, c'_x se refiere a la coordenada x del punto principal en la imagen derecha [27].

Si se tienen las coordenadas homogéneas de un punto de interés en la imagen izquierda junto con su disparidad d respecto a su punto correspondiente en la imagen derecha, la matriz Q proporciona el cálculo de las coordenadas tridimensionales en el mundo físico del punto de interés como se muestra en la Ecuación 2.3.

$$Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \quad (2.3)$$

Donde finalmente las coordenadas en 3D del punto de interés son $(X/W, Y/W, Z/W)$. Al calcular la disparidad d en cada punto de la imagen se obtiene lo que se conoce como mapa de disparidad.



Figura 2-3. Mapa de disparidad formado a partir de imágenes estéreo rectificadas.

Al ver esta representación del mapa de disparidad se puede ver como los puntos más cercanos a la cámara tienen un color gris más claro, lo que quiere decir que el valor de la disparidad es más grande. Por otro lado, los puntos más alejados de la cámara tienen un tono de gris más oscuro, es decir, la disparidad es menor.

2.3.2.1 Calibración estereoscópica

La calibración estéreo busca establecer la relación geométrica entre dos cámaras para obtener los parámetros necesarios (intrínsecos y extrínsecos) que permiten extraer información 3D de una escena. Para ello, se determinan la matriz de rotación RRR y el vector de traslación TTT , conocidos como los parámetros extrínsecos del sistema estéreo (Figura 2-4) [28].

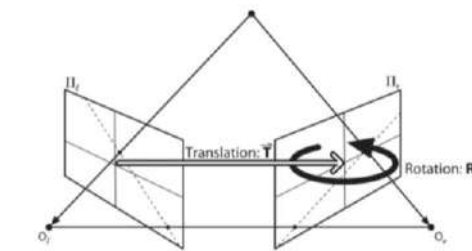


Figura 2-4. Parámetros extrínsecos de la calibración estéreo.

Para empezar el proceso de calibración es necesario seleccionar un patrón que puede ser el de ajedrez o de esferas, normalmente se usa el patrón de ajedrez (Figura 2-5) ya que es muy fácil detectar las esquinas con algún algoritmo de visión por computadora (Harris, Moravec, Shi-Tomasi, entre otros) y geometría es bastante simple.

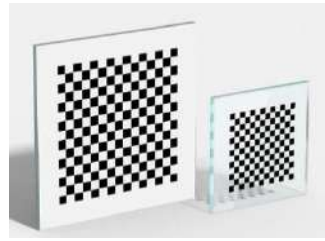


Figura 2-5. Patrón de ajedrez.

El siguiente paso en la calibración consiste en detectar esquinas en pares de imágenes estéreo para segmentar el patrón de cuadros y usar estas esquinas como puntos de correspondencia. Luego, con estos puntos se realiza la rectificación, que alinea horizontalmente las imágenes mediante una deformación geométrica, facilitando así la calibración [28].

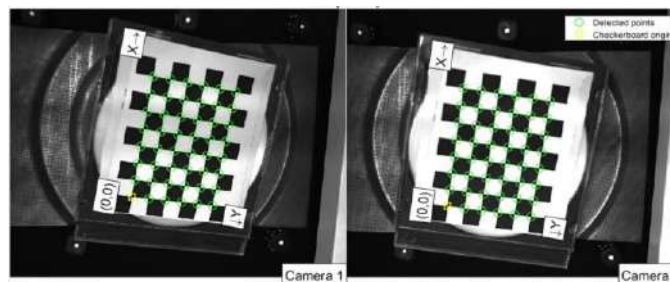


Figura 2-6. Detección de esquinas para posterior rectificación.

Como resultado del proceso de calibración se obtiene la matriz de re-proyección Q . Esta matriz contiene los parámetros intrínsecos y extrínsecos del sistema y con ellos se puede llevar a cabo una reconstrucción tridimensional para obtener medidas de profundidad [28].

2.3.2.2 Procesamiento de imágenes estéreo

El procesamiento de imágenes estereoscópicas permite conocer la tridimensionalidad del entorno, esencial para que una máquina navegue adecuadamente. A través de la visión estereoscópica se obtiene la forma de los objetos midiendo su distancia respecto a un sistema de referencia, lo que permite reconstruir la tercera dimensión de la escena[29]. Normalmente realizan esta actividad comparando la correspondencia entre diferentes puntos de la imagen.

En la visión estereoscópica se pueden distinguir dos grupos de técnicas para establecer la correspondencia a partir de dos imágenes estereoscópicas: basadas en el área y basadas en las características [29].

Técnicas basadas en el área

Las técnicas que se basan en el área utilizan los patrones de intensidad en la vecindad cercana de un píxel en una imagen (Figura 2-7) para comparar con los patrones de intensidad en una vecindad similar de un píxel en la otra imagen del par estereoscópico. Para determinar la similitud se emplean medidas estadísticas o distancias métricas entre píxeles. Un ejemplo de medida estadística sería la correlación [30].

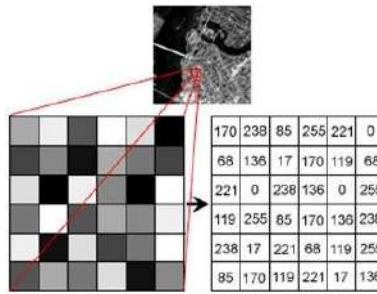


Figura 2-7. Vecindad de píxeles de una imagen en escala de grises.

Técnicas basadas en las características

Las técnicas basadas en características usan estructuras más complejas que los píxeles individuales, como bordes o regiones, para encontrar correspondencias entre imágenes. Generalmente, agrupan píxeles con atributos similares y pueden seguir un enfoque jerárquico, comenzando con bordes o esquinas y luego extendiéndose a regiones completas. [30].

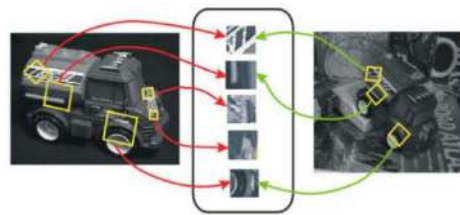


Figura 2-8. Ejemplo de procesamiento de imágenes por extracción de características.

2.3.3 Fusión de imágenes (Stitching)

La fusión de imágenes consiste en integrar varias capturas en una sola imagen de mayor tamaño, comúnmente llamada panorama. Estas imágenes corresponden a una misma escena tridimensional y requieren solaparse en ciertas áreas. El propósito del proceso es lograr una transición continua entre las imágenes contiguas, manteniendo tanto la coherencia geométrica como la calidad visual [31].

Los principales pasos involucrados en el proceso de **unión de imágenes** son los siguientes:

1. **Captura de imágenes:** Se obtienen varias tomas de la misma escena con zonas de solapamiento, generalmente manteniendo la misma orientación y parámetros de exposición.
2. **Detección de puntos característicos:** Se identifican elementos relevantes en cada imagen (esquinas, bordes o patrones). Para ello suelen emplearse algoritmos como **SIFT**, **SURF** u **ORB**.
3. **Emparejamiento de características:** Se buscan coincidencias entre los puntos detectados en las áreas superpuestas, lo que permite alinear correctamente las imágenes.

4. **Estimación de homografía:** Se calcula una matriz de transformación que indica cómo ajustar una imagen para que coincida con la perspectiva de otra.
5. **Deformación y alineación:** A partir de la homografía, se transforman las imágenes hacia un mismo sistema de coordenadas para lograr un solapamiento preciso.
6. **Fusión de imágenes:** Se integran las zonas comunes suavizando las transiciones. Para ello pueden utilizarse técnicas como difuminado, fusión multibanda o corrección de exposición.
7. **Generación del panorama:** Finalmente, se combinan todas las imágenes alineadas y fusionadas en una sola composición panorámica, aplicando ajustes de recorte, color y exposición para obtener un resultado uniforme [32].

2.3.4 Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (CNN), abreviatura en inglés de Convolutional Neural Networks, son un tipo de redes neuronales artificiales ampliamente conocidas por su destacado desempeño en el reconocimiento de imágenes. Su superioridad radica en su capacidad para identificar patrones y clasificar imágenes de manera eficaz. Dentro de este tipo de red neuronal la capa de convolución es la más importante pues funciona como un grupo de filtros, que al ser convolucionados con una cierta entrada (o input) generan un mapa de características en la salida (output) (ver Figura 2-9).

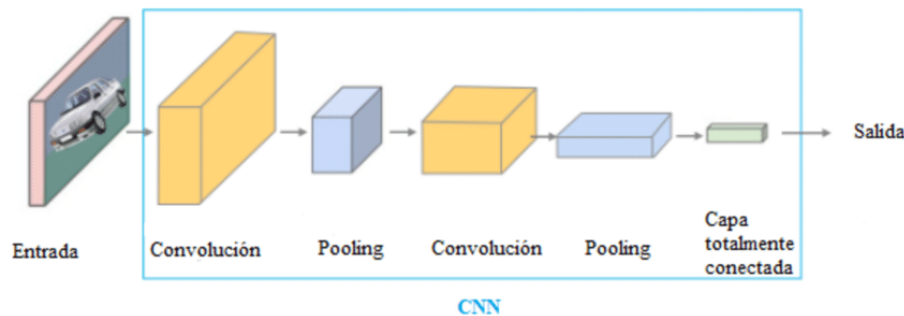


Figura 2-9. Estructura básica de una red neuronal convolucional.

Las CNNs toman un dato, por ejemplo, una imagen y van aplicando filtros a secciones específicas en lugar de a la imagen completa. A medida que avanzan a través de las capas, estas redes neuronales identifican patrones simples, como líneas verticales o curvas, y los combinan para reconocer patrones más complejos, como la forma de un automóvil en una imagen. Este proceso gradual de identificación de patrones ayuda a simplificar la información para un procesamiento más eficiente [33]. Este tipo de redes neuronales se conforman principalmente de 3 capas: la capa convolucional, la capa de agrupación (o pooling) y la capa totalmente conectada (Figura 2-9).

Capa convolucional: En la primera capa de una red convolucional se realizan la mayoría de los cálculos, utilizando tres componentes clave: la entrada (una imagen, por ejemplo, de tamaño alto $\times$ ancho $\times$ 3 canales RGB), el filtro (o kernel) y el mapa de características. El filtro, usualmente una matriz 3×3 , recorre la imagen realizando productos escalares con los píxeles de cada región. Estos resultados se almacenan en una nueva matriz llamada mapa

de características o de activación, que representa las zonas donde se detectan ciertos patrones en la imagen [34].

Después de cada operación de convolución, la CNN aplica una transformación de unidad lineal rectificadora (ReLU) al mapa de características, introduciendo la no linealidad en el modelo [34]. También en esta capa se le pueden añadir más capas convolutivas a la salida, convirtiendo la estructura de la CNN en jerárquica. Por ejemplo, para identificar una bicicleta en una imagen, una red neuronal convolucional analiza sus partes (ruedas, manillar, pedales) como patrones básicos. Al combinar estos componentes, forma un patrón más complejo, creando una jerarquía de características en distintos niveles dentro de la red. [34].

Capa de agrupación: esta capa reduce la dimensión del mapa de características aplicando un filtro sin pesos, es decir, que no detecta características específicas, sino que agrupa valores en la salida. El diseñador de la red puede elegir entre dos métodos: agrupación máxima, que selecciona el valor más alto en un área, o agrupación media, que calcula el promedio. La agrupación media es la más utilizada [34].

Capa totalmente conectada: su nombre describe con precisión lo que es la capa en sí, a diferencia con las capas anteriores en esta no hay un filtro o kernel que separe directamente la entrada con la salida, cada nodo de la capa de salida sí está conectado directamente a un nodo de la capa anterior. Esta capa realiza la tarea de clasificación basándose en las características extraídas de las capas anteriores y sus diferentes filtros, “generalmente usa una función de activación softmax para clasificar adecuadamente las entradas y generar una probabilidad entre 0 y 1” [34].

2.3.5 Seguimiento de Objetos (Object Tracking)

En términos simples, el seguimiento de objetos consiste en estimar su trayectoria en una imagen mientras se mueven por la escena, asignándoles etiquetas coherentes a lo largo de varios fotogramas. Puede incluir información adicional como tamaño, forma u orientación. Sin embargo, este proceso es complejo por factores como proyección 3D a 2D, ruido, oclusiones, movimiento irregular, iluminación variable y la necesidad de procesamiento en tiempo real. [35].

El seguimiento puede darse a un solo objeto o a múltiples objetos, para este trabajo el enfoque está en el seguimiento de múltiples objetos en una escena, por lo que analizaremos algunos de los algoritmos de seguimiento orientados a múltiples objetos. El seguimiento de múltiples objetos suele abordarse mediante un enfoque en dos etapas. En la primera etapa, denominada **detección de objetos**, se determina la posición y las categorías de los elementos relevantes dentro de cada fotograma del video. Posteriormente, se asignan identificadores únicos (ID's) a cada objeto detectado para diferenciarlos y darles continuidad a lo largo de la secuencia [36].

La segunda etapa, conocida como **asociación de instancias**, integra información temporal proveniente de múltiples fotogramas para trazar las trayectorias de cada objeto. Su propósito fundamental es asegurar una asignación coherente de identificadores, de modo que cada objeto conserve su etiqueta única a lo largo de toda la secuencia de video [36].

La asociación de instancias se puede lograr mediante dos formas distintas. En la primera se puede realizar usando señales de movimiento para vincular detecciones con *tracklets*, normalmente con un **Filtro de Kalman**. Este filtro predice la posición futura de los objetos, y luego las nuevas detecciones se emparejan con esas predicciones usando la métrica de **intersección sobre unión (IoU)**. [36].

El segundo método usa las características visuales dentro de los cuadros delimitadores para emparejar objetos entre fotogramas. Para ello, se emplea una red neuronal que extrae dichas características y las compara con una métrica, como la similitud del coseno. Su principal ventaja es que no depende de la posición, lo que lo hace útil ante cambios bruscos o desapariciones temporales del objeto [36].

En la actualidad se han creado algoritmos de seguimiento basados en aprendizaje profundo para el seguimiento de múltiples objetos y en esta sección se abordarán algunos de los más conocidos y resaltantes que son: SORT, DeepSORT, StrongSORT, BYTETrack y BOTSORT.

SORT: (Simple Online and Real-time Tracking) es un algoritmo que utiliza un filtro de Kalman para predecir la posición futura de los objetos y ajustarla con nuevas detecciones usando la métrica IoU. Su diseño ligero permite un procesamiento rápido, ideal para aplicaciones de seguridad en tiempo real [37].

DeepSORT: es una mejora de SORT que combina un detector de objetos basado en redes neuronales profundas con un sistema de seguimiento. Usa un filtro de Kalman y el algoritmo húngaro para asociar detecciones entre fotogramas, y añade una red neuronal convolucional (CNN) para mejorar la precisión en escenarios complejos y con múltiples objetos [38].

StrongSORT: es una mejora de DeepSORT que busca mayor precisión y estabilidad en el seguimiento de objetos. Incorpora el modelo **AFLink** (AFLink, Appearance Free Link), que enlaza trayectorias cortas sin depender de características visuales, y **GSI (Interpolación Suavizada Gaussiana)**, que suaviza y completa trayectorias cuando hay pérdida temporal de detecciones. Esto permite obtener seguimientos más continuos y precisos en escenarios complejos [38].

BYTETrack: BYTE es un método de seguimiento múltiple que mejora la asociación de detecciones entre cuadros al considerar todas las detecciones, incluso las de baja puntuación, evitando perder objetos superpuestos o débiles. Utiliza tracklets, que son secuencias cortas de detecciones de un mismo objeto, para mejorar la identificación y evitar confundir objetos con ruido de fondo [38].

BOTSORT: es una versión mejorada de los algoritmos SORT y DeepSORT para el seguimiento de múltiples objetos. Aumenta la precisión y estabilidad al reducir errores de reasignación de ID. Utiliza un filtro de Kalman con modelo de movimiento de segundo orden para seguir trayectorias no lineales y asocia detecciones con **IoU**. Cuando las detecciones son insuficientes, emplea **Re-ID (Re-identification)** mediante redes neuronales para mantener el rastreo incluso si los objetos se ocultan o cambian de posición [39].

2.3.6 Filtros Kalman

El filtro de Kalman es un algoritmo que estima una variable a partir de datos medidos, siguiendo dos pasos: predicción del estado del sistema y corrección con observaciones. Su objetivo es obtener un estimador óptimo basado en la dinámica del sistema y las observaciones, procesando todas las medidas disponibles para estimar el valor de las variables de interés con precisión [40].

En otras palabras, lo que hace el filtro Kalman es que, a partir de una ecuación de transición del estado, describirá la evolución de una variable a lo largo del tiempo. Junto a esto también existe un observador (ecuación de observaciones) que nos describirá la relación entre las observaciones medidas con la variable de estado [40]. En la Figura se puede ver el algoritmo del filtro Kalman:

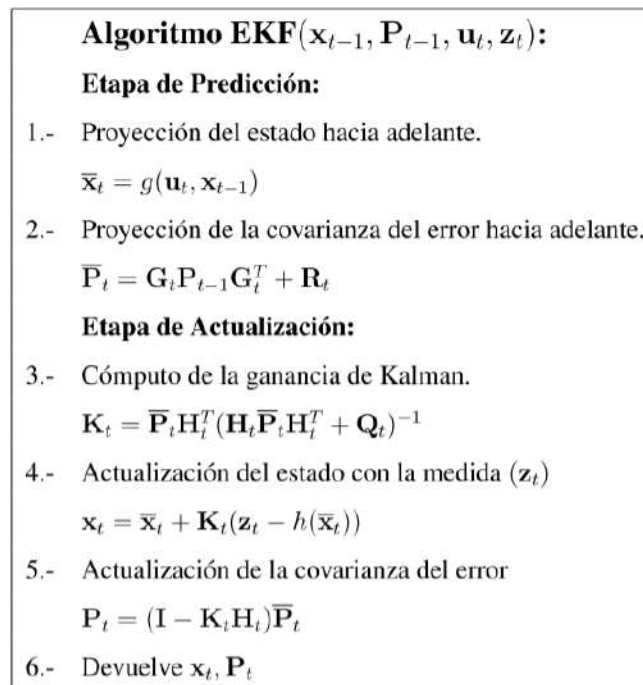


Figura 2-10. Etapas del filtro de Kalman Extendido.

A continuación, se definen las matrices y demás variables que aparecen en la Figura 2-10: $\mathbf{G}_t$ es el Jacobiano del modelo de predicción, de dimensiones $n \times n$ y que corresponde a las Matrices $\mathbf{A}_t$ y $\mathbf{B}_t$ del KF (Ecuación 2.4).

$$\mathbf{G}_t = \frac{\partial g(\mathbf{u}_t, \mathbf{x}_{t-1})}{\partial \mathbf{x}_{t-1}} \quad (2.4)$$

$\mathbf{u}_t$ son las señales de control.

$\mathbf{z}_t$ son las observaciones del entorno que proporcionan los sensores.

$\mathbf{R}_t$ es la matriz $n \times n$ que representa a la covarianza del ruido del proceso (Ecuación 2.5).

$$\mathbf{R}_t = \mathbf{M}_t \mathbf{V}_t \mathbf{M}_t^T \quad (2.5)$$

$\mathbf{M}_t$ es el Jacobiano de dimensiones $n \times l$ (Ecuación 2.6).

$$M_t = \frac{\partial g(u_t, x_{t-1})}{\partial u_t} \quad (2.6)$$

V_t es la matriz de covarianzas de ruido del espacio de control, con dimensiones $l \times l$.
 H_t es el jacobiano del modelo de medida, de dimensiones $m \times n$ y que corresponde a la matriz C_t del KF (Ecuación 2.7).

$$H_t = \frac{\partial h(\bar{x}_t)}{\partial x_t} \quad (2.7)$$

Q_t es la matriz $n \times n$ que representa a la covarianza del ruido de las observaciones.
 K_t es la matriz $n \times m$ que representa a la ganancia de Kalman. La ganancia de Kalman indica la confianza en las características observadas, empleando la incertidumbre de estas junto con una medida de la calidad de los datos proporcionados por los sensores [41].

Este tipo de filtro se usará para realizar la predicción de la trayectoria de los vehículos cercanos respecto al vehículo propio ya que su función será describir la evolución de una variable a lo largo del tiempo, en este caso la posición de los vehículos.

2.3.7 Estimación de colisión con vehículos

En la estimación de riesgo de colisión con otros vehículos, se utilizan diversas fórmulas que toman en cuenta múltiples variables relacionadas con el frenado. Documentos como el referido en [42] detallan con alta precisión las variables que afectan el tiempo de frenado, incluyendo el factor de resistencia del aire, la eficiencia de los frenos, el coeficiente de fricción, el factor de inclinación, la pendiente de la carretera, entre otras. La inclusión de tantas variables puede incrementar la complejidad en la estimación de colisiones, pero también potencialmente mejora la precisión de estas estimaciones. Sin embargo, en entornos cambiantes, es más práctico usar datos sencillos como la distancia y velocidad de los vehículos.

Este enfoque subraya un principio crucial en la prevención de colisiones, que es el “time to collision” o TTC por sus siglas en inglés, este es criterio que ha sido ampliamente aceptado como el criterio básico para diseñar varios tipos de sistemas de alerta de colisión. El TTC básico se define como el tiempo que tardarían los coches en colisionar a su velocidad actual [43]. Está definido como se muestra en la Ecuación 2.8:

$$TTC_t = \frac{L_t}{v_t^E} \quad (2.8)$$

Donde v_t^E es la velocidad del vehículo propio y L_t es la distancia entre el vehículo propio y el delantero. Otra medida importante a tener en cuenta es la **Velocidad relativa** (Δv), pues esta se utiliza como característica para describir la decisión del conductor de frenar. Puede calcularse mediante $\Delta v = v_p - v_E$, donde v_p es la velocidad del vehículo delantero y v_E es la velocidad del vehículo propio.

Ahora bien, con estos términos conocidos se puede hablar de los algoritmos de advertencia basados en la percepción y los basados en la cinemática. Los primeros activan advertencias basadas en el tiempo necesario para que dos vehículos choquen si continúan viajando a su

velocidad y trayectoria actuales. Cuando el tiempo de colisión (TTC) cae por debajo del umbral de percepción humana (cómo se define en el algoritmo), se activa una advertencia para alertar al conductor [44] como se muestra en la figura 2-11. Los segundos, utilizan datos de velocidad de desaceleración y tiempo de reacción para determinar la velocidad y distancia teórica mínima para detenerse con seguridad. Sin embargo, a diferencia de los algoritmos perceptivos que dependen del conocimiento empírico de la percepción humana, los algoritmos cinemáticos determinan la distancia a la que debe ocurrir el inicio del frenado si se quiere evitar una colisión como se observa en la figura 2-12 [44].

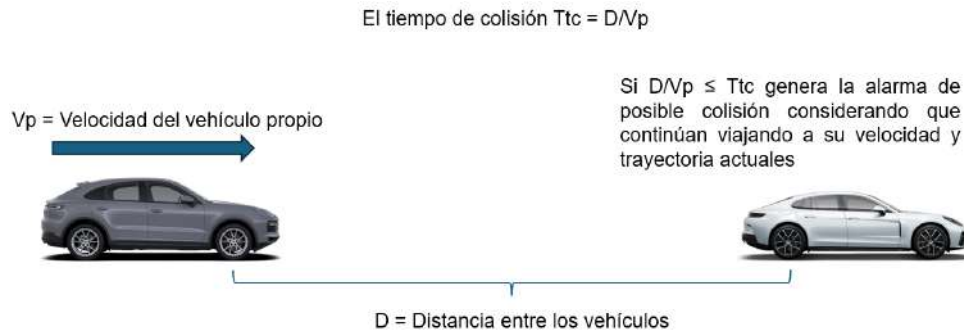


Figura 2-11. Algoritmos basados en la percepción.

Los algoritmos cinemáticos buscan calcular la distancia segura de frenado a partir de las velocidades de los vehículos, las distancias entre ellos y los tiempos de reacción.

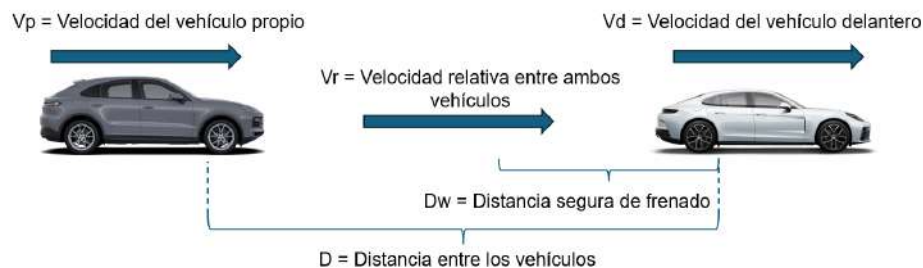


Figura 2-12. Algoritmos cinemáticos.

2.3.8 Metodología RUP

La metodología RUP (Rational Unified Process) [45] es una metodología de desarrollo de software orientado a objeto que establece las bases, plantillas, y ejemplos para todos los aspectos y fases de desarrollo del software. RUP es herramientas de la ingeniería de software que combinan los aspectos del proceso de desarrollo (como fases definidas, técnicas, y prácticas) con otros componentes de desarrollo (como documentos, modelos, manuales, código fuente, etc.) dentro de un framework unificado. RUP establece cuatro fases de desarrollo cada una de las cuales está organizada en varias iteraciones separadas que deben satisfacer criterios definidos antes de emprender la próxima fase [45]. Las fases son:

- **Incepción:** en el que se lleva a cabo la evaluación inicial para delimitar el sistema y determinar si vale la pena continuar con el proyecto.
- **Elaboración:** en la que se realiza una evaluación más detallada, se crea un plan de desarrollo y se mitigan los riesgos clave.

- **Construcción:** Se lleva a cabo la construcción del producto por medio de una serie de iteraciones.
- **Transición:** Se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios [45].

2.4 OBSERVACIONES FINALES

En este capítulo se consolidan los aportes más relevantes derivados de la investigación y de la implementación de sistemas de alerta de colisión entre vehículos. Cada trabajo analizado aborda distintas técnicas y métodos orientados al desarrollo de dichos sistemas. La revisión bibliográfica permitió, además, identificar tendencias, dificultades y oportunidades en este campo de estudio.

De manera particular, se examinan diversas metodologías empleadas para la detección y prevención de colisiones, abarcando tanto enfoques tradicionales como innovaciones recientes basadas en inteligencia artificial y aprendizaje automático. Asimismo, se analizan los principales desafíos técnicos que enfrentan estos sistemas, entre ellos la precisión en la detección de objetos, la integración con otras tecnologías de asistencia a la conducción y la adaptabilidad a condiciones variables de tráfico y clima.

Este capítulo establece, por lo tanto, una base sólida de los conceptos teóricos necesarios para comprender el desarrollo del presente trabajo. Al proporcionar un marco detallado y bien fundamentado, se asegura que los aspectos del proyecto estén claramente entendidos y alineados con las mejores prácticas y conocimientos disponibles. A partir de estos antecedentes, puede afirmarse que el diseño de un sistema de alerta de colisión basado en cámaras estéreo, redes neuronales convolucionales, algoritmos de seguimiento y filtros de Kalman para la predicción, cuenta con un respaldo teórico robusto que justifica la pertinencia de emplear dichos enfoques.

En este contexto, la elección de las cámaras estéreo **ZED2** como sensor principal del entorno resulta especialmente adecuada, ya que ofrecen un sistema de medición de distancias preciso y rápido. El uso de señales de video permite la generación de panorámicas, permitiendo tener un amplio rango de visión para advertir al conductor, así mismo facilita la integración de redes neuronales convolucionales para la detección y seguimiento de vehículos, que como se pudo observar en los antecedentes tienen un buen desempeño y rendimiento. Al complementar esto con filtros de Kalman para la predicción de trayectorias, se hace posible el desarrollo de un sistema robusto, capaz de anticipar de manera temprana posibles colisiones durante la conducción. Por otro lado, teniendo en cuenta las dinámicas que hay en carretera se piensa usar los algoritmos basados en la percepción pues no dependen de muchas variables para generar alertas.

No obstante, para garantizar un funcionamiento en tiempo real y con la fiabilidad requerida, resulta indispensable contar con una plataforma de cómputo con el rendimiento suficiente para procesar simultáneamente la estimación de profundidad, la inferencia de redes neuronales y los algoritmos de seguimiento. En este sentido, dispositivos como la **NVIDIA AGX Orin** representan una solución idónea, al ofrecer la capacidad de cómputo necesaria para integrar de manera eficiente todos estos componentes y asegurar la operación del sistema bajo condiciones reales de conducción.

3. DESARROLLO DE UN APLICATIVO EMBEBIDO DE ALERTA DE COLISIÓN CON OTROS VEHÍCULOS

3.1 INTRODUCCIÓN

La implementación de sistemas de alerta de colisión se ha vuelto una prioridad clave en el desarrollo de tecnologías orientadas a la seguridad vial. Ante el aumento del tráfico y la complejidad de las redes de transporte, los sistemas de asistencia al conductor son fundamentales para prevenir accidentes y mitigar su gravedad. Este trabajo se enfoca en el desarrollo de un sistema inteligente capaz de alertar sobre posibles colisiones con otros vehículos, mejorando así la seguridad de los ocupantes.

El capítulo describe el desarrollo de un aplicativo embebido diseñado específicamente para esta tarea. La construcción del sistema se basó en la metodología RUP, mediante la cual se definieron los requerimientos funcionales y no funcionales. En primer lugar, se presentan estos requerimientos junto con un diagrama conceptual que muestra la relación entre las funciones del sistema. Posteriormente, se incluyen el diagrama de clases, que define el rol específico de cada clase, y el diagrama de nodos, que ilustra el flujo de información entre componentes.

El desarrollo del aplicativo se divide en dos módulos principales. El **módulo de configuración** abarca el diseño de la interfaz gráfica de usuario (GUI), que permite ajustar parámetros clave del sistema. Además, gestiona la extracción y modificación de parámetros de las cámaras estéreo, permitiendo su adaptación a distintas condiciones operativas.

Por su parte, el **módulo de procesamiento** se enfoca en la detección y seguimiento de vehículos, la evaluación del riesgo de colisión y la generación de alertas. Este módulo integra algoritmos y herramientas que permiten identificar vehículos en el entorno, rastrear su movimiento, calcular el riesgo potencial de impacto y emitir alertas oportunas para prevenir accidentes.

3.2 DISEÑO DEL APLICATIVO EMBEBIDO DE ALERTA DE COLISIÓN CON OTROS VEHÍCULOS

Para desarrollar el aplicativo se utilizó la metodología RUP, caracterizada por su enfoque iterativo e incremental que busca productos de alta calidad. Con RUP se definieron los requisitos funcionales y no funcionales, lo que permitió elaborar diagramas de concepto y clases, facilitando una planificación y entrega organizada del software. Otros entregables como casos de uso y diagramas de secuencia se incluyen en los anexos.

3.2.1 Requerimiento funcionales y no funcionales.

Los requerimientos funcionales definen las características que el software debe cumplir según los objetivos del usuario. En el caso de una alerta de colisión, un requerimiento funcional sería la detección de vehículos mediante un algoritmo que analice las imágenes de las cámaras estereoscópicas.

Con esto en mente, entonces se diseñan los siguientes requerimientos funcionales, y se categorizan en dos vertientes: Esenciales (E) y Opcionales (O). Esto se observa en la Tabla 3-1.

Tabla 3-1. Requerimientos funcionales.

Ref #	Funciones	Categoría
1.0	Una GUI que permita configurar los parámetros de riesgo de colisión, selección de la red neuronal, configuración de los parámetros del algoritmo y el control de la ejecución del aplicativo donde se pueda visualizar el mosaico RGB, el mapa de disparidad, la detección, el seguimiento y los datos de calibración de las cámaras estereoscópicas, así como sus matrices de homografía.	E
2.0	Detectar los vehículos por medio de un algoritmo de detección usando las imágenes captadas por las cámaras estereoscópicas.	E
3.0	Realizar el tracking de los vehículos cercanos detectados usando extracción de características visuales.	E
4.0	Estimar la posición relativa de los vehículos a partir de la distancia entre cada cámara estereoscópica y el vehículo cercano.	E
5.0	Calcular la velocidad del vehículo propio.	E
6.0	Determinar la probabilidad de una colisión mediante los datos de distancia, velocidad y los parámetros de riesgo definidos.	E
7.0	Emitir una alerta visual o auditiva si el vehículo está dentro del rango de colisión.	E

Requerimientos no funcionales: Son aquellos que describen los atributos del sistema que no están directamente relacionados con las funciones y capacidades del software, sino con su rendimiento, calidad, seguridad, usabilidad y otros aspectos que no se refieren a las funcionalidades específicas del software. Estos se muestran en la Tabla 3-2.

Tabla 3-2. Requerimientos no funcionales.

Ref #	Descripción	Categoría
1.0	Software	
1.1	Se desarrollará en ROS 2 Humble.	E
1.2	Se utilizará la versión de Linux Ubuntu 22.04 para el sistema embebido.	E
1.3	Se usará el programa QTdesigner para diseñar la interfaz grafica	O
1.4	Se usará ZED ROS 2 wrapper para trabajar con las cámaras estéreo	O
1.5	Se usará una versión de Python mayor o igual a la 3.10	E
2.0	Hardware	
2.1	Se utilizarán 3 cámaras estero ZED.	E

2.2	Se usará el sistema embebido Nvidia Jetson Orin AGX.	O
-----	--	---

3.2.2 Diagrama de concepto

En la sección 1.3 se habló de la solución propuesta y en la Figura 3-1, se muestra el diagrama de concepto, en este se muestra de forma general las funciones del sistema y el actor principal de las mismas que es el aplicativo embebido.

El aplicativo tiene dos módulos: configuración y procesamiento. En el módulo de configuración, el usuario establece los parámetros de configuración, puede seleccionar el modelo de red neuronal que desee usar, así como los niveles de alerta y visualizar diferentes parámetros del aplicativo como parámetros estere, calibración de las cámaras entre otros.

En el módulo de procesamiento, el programa se inicia seleccionando cámaras o archivos “bagfiles” como fuente de datos y se ejecuta con un botón. Se muestran la imagen panorámica RGB y el mapa de disparidad, junto con los vehículos detectados, sus centroides, distancias y las respectivas alertas visuales según sea el caso.

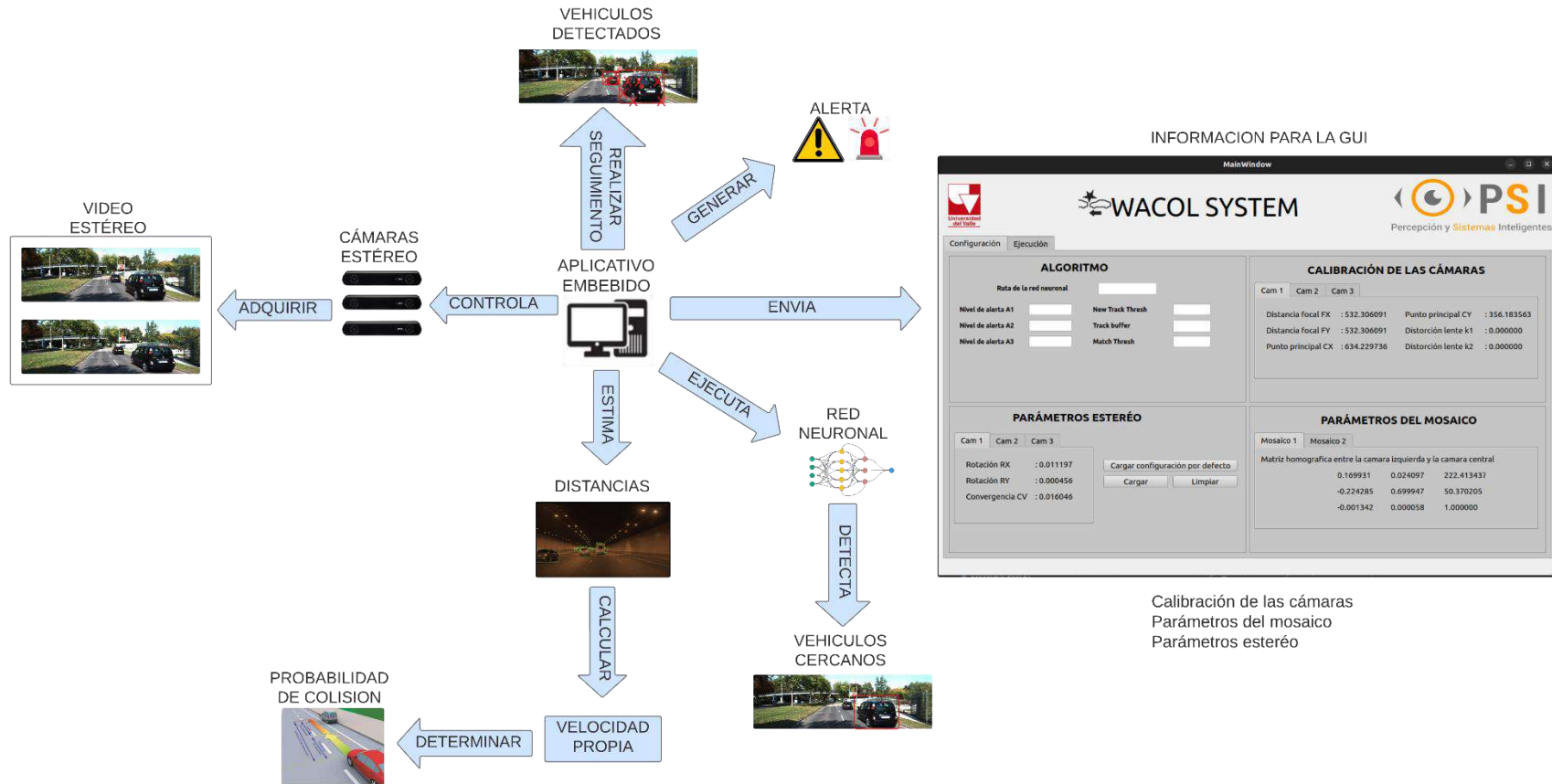


Figura 3-1. Diagrama de conceptual.

3.2.3 Diagrama de clases

El diagrama de clases presentado en la Figura 3-2 describe la estructura principal del sistema desarrollado, destacando las clases involucradas, sus atributos, métodos y las relaciones entre ellas. Este diagrama permite comprender la organización del código y la interacción entre los distintos componentes del proyecto, facilitando su mantenimiento, escalabilidad y reutilización.

Las clases “PanoramicRGBNode”, “DepthVizPanoramaNode” y “PanoramicaDepthNode” constituyen el núcleo del sistema, ya que procesan las imágenes y matrices de profundidad generadas por la clase “ZED_PUBLISHER”, generando las panorámicas RGB, de visualización de profundidad y de distancias, respectivamente. Como se observa en el diagrama, todas las demás clases dependen de estas para obtener información. La única clase que se relaciona directamente con “ZED_PUBLISHER” es “VelocityPublisher”, encargada de publicar la velocidad. Esta relación directa se debe a que necesita los datos de aceleración de la cámara frontal, a partir de los cuales calcula la velocidad y, con ella, el tiempo estimado de colisión.

Las clases “CameraSubscriber” y “YoloInferenceNode” se encuentran en el mismo archivo y trabajan en conjunto para llevar a cabo las inferencias. “CameraSubscriber” gestiona la recepción de las imágenes provenientes de “PanoramicRGBNode” y las entrega a “YoloInferenceNode”, donde la red neuronal YOLO realiza las detecciones.

Por otro lado, “YoloSubscriber” incorpora internamente la clase “KalmanTracker”, que define las matrices necesarias y contiene los métodos para actualizar y predecir la trayectoria de los objetos detectados. El “KalmanTracker”, se usa para predecir posibles trayectorias futuras, y la clase “YoloSubscriber” lo emplea para graficar dicha trayectoria y visualizarla en la panorámica RGB, permitiendo al conductor anticipar el movimiento de los objetos.

La clase “DistanceNode” recibe los datos de la panorámica de profundidad generados por “PanoramicaDepthNode” y las inferencias de “YoloInferenceNode”, que le indican en qué puntos de la imagen debe consultar y retornar el valor de distancia.

Por último, se encuentra la clase “GuiNode”, que integra toda la información procesada por el sistema y la presenta de forma clara al conductor. No obstante, la visualización principal de la interfaz es generada por la clase “YoloSubscriber”, ya que esta se encarga de graficar sobre la panorámica RGB las inferencias de la red neuronal, los identificadores (ID) de cada objeto detectado, sus respectivas distancias, trayectorias estimadas y las alertas generadas.

A lo largo de este capítulo se explicará con mayor detalle el funcionamiento de cada sección del sistema y su diseño.

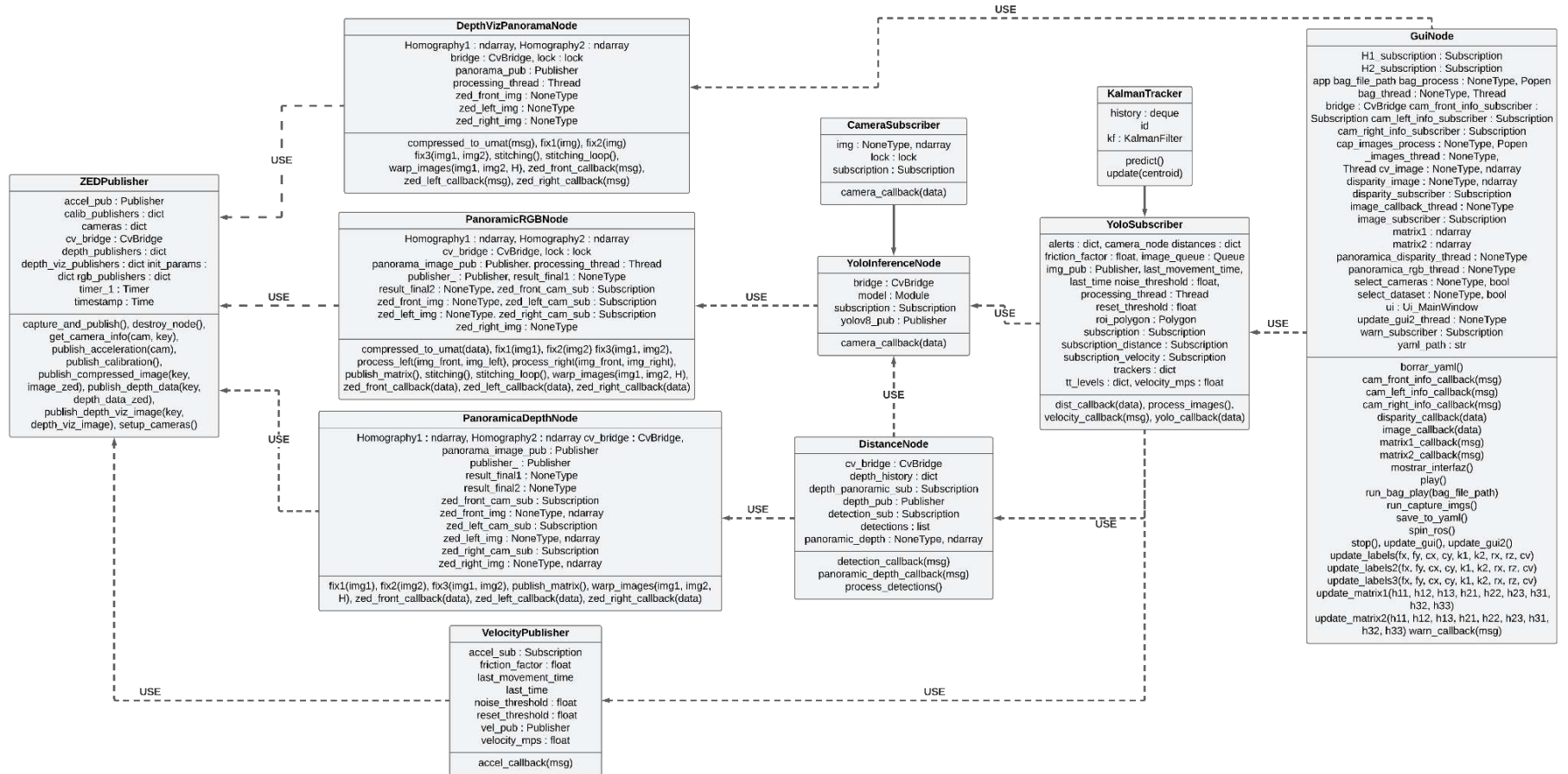


Figura 3-2. Diagrama de Clases.

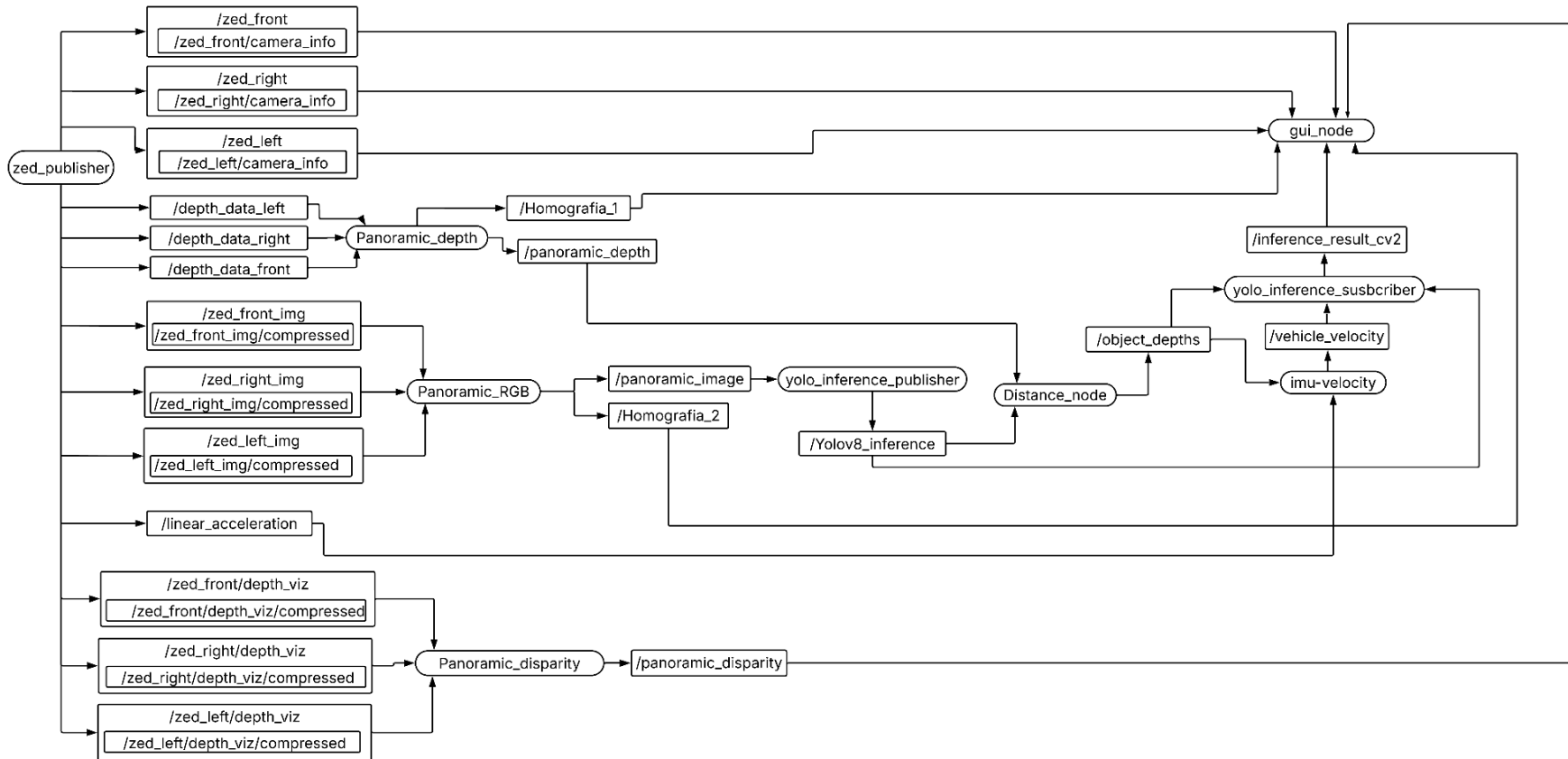


Figura 3-3. Diagrama de nodos.

3.2.4 Diagrama de nodos

En esta sección se describen los nodos desarrollados en ROS2, cada uno con una función específica. Estos nodos se comunican mediante tópicos, que actúan como canales para intercambiar información por publicación y suscripción. La figura 3-3 muestra un diagrama con los nodos (óvalos) y sus conexiones a través de los tópicos (rectángulos).

A continuación, se va a dar una breve descripción de lo que realiza cada nodo. Empezando por el principal que extrae la información del exterior y la comparte con los demás nodos:

- **zed_publisher:** Este nodo se encarga de capturar y publicar la información del entorno. Transmite imágenes RGB comprimidas (figura 3-4), así mismo publica las matrices de profundidad (figura 3-5), los parámetros de calibración extrínseca e intrínseca de cada cámara y las imágenes en escala de grises para la visualización de la profundidad (figura 3-6). Organizando estos datos en tópicos específicos, identificando a qué cámara pertenece cada uno.



Figura 3-4. Imágenes RGB de las cámaras izquierda, central y derecha.



Figura 3-5. Matrices de profundidad de las cámaras izquierda, central y derecha.



Figura 3-6. Imágenes en escala de grises para la visualización de la profundidad.

- **Panoramic_depth:** El nodo recibe matrices de profundidad para generar una panorámica con información de distancia de toda la escena, publicando la matriz llamada “**panoramic_depth**” (figura 3-7). Además, publica la matriz “**Homografía_1**” (más adelante se explica que es la homografía), que el nodo de la GUI usa para mostrar la configuración. Como hay 3 cámaras, se requieren dos

homografías: una que relaciona la cámara izquierda con la central y otra que conecta la cámara derecha con la central



Figura 3-7. Panorámica de las matrices de profundidad de cada cámara.

- **Panoramic_RGB:** Este nodo, al igual que *Panoramic_depth*, genera una imagen panorámica, pero a partir de las imágenes RGB comprimidas de las tres cámaras. Como resultado, publica una imagen de mayor dimensión que las originales, permitiendo capturar en una sola vista la perspectiva de cada cámara (figura 3-8). Esta imagen se transmite a través del tópico *panoramic_image*. Además, el nodo también publica *Homografía_2*, que representa la transformación de perspectiva entre la cámara izquierda y la cámara central.



Figura 3-8. Panorámica RGB.

- **Panoramic_disparity:** Este nodo combina imágenes de disparidad de varias cámaras en una panorámica y aplica un esquema de falso color (rojo para objetos cercanos y azul para los lejanos) para mejorar la visualización (figura 3-9). La imagen resultante se publica en la interfaz gráfica mediante el tópico ROS2 "panoramic_disparity".

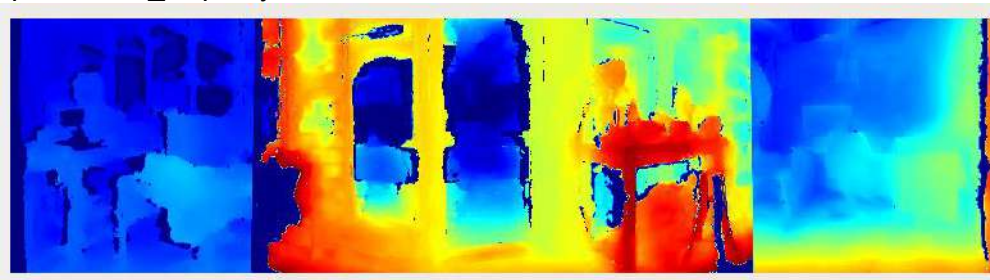


Figura 3-9. Panorámica en escala de grises normalizada con falso color.

- **yolo_inference_publisher:** Este nodo ejecuta la red neuronal y realiza el seguimiento de objetos, identificándolos en la panorámica RGB, asignándoles una ID única y determinando su centroide. Los datos, incluidos el bounding box, el centroide y la ID, se publican como un mensaje personalizado en el tópico “/Yolov8_inference”. (figura 3-10).

```
header:
  stamp:
    sec: 1742588038
    nanosec: 641120980
  frame_id: inference
yolov8_inference:
- class_name: ''
  id: 1
  top: 116
  left: 212
  bottom: 228
  right: 313
  centroid_x: 262
  centroid_y: 172
- class_name: ''
  id: 2
  top: 130
  left: 594
  bottom: 205
  right: 662
  centroid_x: 628
  centroid_y: 167
```

Figura 3-10. Mensaje custom para los objetos detectados.

- **Distance_node:** Este nodo calcula la distancia de los objetos detectados usando las coordenadas del centroide obtenidas del tópico “/Yolov8_inference” y consultando la distancia para esas coordenadas en la panorámica de profundidades del tópico “/panoramic_depths”. Luego, publica el ID de cada objeto junto con su distancia en el tópico “/object_depths” (figura 3-11).

```
---
header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
distance_measure:
- id: 1
  distance: 1.40347314453125
- id: 4
  distance: 1.4183009033203127
---
```

Figura 3-11. Mensaje custom para publicar las distancias de los objetos detectados.

- **imu_velocity:** Este nodo procesa los datos de aceleración en el eje Z suministrados por la IMU de la cámara ZED frontal, y los convierte en estimaciones de velocidad para el vehículo propio.
- **yolo_inference_susbcriber:** Este nodo recibe como entrada los datos de inferencia generados por el modelo YOLO, la información de distancias a los objetos detectados

y la velocidad, para generar las respectivas alertas de colisión. Su función principal es superponer esta información sobre la imagen panorámica, permitiendo visualizar las detecciones, las distancias asociadas y el nivel de alerta generado para cada caso (figura 3-12). Adicionalmente, cuando se identifican objetos en movimiento, el nodo muestra la predicción de su trayectoria mediante una línea de color morado, como se ilustra en la Figura 3-13.



Figura 3-12. Imagen con la colisión evaluada y las distancias de seguridad visibles.



Figura 3-13. Imagen con la proyección de la trayectoria.

- **gui_node:** Este nodo tiene la función de proporcionar una visualización completa de toda la información procesada por los demás nodos, permitiendo al usuario monitorear los datos obtenidos del entorno. Además, sirve como una interfaz de configuración, permitiendo ajustar distintos parámetros que afectan el funcionamiento de los nodos.

3.3 IMPLEMENTACIÓN APLICATIVO EMBEBIDO DE ALERTA DE COLISIÓN CON OTROS VEHÍCULO

En esta sección se describen los módulos del aplicativo para facilitar la comprensión y futura modificación del software. El sistema cuenta con dos módulos: configuración y procesamiento. El módulo de configuración, que es el núcleo del aplicativo, define los valores que utilizará el módulo de procesamiento. Al aplicativo se le asignó el nombre **WACOL SYSTEM**. Este proviene de la combinación en inglés de las palabras *warn* (alerta) y *collision* (colisión). Para su construcción se tomaron las dos primeras letras de **WARN** y tres primeras letras de **COLISSION**, dando origen a la abreviatura **WACOL**, que identifica al sistema de alerta de colisión.

3.3.1 Módulo de configuración

Este módulo gestiona los valores de configuración antes de ejecutar el procesamiento. Permite al usuario ingresar los niveles de alerta, red neuronal y parámetros para el seguimiento de objetos. Así como visualizar ciertos parámetros como la calibración de las cámaras, las homografías de la panorámica y los parámetros estéreo.

3.3.1.1 GUI

En el siguiente apartado se presentará la GUI diseñada para el módulo de configuración indicando cada una de sus partes y las funciones que cumplen en la Figura 3-14.

En el componente 1, de la Figura 3-14, la cual corresponde a la zona de configuración del algoritmo, es donde se ingresan los valores que configuran los nodos de “Colission” y “yolo_subcriber”. En este se ingresa la “ruta” o el “path” para cargar la red neuronal, así como los umbrales según los cuales va a funcionar el tracker, que en este caso es ByteTrack (más adelante se explicará su funcionamiento). Los 3 parámetros ahí expuestos hacen referencia a lo siguiente:

- **New track Thresh:** es el umbral para empezar un nuevo seguimiento.
- **Track buffer:** es la cantidad de frames en los cuales mantiene un ID para un objeto aun cuando ya salió de escena dicho objeto.
- **Match Thresh:** es el umbral para hacer match entre los objetos identificados a los cuales ya se les inicio el seguimiento.

Por otro lado, los niveles de alerta A1, A2 y A3, reciben los valores numéricos para realizar la advertencia de acuerdo con el nodo de “Colission” (este se explicará más adelante).

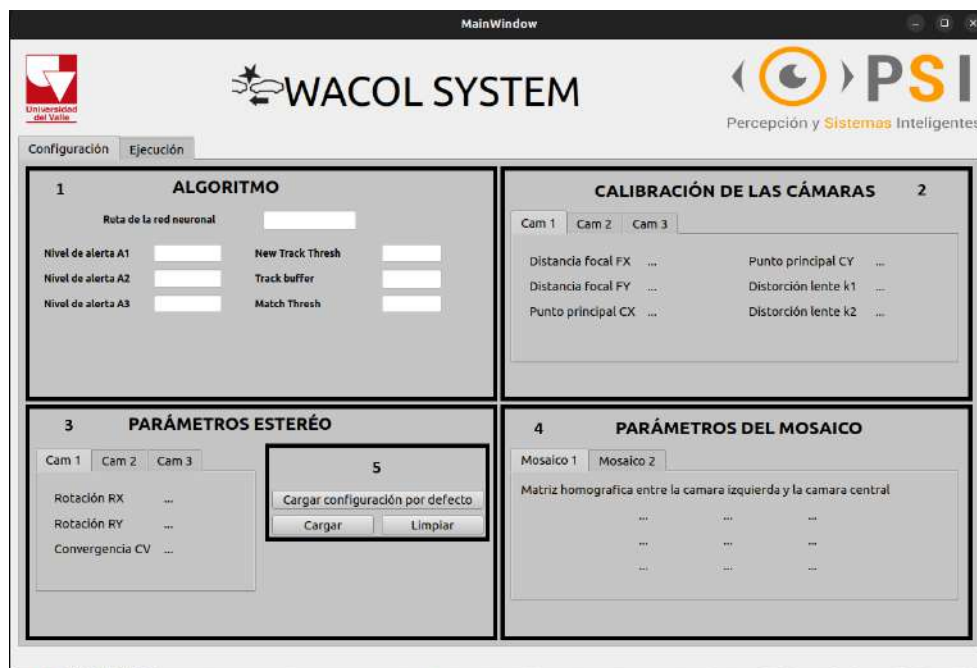


Figura 3-14. GUI módulo de configuración.

En el componente 2 se visualizan los valores de calibración para cada una de las cámaras y son los siguientes:

- **Distancia focal (F):** es la distancia del plano de la imagen al centro de la lente [46].
- **Punto principal (C):** interseca el eje óptico de la cámara y el plano de la imagen [46].
- **Distorsión del lente(K):** representa el desplazamiento radial en función de la calidad del lente de la cámara utilizada y la distancia entre el punto en el espacio y el centro de la imagen [46].

En el componente 3 se visualizan también los valores que corresponden a los parámetros estéreo de las cámaras, estos parámetros son los siguientes:

- **Convergencia (CV):** mide la convergencia óptica [47].
- **Rotación (RX y RZ):** son el otro eje de rotación que describe la transformación entre ambos sensores [47].

En el componente 4 se muestran las matrices de homografía utilizadas para generar la panorámica con las 3 cámaras. Estas matrices permiten relacionar puntos comunes entre las imágenes tomadas desde distintas perspectivas (figura 3-15), lo cual es esencial para unir las correctamente en una panorámica [48].

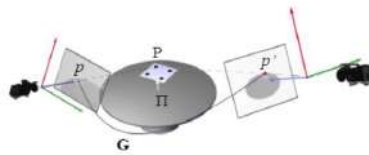


Figura 3-15. Homografía 2D.

Por último, está el componente 5 que son los botones para cargar las configuraciones por defecto o las generadas por el usuario y así mismo limpiar los campos llenados. En la figura 3-16 se muestra como se ve la GUI cuando inicia, mostrando los datos de calibración de cada cámara, los parámetros estéreo y los parámetros del mosaico.

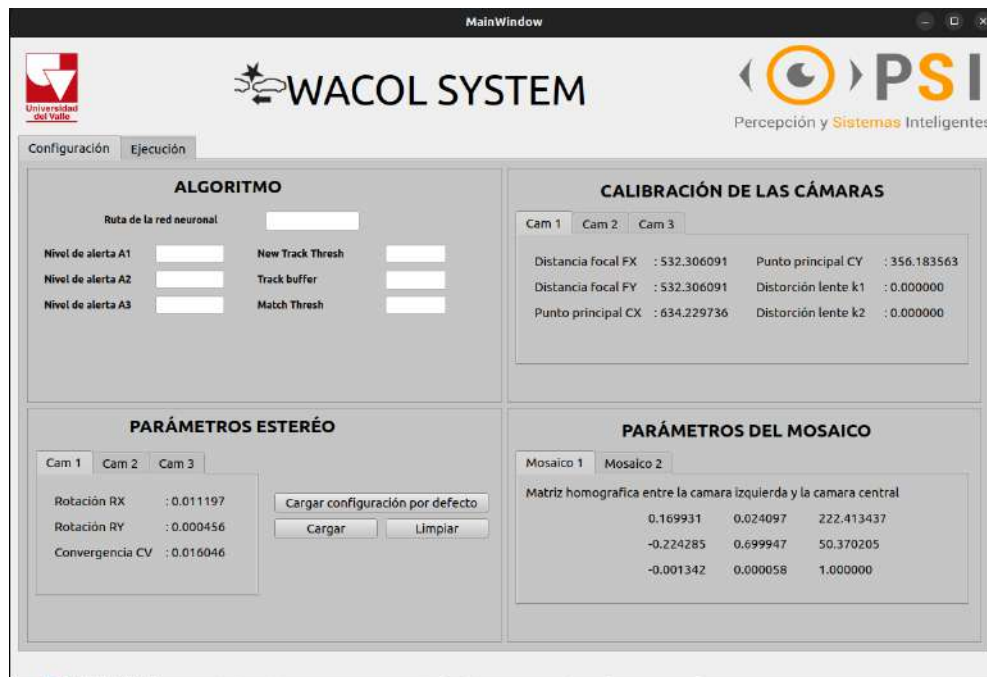


Figura 3-16. GUI con la información cargada.

3.3.1.2 Extracción de parámetros de las cámaras

Para la extracción de los parámetros intrínsecos se puede usar el SDK de ZED con el cual se puede abrir cada cámara y al mismo tiempo acceder a sus parámetros y modificarlos usando la aplicación ZED_Explorer. Por otro lado, también se pueden acceder y modificar mediante líneas de código en Python usando la librería "pyzed" con la cual se puede acceder a los parámetros de cada cámara usando "calibration_params = zed.get_camera_information().camera_configuration.calibration_parameters". En los comentarios de la página de Stereo Labs se comentaba que en caso de tener las cámaras ZED 2 no es necesario calibrarlas, pues estas vienen de fábrica con una calibración muy rigurosa, por lo cual volverlas a calibrar podría rebajar su calidad y desempeño [49].

3.3.2 Módulo de procesamiento

Este módulo procesa las imágenes y permite visualizar la panorámica RGB, el mapa de disparidad y las detecciones de objetos con sus distancias y centroides. Además, realiza el seguimiento de vehículos, calcula las distancias, la velocidad del vehículo propio y utiliza un filtro de Kalman para predecir trayectorias y evaluar el riesgo de colisión, generando alertas visuales según el nivel de peligro.

3.3.2.1 GUI

En el siguiente apartado se presentará la GUI diseñada para el módulo de procesamiento o ejecución indicando cada una de sus partes y las funciones que cumplen en la Figura 3-17.

En el componente 1 de la Figura 3-17, correspondiente a la vista "Panoramic RGB", donde se visualiza el resultado del procesamiento de las imágenes RGB captadas por las tres cámaras. Este componente muestra la panorámica generada, así como la detección de los

vehículos presentes en la escena, incluyendo su identificador (ID), la distancia estimada, el nivel de riesgo de colisión y su trayectoria futura.

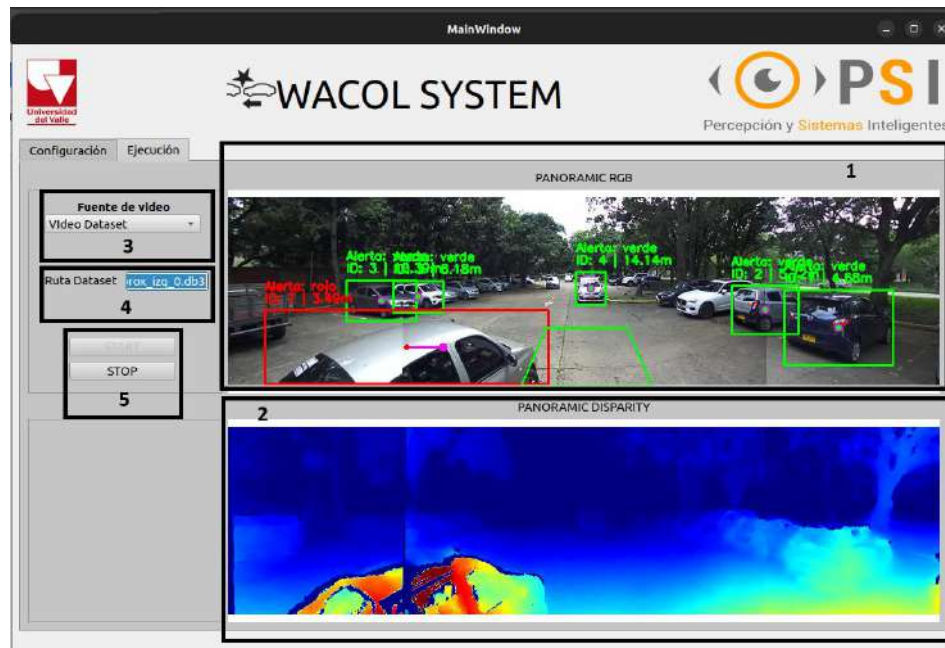


Figura 3-17. GUI módulo de procesamiento o ejecución.

El componente 2 corresponde a la vista denominada “Panoramic Disparity”. En esta se pueden visualizar con claridad los objetos cercanos en la escena, ya que se utiliza una escala de colores para representar la distancia. Los objetos más próximos al vehículo aparecen en tonos cálidos, mientras que los más lejanos se representan con colores fríos.

El componente 3, denominado “Fuente de video”, permite seleccionar la fuente de datos que el sistema utilizará para procesar la información. Al hacer clic sobre este componente, se despliega un menú con dos opciones: las cámaras estéreo y el video dataset (que es un conjunto de datos tipo rosbag). El componente 4 funciona en conjunto con el componente 3, ya que permite ingresar la ruta (*path*) del conjunto de datos cuando, en el componente 3, se selecciona la opción “video dataset”. Por último, el componente 5 tiene la función de controlar la ejecución del aplicativo, permitiendo al usuario iniciar o detener el sistema según sea necesario.

3.3.2.2 Generación de las panorámicas

Como se mencionó en la sección 3.2.4 Diagrama de nodos hay 3 panorámicas que son el input del sistema, por lo que es pertinente mencionar el proceso que se realizó para poder generarlas.

Las panorámicas son el resultado de un proceso llamado stitching, que en su significado literal del español hace referencia a las puntadas que se hacen para coser la ropa, sin embargo, en el ámbito de la fotografía hace referencia a la unión de imágenes o vídeos para generar panorámicas.

El proceso de stitching busca unir diferentes vistas de un paisaje para formar una sola imagen, teniendo en cuenta que entre las imágenes comparten regiones en común. El stitching implica lo siguiente:

1. Detectar puntos en común entre imágenes (features como SIFT, ORB, SURF, etc.).
2. Calcular las transformaciones geométricas necesarias (homografías, traslaciones, rotaciones).
3. Superponer y fusionar las imágenes de forma continua, buscando eliminar bordes visibles, diferencias de color o distorsiones.

Teniendo esto presente a continuación se presentan los pasos desarrollados para generar las panorámicas. Cabe mencionar que el procedimiento de stitching se realizó entre 3 de las 6 cámaras, ya que hacer un stitching de 6 cámaras era costoso computacionalmente, en especial considerando que se necesitaba realizar un stitching de video, por lo que se realizó como se muestra en la figura



Figura 3-18. Disposición de las cámaras.

La cámara 1 corresponde a la ubicada en el lado derecho, la cámara 2 a la frontal y la cámara 3 a la izquierda. La extracción de características comunes entre imágenes se realizó considerando los pares derecha–frontal e izquierda–frontal. Es importante precisar que, en el caso de la cámara frontal, se empleó únicamente la subcámara del lado derecho (señalada con el número 2 en la figura 3-18). En consecuencia, se llevaron a cabo dos procesos de *stitching*: uno entre las imágenes de la cámara derecha y la frontal, y otro entre la cámara izquierda y la frontal. Finalmente, los resultados de ambos procesos se integraron para obtener la imagen panorámica completa. Este procedimiento se detalla paso a paso en las secciones siguientes, lo que permitirá comprender con mayor claridad su desarrollo.

Ahora bien, el extractor de características usado para el proceso fue SIFT (Scale-Invariant Feature Transform) este es un algoritmo de visión por computador que se utiliza para detectar y describir puntos clave (características locales) en imágenes. Fue desarrollado por David Lowe en 1999 y es muy usado porque permite encontrar puntos de interés en una imagen que son **invariantes a escala, rotación y cambios de iluminación** [50].

El proceso de stitching se realiza inicialmente con imágenes estáticas para obtener las homografías entre las cámaras derecha-frontal e izquierda-frontal. Ya que con estas se puede generar la panorámica del video. El proceso para obtener dichas homografías fue el siguiente:

1. **Carga de imágenes:** El programa comienza leyendo dos imágenes desde disco, que corresponden a capturas de distintas cámaras (ejemplo: derecha-frontal figura 3-19).



Figura 3-19. Par de imágenes a unir.

2. **Extracción de características (SIFT)** Se aplica el algoritmo **SIFT (Scale-Invariant Feature Transform)** para detectar puntos clave (*keypoints*) en las imágenes y describirlos mediante descriptores (*features*).

- Esto permite identificar estructuras relevantes de la imagen que son invariantes a escala, rotación e iluminación.

3. **Matching de características:** Se utiliza un **BMatcher (Brute Force Matcher)** para encontrar correspondencias entre los descriptores de una imagen y los de la otra .

- Se aplica el **ratio test de Lowe** para filtrar los matches más confiables y eliminar correspondencias falsas.



Figura 3-20. Matching de las características.

4. **Cálculo de la Homografía:** Con los puntos válidos obtenidos, se calcula una **matriz de homografía** mediante RANSAC.

- Esta matriz representa la transformación proyectiva que permite mapear una imagen sobre la otra (traslación, rotación, escala y perspectiva).

Con las homografías previamente calculadas, el siguiente paso del proceso consiste en aplicar la transformación de warping (deformación o proyección), mediante la cual las imágenes son ajustadas y alineadas dentro de un mismo plano de referencia.

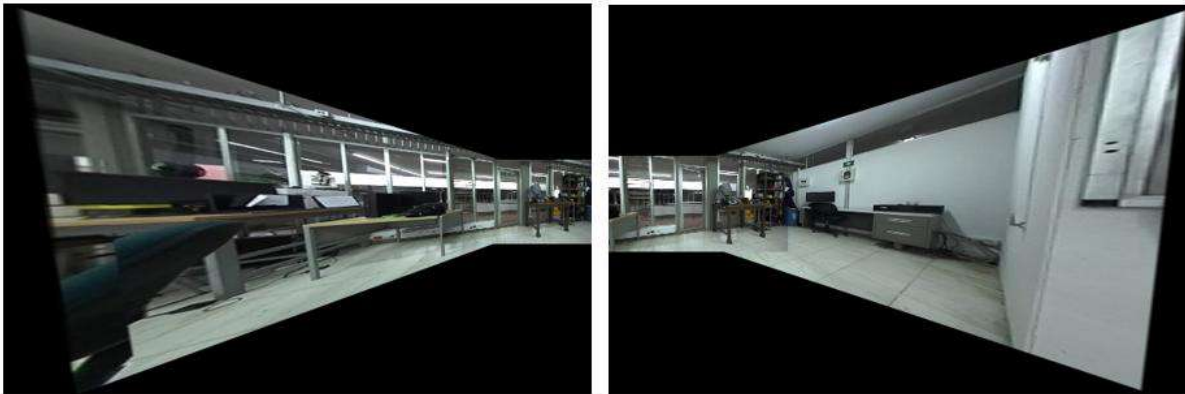


Figura 3-21. Imágenes resultantes después de aplicar el warping.

Después, se aplican transformaciones de perspectiva adicionales para normalizar ciertas regiones de interés, lo que permite representarlas de manera más clara y uniforme (figura 3-22). Luego, los fragmentos resultantes se recortan y concatenan, conformando la panorámica final (figura 3-23).



Figura 3-22. Imágenes resultantes después de aplicar transformaciones de perspectiva.



Figura 3-23. Panorámica resultante.

Este mismo procedimiento se aplica también en la generación de la panorámica de disparidad y en la de profundidad, con el fin de mantener la uniformidad de los datos. De esta manera, los vehículos y objetos visibles en la imagen RGB se corresponden con lo que aparece en la panorámica de disparidad. Del mismo modo, las coordenadas de los vehículos detectados deben coincidir con las de la panorámica de profundidad, garantizando que la distancia medida se asocie efectivamente a las coordenadas del objeto observado en el entorno y no a otros presentes dentro del campo de visión.

3.3.2.3 Detección de vehículos

Para la detección de vehículos se evaluaron varios métodos mencionados en los antecedentes, con especial énfasis en las técnicas basadas en aprendizaje profundo. Entre ellas, el algoritmo YOLO (You Only Look Once) destacó por su alta velocidad de procesamiento y precisión, aspectos clave para esta aplicación. Además, su implementación es sencilla y adaptable a distintos conjuntos de datos, lo que permite ajustarlo fácilmente a las necesidades específicas del proyecto. Estas cualidades hacen de YOLO una opción ideal para el desarrollo del sistema de detección de vehículos.

La tabla 3-3 contiene los resultados en cuanto a desempeño de las diferentes versiones de redes neuronales pertenecientes a YOLO, estas pruebas de desempeño fueron realizadas por la empresa Stereo Labs en dos equipos de hardware diferentes que son la Nvidia AGX ORIN y la Nvidia RTX 4070 TI donde AP es (Average Precision) la precisión promedio y AP 0.5 es la precisión promedio de todas las clases cuando el cuadro delimitador previsto tiene un IoU > 0,5 con verdad fundamental. El IoU (Intersección sobre Unión) mide la superposición entre dos cuadros delimitadores. Un valor de 1 indica una coincidencia perfecta, mientras que 0 indica que no hay superposición.

Tabla 3-3. Comparación entre los diferentes modelos YOLO más recientes del 2023 [51].

MODELO	AP	AP0.5	AGX ORIN (FPS)	RTX4070TI (FPS)
v5n	28	45.7	370	934
v8n	37.3	52.5	383	1163
v7-tiny	37.4	55.2	290	917
v5s	37.4	56.8	277	877
v8s	44.9	61.8	260	925
v5m	45.4	64.1	160	586
v8m	50.2	67.2	137	540
v5l	49	67.3	116	446
v7	51.2	69.7	115	452
v8l	52.9	69.8	95	391
v5x	50.7	68.9	67	252
v7x	52.9	71.1	77	294
v8x	53.9	71.0	64	236

Analizando los datos presentados en la tabla 3-3 se puede decir que los modelos YOLOv8m y YOLOv5l son los modelos más balanceados en cuanto a su precisión promedio y su tiempo de procesamiento. El cual, se puede ver reflejado en la cantidad de FPS, pues entre más rápido sea el procesamiento más FPS puede mostrar, se descartan las versiones “x”, ya que estas hacen referencia a los modelos más grandes, es decir, con más capas y los modelos “s” y “n” ya que su procesamiento es más veloz, sin embargo, la precisión es mucho menor y sus capas también son pocas.

3.3.2.3.1 Selección del Dataset

Para el dataset, se seleccionaron dos opciones de la plataforma Roboflow, una empresa que facilita a los desarrolladores la creación de aplicaciones de visión artificial al proporcionar

herramientas para la gestión de datasets, el etiquetado de imágenes y el entrenamiento de modelos.

El primer dataset elegido contiene dos clases: "novehicle" y "vehicle". La categoría "novehicle" incluye elementos presentes en las vías que no son vehículos, como peatones, semáforos y señales de tránsito. Por otro lado, la categoría "vehicle" abarca distintos tipos de vehículos que pueden encontrarse en la carretera, como automóviles, buses, motocicletas, camionetas y camiones. Este dataset está compuesto por 2,601 imágenes de entrenamiento, 372 imágenes de validación y 557 imágenes de prueba, sumando un total de 3,530 imágenes.

Además, se seleccionó un segundo dataset con etiquetas más específicas: "bus", "car", "motorbike" y "truck". Este conjunto de datos incluye 2,380 imágenes de entrenamiento, 100 imágenes de validación y 100 imágenes de prueba, con un total de 2,580 imágenes. En la figura 3-24 se puede observar la distribución porcentual de ambos datasets.

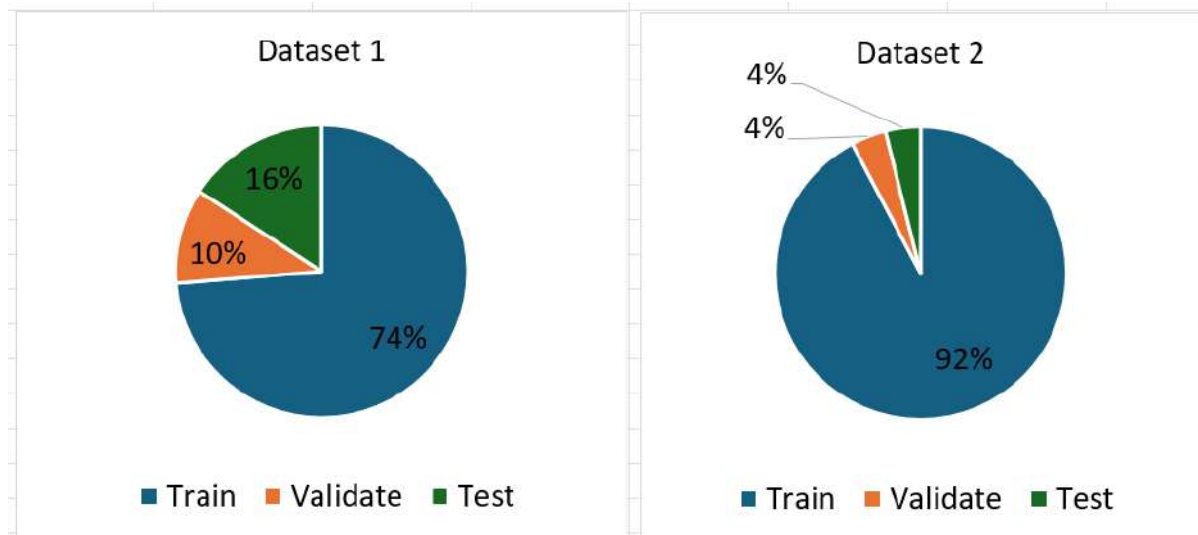


Figura 3-24. Distribución de porcentajes para cada data set.

Cabe mencionar que a cada dataset se le había realizado el data augmentation, es decir, se aumentaron los datos base realizando ciertas transformaciones que no alteran la información esencial, pero sí aportan variabilidad al conjunto de entrenamiento. Es importante recordar que los números totales señalados anteriormente pertenecen al total después de haber realizado el data augmentation.



Figura 3-25. Data augmentation de cada dataset.

3.3.2.3.2 Implementación de la red neuronal

Una vez seleccionado algoritmo de detección y el dataset se procede a realizar un entrenamiento, a este proceso se le conoce como aprendizaje por transferencia, donde se toma un modelo ya entrenado con miles de imágenes, pero se adapta a las necesidades del desarrollador, en este caso la detección de vehículos.

Durante el proceso de entrenamiento, se obtienen métricas interesantes que reflejan el proceso de aprendizaje y clasificación que refleja los diferentes modelos entrenados, las métricas utilizadas para evaluar el proceso de entrenamiento son las siguientes:

- **Box Loss (Pérdida de las Cajas):** Mide el error en la predicción de las coordenadas de los cuadros delimitadores. Evalúa qué tan bien el modelo predice la posición y tamaño de cada objeto.
- **Class Loss (Pérdida de Clasificación):** Evalúa el error en la clasificación de cada objeto detectado.
- **mAP50:** Precisión media con un umbral de IoU de 0.50, útil para evaluar detecciones sencillas.
- **mAP50-95:** la media de la precisión media calculada con distintos umbrales de IoU, que van de 0,50 a 0,95. Ofrece una visión global del rendimiento del modelo en distintos niveles de dificultad de detección.
- **P (Precisión):** La precisión de los objetos detectados, que indica cuántas detecciones fueron correctas.
- **Recall:** El recall, también conocido como sensibilidad o tasa de verdaderos positivos, es una métrica que evalúa la capacidad de un modelo de detección para identificar correctamente todos los ejemplos positivos en un conjunto de datos.

A continuación, se presentan los resultados y matrices de confusión obtenidos. La Tabla 3-4 muestra las métricas de los modelos YOLOv8m y YOLOv5l entrenados con el Dataset 1, que incluye las clases "Vehicle" y "Novehicle". La Tabla 3-5 presenta los resultados con el Dataset 2, que contiene las clases "bus", "car", "motorbike" y "truck". El entrenamiento se realizó en Google Colab utilizando las herramientas de Ultralytics.

Tabla 3-4. Métricas obtenidas después del entrenamiento con el Dataset 1.

Modelo	Clase	Épocas	Precisión	Recall	mAP50:	mAP50-95:
YOLOv5l	Vehicle	200	0.811	0.854	0.863	0.637
	Novehicle	200	0.686	0.275	0.335	0.195
YOLOv8m	Vehicle	200	0.781	0.844	0.85	0.635
	Novehicle	200	0.598	0.248	0.297	0.177

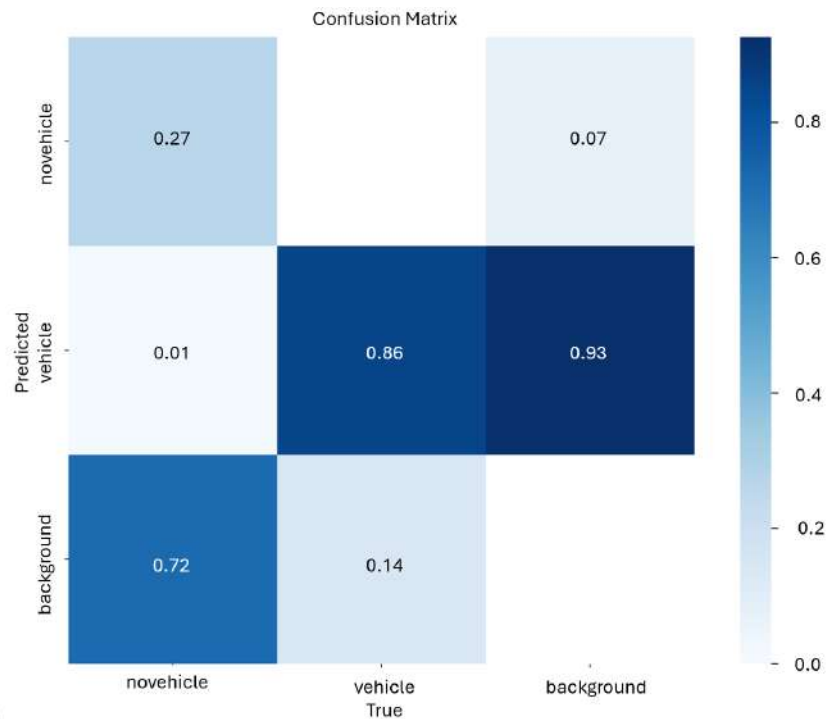


Figura 3-26. Matriz de confusión YOLOv5l con Dataset 1.

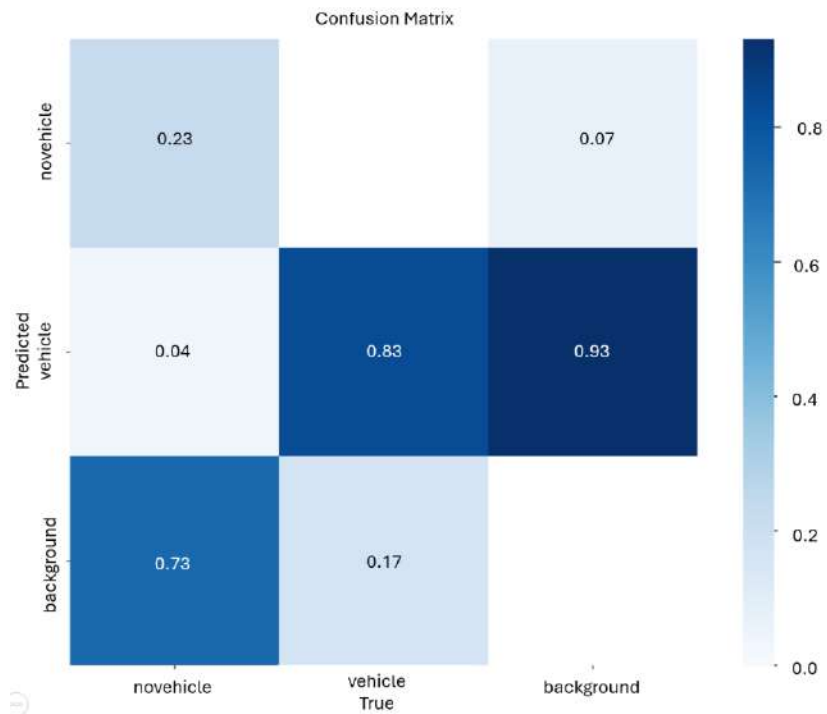


Figura 3-27. Matriz de confusión YOLOv8m con Dataset 1.

Tabla 3-5. Métricas obtenidas después del entrenamiento con el Dataset 2.

Modelo	Clase	Épocas	Precisión	Recall	mAP50:	mAP50-95:
YOLOv5l	Car	100	0.947	0.951	0.975	0.778
	Motorcycle	100	0.888	0.815	0.892	0.539
	Bus	100	0.896	0.928	0.937	0.856
	Truck	100	0.9	0.966	0.953	0.808
YOLOv8m	Car	100	0.933	0.948	0.98	0.814
	Motorcycle	100	0.903	0.775	0.874	0.53
	Bus	100	0.967	0.946	0.957	0.898
	Truck	100	0.93	0.968	0.971	0.85

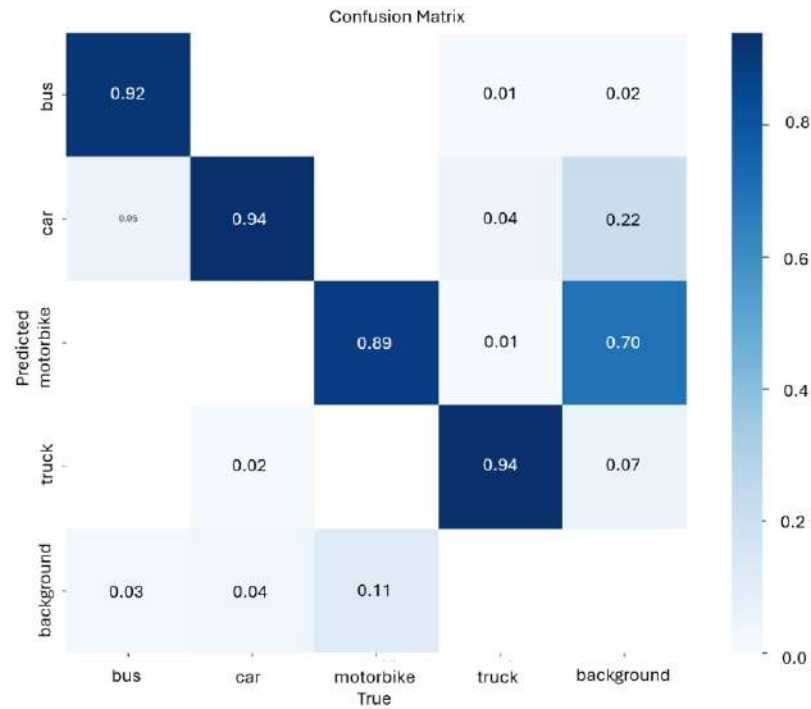


Figura 3-28. Matriz de confusión YOLOv5l Dataset 2.

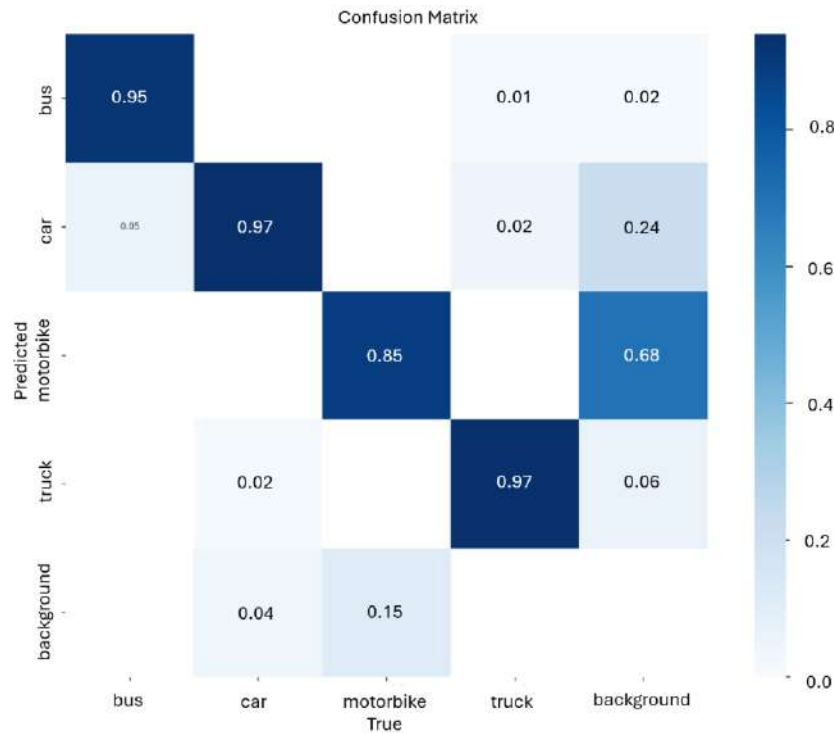


Figura 3-29. Matriz de confusión YOLOv8m con Dataset 2.

En la Tabla 3-4 se observa que los resultados para la clase **"Vehicle"** son superiores a los de la clase **"Novehicle"**, lo cual era de esperarse. Esto se debe a que, en el Dataset 1, los objetos pertenecientes a la clase **"Novehicle"** presentan una gran variabilidad entre sí, lo que dificulta que la red neuronal extraiga características comunes. Por ejemplo, una señal de tránsito, un cartel de carretera, un peatón y un semáforo tienen diferencias significativas en su apariencia. En contraste, la clase **"Vehicle"** muestra mejores resultados debido a la mayor similitud entre las imágenes, ya que los automóviles, camiones, buses y motocicletas comparten múltiples características en común.

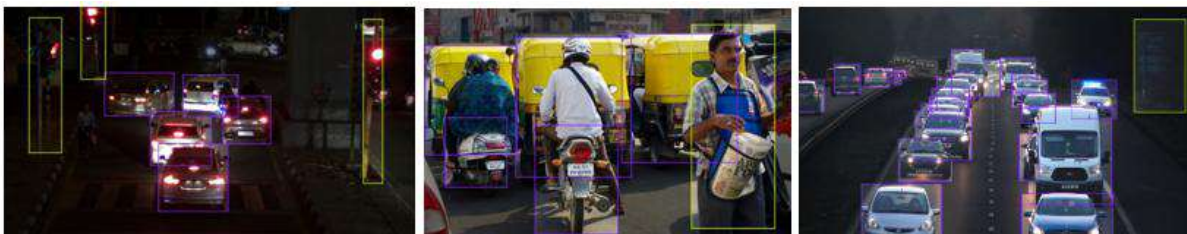


Figura 3-30. Imágenes del dataset 1 (bounding box amarilla = nonehicle, bounding box morado = vehicle).

Sin embargo, al analizar la matriz de confusión (figura 3-26 y figura 3-27) tanto de **YOLOv5l** como de **YOLOv8m**, se puede notar que la clase **"Vehicle"** presenta dificultades para diferenciarse del fondo (**"Background"**), lo que sugiere que la red neuronal tiene problemas para distinguir ciertos vehículos en entornos específicos.

Por otro lado, en la Tabla 3-5 se observa una mejora significativa en los resultados para la mayoría de las clases. Esto se debe a un cambio en la estrategia de entrenamiento: primero, la red neuronal fue entrenada durante **50 épocas**, y posteriormente, el modelo resultante fue reentrenado por otras **50 épocas**, congelando **10 capas** de la red. Este enfoque permitió un ajuste más preciso de los pesos para cada clase, lo cual se refleja claramente en la matriz de confusión (figura 3-28 y figura 3-29).

Ya con la red neuronal entrenada se creó el nodo “yolo_inference_publisher” este se suscribe a la panorámica RGB, donde con las imágenes de input publica un mensaje personalizado que contiene los datos del bounding box, el centroide y el ID de cada inferencia.

3.3.2.4 Extracción de características y seguimiento de vehículos

Como se explicó en el capítulo anterior, la detección de vehículos se realizó utilizando la red neuronal YOLO (You Only Look Once), la cual se basa en redes neuronales convolucionales (CNN) lo que permite la extracción y procesamiento de características en las imágenes.

En YOLO, la extracción de características inicia en las capas convolucionales, detectando patrones simples que se combinan en mapas complejos para reconocer partes de los vehículos. Su arquitectura profunda permite una detección rápida y precisa. Para el seguimiento, se revisaron varios algoritmos compatibles con redes neuronales como YOLO, y se eligió BYTETrack. En esta sección se explican las razones de esta elección y se presentan resultados en escenarios reales del proyecto.

Para entender mejor este algoritmo, es importante conocer sus principios. Se trata de un método de seguimiento de múltiples objetos (MOT) que optimiza la asociación de detecciones entre cuadros de video. A diferencia de otros enfoques que priorizan solo las detecciones con puntuaciones altas, este algoritmo considera también las de puntuación baja, lo que mejora el seguimiento en casos de solapamiento o detecciones débiles.

Un **tracklet** es una secuencia corta de detecciones de un mismo objeto a lo largo de varios cuadros. BYTETrack calcula estos tracklets y los emplea para mejorar la asociación de objetos en cuadros posteriores. Gracias a esta estrategia, es capaz de identificar correctamente objetos que de otro modo podrían confundirse con el ruido de fondo.

El proceso de seguimiento en ByteTrack se divide en cuatro etapas principales, así mismo la figura 3-31 sirve para entender su flujo de trabajo:

- **Detección:** En la primera fase, se identifican los objetos en el video utilizando un detector de objetos [38].
- **Tracklets:** Luego, se inicializan los **tracklets** para cada objeto detectado. Un tracklet es una secuencia de cuadros de detección asociados al mismo objeto. Para rastrear el movimiento de cada objeto a lo largo del tiempo, ByteTrack emplea un filtro de Kalman [38].
- **Asociación:** En esta etapa, se vinculan los cuadros de detección con los tracklets existentes. ByteTrack realiza esta asociación en dos pasos:
 - **Primer paso:** Se asocian los cuadros de detección con puntuaciones altas a los tracklets, utilizando el coeficiente de superposición **IoU** (Intersección sobre Unión) como criterio de coincidencia [38].

- **Segundo paso:** Se asocian los cuadros de detección con puntuaciones más bajas a los tracklets, basándose en las similitudes entre ellos. Estas similitudes pueden calcularse a partir de características extraídas tanto de los cuadros de detección como de los tracklets [38].
- **Actualización del estado de los tracklets:** Finalmente, se actualiza el estado de cada tracklet, incluyendo la identidad del objeto, la posición del cuadro delimitador y el puntaje de confianza. Esta actualización se realiza nuevamente mediante el **filtro de Kalman**, lo que permite mejorar la precisión del seguimiento [38].

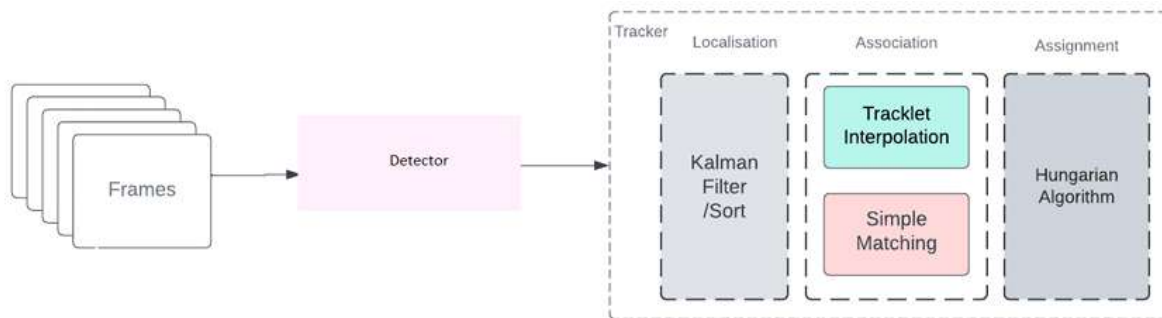


Figura 3-31. ByteTrack workflow Imagen tomada de [38].

ByteTrack se destaca por ser una arquitectura simple y eficiente para el seguimiento de objetos en videos. Su enfoque permite recuperar objetos reales que de otro modo podrían haberse perdido y manejar de manera efectiva situaciones como **cambios en la apariencia de los objetos y oclusiones** [38].

Una vez entendido el funcionamiento del algoritmo, es importante definir las métricas para evaluar su desempeño frente a otros métodos de seguimiento. Se utilizará MOTA (Múltiple Object Tracking Accuracy), que mide la precisión general del seguimiento considerando tres factores clave que afectan su calidad

Falsos positivos (FP): Cuando el sistema detecta un objeto que en realidad no existe.

Falsos negativos (FN): Cuando el sistema no detecta un objeto que sí está presente.

Cambios de identidad (IDs): Cuando un objeto rastreado cambia erróneamente su identificador durante la secuencia [36].

El cálculo de MOTA se expresa mediante la ecuación 3.1:

$$MOTA = 1 - \frac{FN + FP + ID's}{GT} \quad (3.1)$$

Donde:

- **FN** = Falsos negativos (objetos no detectados).
- **FP** = Falsos positivos (detecciones erróneas).
- **IDs** = Cambios de identidad en el seguimiento.
- **GT** = Número total de objetos reales (ground truth).

Otra métrica importante a tener en cuenta es **IDF1** (*Identity F1-score*) mide la precisión del seguimiento de objetos al evaluar cuán bien un sistema mantiene la identidad de los objetos a lo largo de una secuencia de video. Se calcula utilizando la precisión y el recall de identidad, como se muestra en la ecuación 3.2 [52]:

$$IDF1 = \frac{2 \times IDTP}{2 \times IDTP + IDFP + IDFN} \quad (3.2)$$

Donde:

- **IDTP** (*Identity True Positives*): Detecciones correctamente asociadas a la identidad real del objeto.
- **IDFP** (*Identity False Positives*): Detecciones incorrectamente asignadas a una identidad errónea.
- **IDFN** (*Identity False Negatives*): Objetos rastreados que pierden su identidad en algún momento de la secuencia.

A diferencia de MOTA, que mide la precisión general del seguimiento considerando falsos positivos y negativos, IDF1 evalúa la consistencia de identidad a lo largo del tiempo. Un IDF1 alto indica que el algoritmo mantiene correctamente la identidad de los objetos sin confundirlos ni perderlos [52].

Por otro lado la métrica HOTA (High Order Tracking Accuracy) ofrece una evaluación más completa del desempeño de los algoritmos de seguimiento, al integrar tanto la precisión en la detección como la consistencia en la asociación de identidades. Se basa en el IoU entre los cuadros predichos y los reales, y analiza no solo la exactitud en cada fotograma, sino también qué tan bien se mantiene la identidad de los objetos rastreados a lo largo del tiempo.[53].

Se calcula combinando los siguientes factores:

1. **Precisión de detección:** Evalúa qué tan bien se detectan los objetos en cada cuadro.
2. **Consistencia de la identidad:** Mide qué tan correctamente se asocian los objetos en toda la secuencia de video.
3. **Penalización por errores en la trayectoria:** Penaliza cambios de identidad y objetos perdidos.

HOTA promedia estos valores en múltiples umbrales de IoU para obtener un único puntaje que refleje el desempeño general del algoritmo de seguimiento.

Teniendo en cuenta las métricas anteriormente mencionadas hay estudios en los que BYTETrack se evalúa con diferentes bases de datos y muestra resultados altamente positivos, el estudio realizado en [54] ejecuto pruebas con BYTETrack usando el dataset MOT17 para evaluar el desempeño del mismo y compararlo con otros trackers obteniendo los resultados mostrados en la figura 3-23:

Tabla 3-6. Comparación de los métodos de última generación bajo el protocolo “detector privado” en el conjunto de pruebas MOT17. Resultados obtenidos de [50].

Tracker	MOTA↑	IDF1↑	HOTA↑	FP↓	FN↓	IDs↓	FPS↑
DAN [61]	52.4	49.5	39.3	25423	234592	8431	<3.9
Tube.TK [46]	63.0	58.6	48.0	27060	177483	4137	3.0
MOTR [80]	65.1	66.4	-	45486	149307	2049	-
CTracker [48]	66.6	57.4	49.0	22284	160491	5529	6.8
CenterTrack [89]	67.8	64.7	52.2	18498	160332	3039	17.5
QuasiDense [47]	68.7	66.3	53.9	26589	146643	3378	20.3
TraDes [71]	69.1	63.9	52.7	20892	150060	3555	17.5
MAT [25]	69.5	63.1	53.8	30660	138741	2844	9.0
SOTMOT [87]	71.0	71.9	-	39537	118983	5184	16.0
TransCenter [75]	73.2	62.2	54.5	23112	123738	4614	1.0
GSDT [67]	73.2	66.5	55.2	26397	120666	3891	4.9
Semi-TCL [32]	73.3	73.2	59.8	22944	124980	2790	-
FairMOT [85]	73.7	72.3	59.3	27507	117477	3303	25.9
RelationTrack [78]	73.8	74.7	61.0	27999	118623	1374	8.5
PermaTrackPr [63]	73.8	68.9	55.5	28998	115104	3699	11.9
CSTrack [33]	74.9	72.6	59.3	23847	114303	3567	15.8
TransTrack [59]	75.2	63.5	54.1	50157	86442	3603	10.0
FUFET [54]	76.2	68.0	57.9	32796	98475	3237	6.8
SiamMOT [34]	76.3	72.3	-	-	-	-	12.8
CorrTracker [65]	76.5	73.6	60.7	29808	99510	3369	15.6
TransMOT [15]	76.7	75.1	61.7	36231	93150	2346	9.6
ReMOT [76]	77.0	72.0	59.7	33204	93612	2853	1.8
ByteTrack (nuestro)	80.3	77.3	63.1	25491	83721	2196	29.6

ByteTrack se presenta como un rastreador altamente eficiente, logrando un **MOTA de 80.3**, un **IDF1 de 77.3** y un **HOTA de 63.1** en el conjunto de prueba **MOT17**, con un rendimiento de casi **30 FPS**. Estos resultados lo posicionan en el primer lugar entre todos los rastreadores de la clasificación tenidos en cuenta a la hora de realizar la prueba. Estos resultados muestran que una de las principales fortalezas de **ByteTrack** es su capacidad para manejar la **oclusión**, gracias a su alta precisión en la detección y su estrategia de asociación que incorpora cuadros de detección de baja puntuación. Además, este enfoque demuestra cómo aprovechar al máximo los resultados de detección para mejorar el rendimiento del **seguimiento de múltiples objetos** [54].

Con su **alta precisión, velocidad y simplicidad**, ByteTrack se consolida como una solución eficaz para problema de seguimiento de vehículos de este trabajo.

Así mismo se realizaron unas pequeñas pruebas usando ByteTrack con videos tomados desde un celular en un carro conduciendo durante el día logrando tener buenos resultados pues en la secuencia del video nunca se repitieron los ID's ni se intercambiaron entre ellos (figura 3-32 y figura 3-33).



A

B

Figura 3-32. Fotogramas A y B con algoritmo BYTETRACK implementado en video de carretera. Se conserva el ID:2



C

D

Figura 3-33 Fotogramas C y D con algoritmo BYTETRACK implementado en video de carretera. Se conserva el ID:9

Ahora bien, para implementar el tracker Bytetrack con la red neuronal solo es necesario modificar la línea que inicializa el modelo de la siguiente forma: **results = model.track(source = img, tracker = 'bytetrack.yaml')**, de este modo se inicializa la red neuronal para que en la imagen panorámica detecte los objetos y al mismo tiempo le añade a cada uno su ID correspondiente.

3.3.2.5 Implementación modelo de Kalman

Para realizar la predicción de trayectorias de objetos en movimiento dentro de una secuencia de imágenes, se implementó una clase denominada KalmanTracker(), la cual utiliza un Filtro de Kalman para estimar y predecir la posición futura de un objeto.

Cada objeto detectado se identifica mediante un *unique_id* entregado por las inferencias de la red neuronal YOLO con *ByteTrack* y se asocia a una instancia única de la clase *KalmanTracker()*. Este seguimiento se realiza en el espacio bidimensional de la imagen panorámica, estimando la posición y velocidad del objeto.

El filtro de Kalman implementado posee las siguientes variables y sus correspondientes inicializaciones:

- **Estado (dim_x=4):** Representado por el vector $[x, y, v_x, v_y]$, donde x y y son las coordenadas del objeto y v_x , v_y corresponden a su velocidad en cada eje.
- **Medición (dim_z=2):** Compuesta por $[x, y]$, es decir, las coordenadas detectadas del objeto.

Durante la inicialización, se definen las siguientes matrices del filtro:

- **F (matriz de transición):** Modela cómo evoluciona el estado entre frames.

```
self.kf.F = np.array([[1, 0, 1, 0],  
                      [0, 1, 0, 1],  
                      [0, 0, 1, 0],  
                      [0, 0, 0, 1]])
```
- **H (matriz de observación):** Describe la relación entre el estado y la medición observada.

```
self.kf.H = np.array([[1, 0, 0, 0],  
                      [0, 1, 0, 0]])
```
- **P (matriz de covarianza):** Representa la incertidumbre del estado inicial. Donde P es una matriz identidad 4x4 que se multiplica por 1000 para que el filtro de Kalman se adapte más rápidamente a las mediciones reales y no se aferre a su suposición inicial.

```
self.kf.P = np.array([[1, 0, 1, 0],  
                      [0, 1, 0, 1],  
                      [0, 0, 1, 0],  
                      [0, 0, 0, 1]])  
  
self.kf.P *= 1000
```
- **Q (ruido del proceso):** Modela la incertidumbre inherente al movimiento del objeto.

```
self.kf.Q = np.eye(4) * 0.005
```
- **R (ruido de medición):** Representa el error asociado a la medición observada.

```
self.kf.R = np.array([[2, 0],  
                      [0, 2]])
```

Además, cada instancia mantiene un historial de las últimas posiciones detectadas mediante una estructura con una longitud máxima de 10 elementos. El procedimiento de seguimiento usando el filtro de Kalman funciona como se muestra en el Algoritmo 3.1:

// Inicialización

PARA cada objeto detectado EN el frame ACTUAL:

SI `unique_id` NO está en el diccionario de rastreadores:

Crear nueva instancia de KalmanTracker

Agregar al diccionario con clave `unique_id`

// Actualizar posición y filtro

`centroid` ← posición actual del objeto

Actualizar KalmanTracker correspondiente con `centroid`

Agregar `centroid` al historial de posiciones del objeto

// Estimar dirección y proyectar trayectoria

SI historial de posiciones tiene al menos 7 elementos:

`direcciones` ← lista vacía

PARA `i` desde 1 hasta longitud del historial - 1:

`dirección_i` ← `historial[i] - historial[i - 1]`

Agregar `dirección_i` a `direcciones`

`dirección_promedio` ← `promedio(direcciones)`

```
FACTOR ← 25 // Este factor amplifica la proyección. Este valor se eligió para dar
mayor visibilidad a la proyección pues inicialmente es corta.
punto_proyectado ← último centroid + (dirección_promedio * FACTOR)

// Visualización
Dibujar línea desde último centroid hasta punto_proyectado en la imagen
Dibujar un círculo en punto_proyectado

// Limpieza de objetos no detectados en este frame
PARA cada unique_id en el diccionario de rastreadores:
  SI unique_id NO fue detectado en este frame:

    Eliminar unique_id del diccionario
```

Algoritmo 3.1. Funcionamiento filtro de Kalman.

3.3.2.6 Evaluación peligro de colisión

Para evaluar el nivel de peligro, se usan algoritmos de percepción basados en el Tiempo hasta la Colisión (TTC), que calcula cuánto tardarían en colisionar dos vehículos si mantienen su velocidad y trayectoria. También se establecen distancias de seguridad frontales y laterales; si un vehículo se acerca demasiado, se activa una alerta visual marcándolo con un contorno rojo en la imagen.

Cálculo del Tiempo hasta la Colisión (TTC)

El **TTC** se obtiene a partir de la relación entre la **distancia relativa entre los vehículos** y la **velocidad de aproximación**. Cuando este valor cae por debajo de un **umbral crítico**, definido según la capacidad de percepción y reacción humana, el sistema genera una **advertencia temprana** para alertar al conductor sobre un posible accidente.

Este cálculo depende de dos factores fundamentales:

1. **Distancia entre los vehículos:** Se obtiene mediante sensores o cámaras que capturan la posición relativa de los objetos en el entorno del conductor.
2. **Velocidad del vehículo propio:** Se estima a partir de los datos de aceleración y desplazamiento del vehículo que se está conduciendo.

En este proyecto, el cálculo del TTC se basa en datos obtenidos por cámaras estéreo ZED, que generan mapas de profundidad y permiten detectar la posición 3D de los vehículos en la vista panorámica. Estas cámaras también proporcionan mediciones de distancia y, en el caso de las ZED2, incluyen un acelerómetro que permite estimar la velocidad del vehículo propio. Las siguientes secciones abordan los cálculos de distancia y velocidad en detalle.

Asimismo, la estimación de colisiones incorpora un filtro de Kalman para predecir las trayectorias de los objetos. Cuando estas trayectorias interceptan una región de interés (ROI,

por sus siglas en inglés) previamente definida, se genera una alerta. Este proceso se detalla más adelante.

3.3.2.5.1 Cálculo de distancia

La cámara de profundidad **ZED2** es un dispositivo de medición pasiva que no incorpora un sensor de profundidad activo, ya que no cuenta con un proyector láser infrarrojo (*IR*). En su lugar, utiliza un **sistema de visión estereoscópica** basado en una configuración de **cámara binocular**, lo que le permite capturar escenas en tres dimensiones [55].

La profundidad se calcula a partir de la disparidad entre imágenes captadas por ambas lentes, usando un algoritmo de coincidencia estéreo para comparar puntos equivalentes. Con esta información y los parámetros del sensor, se genera un mapa de profundidad que estima con precisión la distancia a los objetos. La cámara ZED2 proporciona imágenes de profundidad en 32 bits, mejorando la precisión. [55].

El sensor de profundidad **ZED2** está optimizado para el cálculo de profundidad en tiempo real mediante la tecnología **CUDA** de **NVIDIA**. Por ello, su funcionamiento requiere una **GPU** y un hardware computacional adecuado.

```
---
header:
  stamp:
    sec: 0
    nanosec: 0
  frame_id: ''
distance_measure:
- id: 1
  distance: 1.40347314453125
- id: 4
  distance: 1.4183009033203127
```

Figura 3-34. Ejemplo de publicación del mensaje que contiene ID y distancia correspondiente.

Para obtener la distancia del entorno con respecto a la cámara ZED, se pueden utilizar dos métodos: el **mapa de profundidad** de 32 bits o la **nube de puntos**. En el primer caso, la distancia se obtiene a través del SDK de ZED con el comando **zed.retrieve_measure(depth_map, sl.MEASURE.DEPTH)**, que devuelve el mapa de profundidad completo, y luego con **depth_value = depth_map.get_value(x, y)**, se obtiene la distancia en un punto específico. En el segundo método, se genera una nube de puntos en la que cada píxel contiene información tridimensional, utilizando **zed.retrieve_measure(point_cloud, sl.MEASURE.XYZRGBA)**. Para calcular la distancia en un punto específico (i, j), se extraen sus coordenadas espaciales con **point3D = point_cloud.get_value(i, j)** y se aplica la ecuación euclidiana **distance = math.sqrt(point3D[0]**2 + point3D[1]**2 + point3D[2]**2)**, donde X, Y y Z representan la posición del punto en el espacio tridimensional. Ambos métodos permiten obtener información precisa sobre la profundidad y la distancia de los objetos en la escena. Finalmente, como resultado en la figura 3-26 se puede observar el mensaje custom que contiene los datos de distancia asociada al ID del objeto detectado.

3.3.2.6.2 Cálculo de velocidad

La velocidad se estima en el nodo *imu_velocity* mediante la clase “VelocityEstimator”, que integra los datos de aceleración del sensor inercial (IMU) de la cámara frontal ZED2. Esta IMU, proporciona datos de aceleración en los ejes [X, Y, Z], donde se utiliza principalmente el eje Z. El cálculo se realiza en el método *accel_callback*, activado al recibir nuevos datos del tópico *//linear_acceleration*, que contiene la aceleración en Z.

El procedimiento para estimar la velocidad se desarrolla de acuerdo con el Algoritmo 3.2:

```
//Inicializar:
velocity_mps ← 0.0
last_time ← tiempo_actual()
last_movement_time ← last_time
noise_threshold ← 0.9 // Valor basado en pruebas: el sensor puede registrar
aceleraciones cercanas a  $\pm 0.9$  m/s2 estando en reposo, por lo tanto, se considera ruido y
se descarta cualquier lectura inferior a este umbral.
friction_factor ← 0.67 // Determinado empíricamente: se probaron valores entre 0 y 1.
Se encontró que entre 0.6 y 0.7 la desaceleración simulada era mejor y permitía una
detención gradual sin acumular error.
reset_threshold ← 1.0

Al recibir nueva lectura de aceleración (accel):
current_time ← tiempo_actual()
dt ← current_time - last_time
last_time ← current_time
acceleration ← -accel // invertir si es necesario

// Filtrado de ruido
Si |acceleration| < noise_threshold Entonces
    acceleration ← 0.0

// Integración de la aceleración
velocity_mps ← velocity_mps + (acceleration × dt)

// Simulación de fricción
Si acceleration = 0.0 Entonces
    velocity_mps ← velocity_mps × (1 - friction_factor × dt)

// Actualizar último tiempo de movimiento
Si acceleration ≠ 0.0 Entonces
    last_movement_time ← current_time

// Reinicio automático por inactividad
Si (current_time - last_movement_time) > reset_threshold Entonces
    velocity_mps ← 0.0
```

// Filtrado final de pequeñas variaciones

Si $|velocity_mps| < 0.4$ Entonces
 $velocity_mps \leftarrow 0.0$

Algoritmo 3.2. Funcionamiento del cálculo de la velocidad.

Esta estrategia de estimación permite obtener una aproximación de la velocidad real sin necesidad de contar con encoders, GPS u otros sensores externos. Al integrar directamente los datos de aceleración de la cámara.

3.3.2.6 Alerta de colisión

La alerta de colisión se implementa en un nodo ROS 2 llamado "CollisionEstimatorNode", que combina datos visuales, de profundidad y velocidad para generar alertas predictivas. Además, usa filtros de Kalman para predecir la trayectoria de los objetos detectados.

El nodo se apoya en diversas librerías y mensajes de ROS2 para realizar sus funciones:

- **shapely**: para el manejo de geometrías y cálculo de intersecciones.
- **filterpy.kalman**: para la implementación de filtros de Kalman.
- **numpy**: para operaciones matemáticas y vectoriales.
- **Mensajes ROS 2 personalizados y estándar**, como Float32, DistanceMeasure, Yolov8Inference y String, que permiten recibir y publicar datos esenciales como aceleración, distancias, inferencias visuales y alertas.

Para calcular la trayectoria de cada objeto, se utiliza la clase *KalmanTracker*, que asigna una instancia del filtro de Kalman a cada objeto identificado. Este filtro suaviza la trayectoria y predice posiciones futuras basándose en su historial de movimiento, lo cual es esencial para anticipar si un objeto se dirige hacia la Región de Interés (ROI) (en la sección 3.3.2.4 se explica a profundidad el funcionamiento del filtro Kalman).

Durante la inicialización, del nodo responsable de la alerta de colisión, este se suscribe a los siguientes tópicos:

- **/linear_acceleration**: para calcular la velocidad del sistema (publicado por el nodo ZED_PUBLISHER)
- **/object_depths**: para obtener la profundidad de cada objeto detectado (publicado por el nodo de distance_node).
- **/Yolov8_inference**: para recibir las posiciones de los objetos (centroides) detectados (publicado por Yolo_inference_publisher).

También publica en el tópico /collision_alert, donde emite alertas generadas por posibles colisiones.

Se definen variables internas para almacenar centroides, distancias, filtros de Kalman, y un historial de alertas. Además, se configura una ROI, que es una región crítica en la imagen donde se monitorean objetos que podrían representar una amenaza directa de colisión.

Los niveles de alerta (verde, amarillo, naranja, rojo) se determinan en función del "tiempo hasta colisión" (Tt) (tabla 3-6), el cual se calcula como la distancia entre el vehículo y el objeto

dividido entre la velocidad actual. Estos niveles se pueden personalizar mediante un archivo de configuración YAML.

Tabla 3-7. Niveles de alerta de acuerdo con el Tiempo de Colisión.

Tiempo de colisión (Definidos por defecto)	Nivel de alerta
> 5 segundos	“verde”
3 – 5 segundos	“amarillo”
2 – 3 segundos	“naranja”
<= 2 segundos	“rojo”

Evaluación de posibles colisiones

La lógica principal del nodo reside en su capacidad para analizar cada objeto en función de su distancia, su posición en la imagen y su trayectoria proyectada como se observa en el Algoritmo 3.3:

Para cada objeto detectado:

Si objeto tiene una distancia válida y un ID válido Entonces:

Calcular T_t (tiempo para colisión) usando: $T_t = \text{distancia} / \text{velocidad_actual}$

Determinar el nivel de alerta según T_t :

Si $T_t > \text{umbral_verde}$ → alerta ← “verde”

Si T_t entre naranja y rojo → alerta ← “naranja”

Si $T_t < \text{umbral_rojo}$ → alerta ← “rojo”

Si el objeto ha sido observado por suficiente tiempo (por ejemplo, ≥ 7 cuadros):

Proyectar su trayectoria futura usando el filtro de Kalman

Si la trayectoria proyectada intercepta la Región de Interés (ROI):

Elevar el nivel de alerta, incluso si la distancia actual no es crítica

Si la distancia del objeto es muy cercana:

Aplicar lógica más estricta para elevar el nivel de alerta

Algoritmo 3.3. Funcionamiento para la evaluación de la colisión.

Finalmente, todas las alertas generadas se publican, y el nodo de “YoloSuscribirNode” se encarga de recibir esa información y graficarla sobre la panorámica rgb.

3.4 CONCLUSIONES

La implementación del aplicativo desarrollado representa una solución integral y robusta para la percepción del entorno y la evaluación de riesgos en contextos dinámicos de conducción. El sistema se construyó a partir de un diseño conceptual claro, en el que se definieron tanto los requerimientos funcionales como los no funcionales, estableciendo así las bases para su operación. Esta etapa inicial fue complementada con un diagrama conceptual que ofreció una visión estructurada de la arquitectura general, facilitando la comprensión de cómo interactúan sus distintos componentes.

En fases posteriores, se desarrollaron los diagramas de clases y de nodos. El primero permitió identificar las entidades principales del sistema junto con sus atributos y métodos, brindando una visión detallada de la organización del código. El segundo explicó la interacción entre los distintos procesos, especificando qué información produce y utiliza cada nodo, así como su función dentro del ecosistema del aplicativo.

La interfaz gráfica del sistema se diseñó en torno a dos módulos principales: el módulo de configuración y el de procesamiento. El primero permite al usuario ajustar parámetros esenciales como la selección de la red neuronal, los niveles de alerta para detección de colisiones y opciones del algoritmo ByteTrack, encargado del seguimiento de objetos. Además, contempla la calibración de las cámaras y la configuración de homografías, fundamentales para alinear correctamente las imágenes desde distintas perspectivas.

Por su parte, el módulo de procesamiento gestiona la selección de detectores de vehículos, la definición del conjunto de datos de entrenamiento y la lógica de seguimiento. Estos elementos se integran en el módulo de estimación de colisiones, que utiliza filtros de Kalman para predecir trayectorias de objetos en movimiento, permitiendo así anticipar escenarios de riesgo.

Finalmente, la combinación de la estimación de trayectorias con una región de interés (ROI) en el campo visual permite calcular el tiempo estimado de colisión (TTC), lo que habilita un sistema de alertas tempranas para advertir al conductor antes de que ocurra una situación peligrosa. Esta integración de visión por computador, aprendizaje profundo, seguimiento de objetos y predicción de trayectorias demuestra no solo la viabilidad técnica del sistema, sino también su potencial para evolucionar hacia soluciones de asistencia a la conducción más avanzadas y seguras.

4. RESULTADOS Y ANÁLISIS

4.1 INTRODUCCIÓN

El siguiente capítulo expone el procedimiento usado para evaluar la calidad y funcionalidad del aplicativo, con el propósito de obtener un resultado confiable y correcto que responda a las expectativas del usuario final. Se presentan los resultados obtenidos a partir de las pruebas de integración y de campo. Las pruebas de integración permiten validar la interacción entre los diferentes componentes de cada módulo, asegurando la correcta operatividad de la interfaz y la modificación de las configuraciones de acuerdo con lo solicitado por el usuario. Finalmente, las pruebas de campo se llevan a cabo en condiciones reales o cercanos a la realidad, con el objetivo de verificar el funcionamiento del aplicativo para alertar y funcionar adecuadamente en entornos operativos.

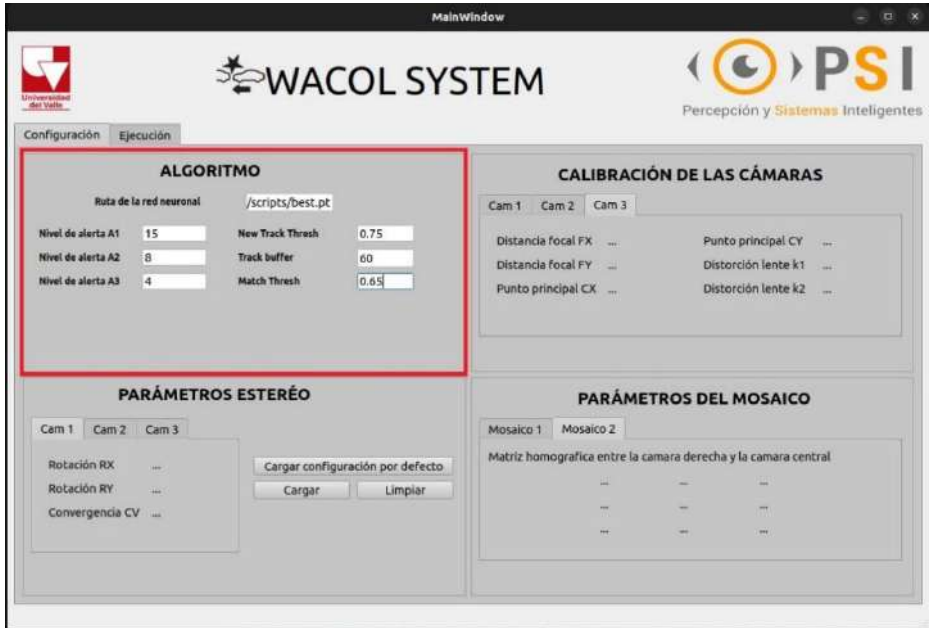
4.3 PRUEBAS DE INTEGRACIÓN.

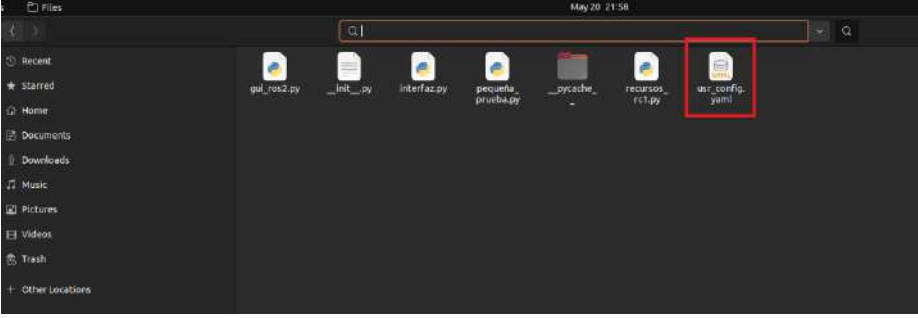
Antes de empezar es necesario aclarar que las pruebas tanto de integración como de campo no se realizaron en la Nvidia AGX Orin, esto debido a un problema de ancho de banda donde las cámaras no lograban abrir bien las 3 al mismo tiempo, siempre se lograba levantar dos de las cámaras teniendo un desempeño normal, pero cuando se lograba levantar las 3, presentaba un bajo rendimiento con fotogramas lentos. Por lo que se usó un computador portátil con las siguientes especificaciones:

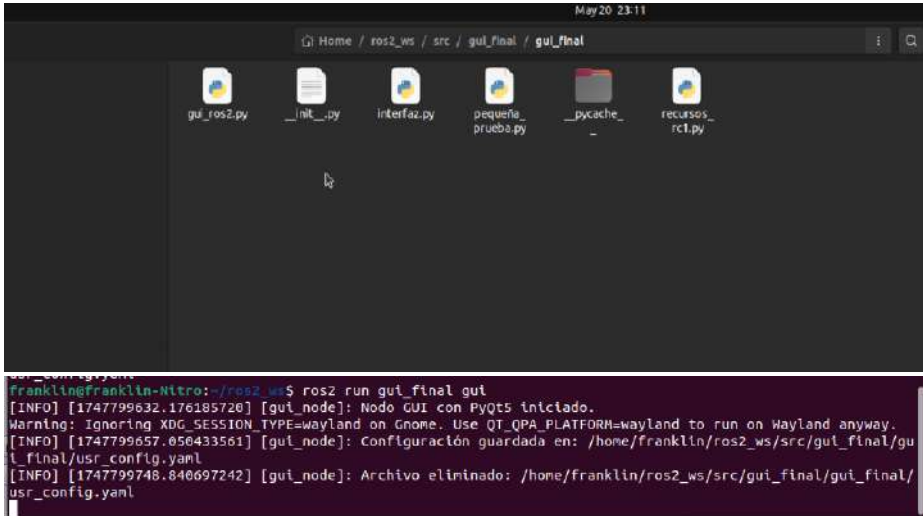
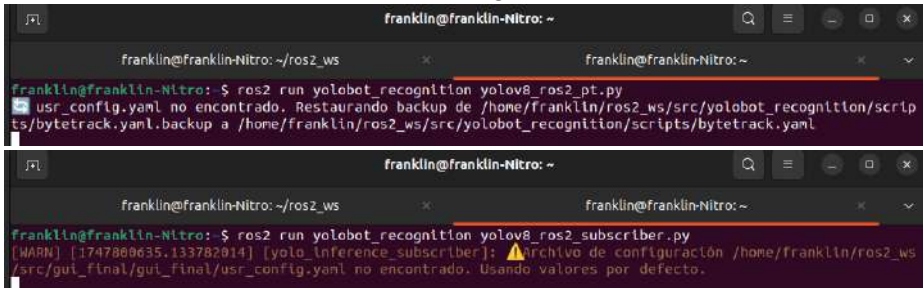
- Procesador (CPU): Intel core i5 10300 H.
- Tarjeta gráfica (GPU): Nvidia GTX 1650 Mobile.
- Memoria RAM: 16 GB.
- Memoria RAM de la GPU: 4 GB.
- Sistema Operativo: Ubuntu 22.04.

Tabla 4-1. Pruebas de integración Modulo de Configuración.

PRUEBA DE INTEGRACIÓN – MÓDULO DE CONFIGURACIÓN	
• Modificar la red neuronal, los niveles de alerta y parámetros del tracker. • Eliminar el “usr_config.yaml” cuando se quiera cargar la configuración. • Visualizar los parámetros extrínsecos e intrínsecos de las cámaras, las homografías y los parámetros estéreo.	
Tipo de prueba	Integración
Hardware Requerido	Portátil: • Procesador (CPU): Intel core i5 10300 H. • Tarjeta gráfica (GPU): Nvidia GTX 1650 Mobile. • Memoria RAM: 16 GB. • Memoria RAM de la GPU: 4 GB. 3 cámaras ZED2

Software requerido	Aplicativo de alerta de colisión con otros vehículos, (Python 3.10.12), ZED SDK, Ubuntu 22.04.5 LTS, Qt designer 5, CUDA, NVCC.
Objetivo	Validar el correcto funcionamiento del módulo de configuración.
Modificar la red neuronal a usar, los niveles de alerta y parámetros del tracker.	
Objetivo	Validar el correcto funcionamiento del aplicativo para generar el archivo “.yaml” que contenga los valores de configuración a usar.
Datos de la prueba	En esta prueba se llenan los datos requeridos en los qlabel en la pestaña configuración para ingresar el path que lleva a la red neuronal a utilizar, así como los valores para el tracker y los niveles de alerta. Yolo path: /home/franklin/ros2_ws/src/yolobot_recognition/scripts/best.pt Nivel de alerta A1: 15 segundos Nivel de alerta A2: 8 segundos Nivel de alerta A3: 4 segundos New Track Thresh: 0.75 Track Buffer: 60 Match Thresh: 0.65
Procedimiento	1. Ingresar el path de la red neuronal que se desea utilizar. 2. Ingresar los valores para los niveles de alerta. 3. Ingresar los parámetros del tracker.
Resultado esperado	1. Se genera el archivo “usr_config.yaml”. 2. Se revisa internamente que los valores se han cargado exitosamente en los nodos correspondientes.
Resultados obtenidos	Se ingresan los valores deseados para cada parámetro. 

	Se genera el archivo “usr_config.yaml” con los valores insertados. <pre>franklin@franklin-Nitro:~/ros2_ws\$ ros2 run gui_final gui [INFO] [1747799632.176185720] [gui_node]: Nodo GUI con PyQt5 iniciado. Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run on Wayland anyway. [INFO] [1747799657.050433561] [gui_node]: Configuración guardada en: /home/franklin/ros2_ws/src/gui_final/gui_final/usr_config.yaml</pre>  Revisión de uso exitoso del archivo “usr_config.yaml” Se carga la red neuronal y los parámetros del bytetrack <pre>franklin@franklin-Nitro: ~ franklin@franklin-Nitro: ~/ros2_ws franklin@franklin-Nitro: ~\$ ros2 run yolobot_recognition yolov8_ros2_pt.py [INFO] [1747796442.719978895] [yolo_subscriber]: Modelo YOLO cargado desde configuración: /home/franklin/ros2_ws/src/yolobot_recognition/scripts/best.pt Copia de seguridad creada: /home/franklin/ros2_ws/src/yolobot_recognition/scripts/bytetrack.yaml.backup Configuración guardada en /home/franklin/ros2_ws/src/yolobot_recognition/scripts/bytetrack.yaml</pre> Se cargan los niveles de alerta <pre>franklin@franklin-Nitro: ~ franklin@franklin-Nitro: ~/ros2_ws franklin@franklin-Nitro: ~\$ ros2 run yolobot_recognition yolov8_ros2_subscriber.py [INFO] [1747796515.571846435] [yolo_inference_subscriber]: Configuración de alertas cargada desde /home/franklin/ros2_ws/src/gui_final/gui_final/usr_config.yaml</pre>
Eliminar el usr_config.yaml cuando se carga la configuración por defecto.	
Objetivo	Validar que el archivo “usr_config.yaml” se puede eliminar al cargar la configuración por defecto y cada nodo usa su configuración por defecto.
Datos de la prueba	En esta prueba solo se necesita revisar que al presionar el boton de cargar configuración por defecto el archivo “usr_config.yaml” se elimine y los nodos funcionen con su configuración por defecto.
Procedimiento	1. Presionar el botón “cargar configuración por defecto” 2. Revisar que el archivo fue borrado exitosamente. 3. Revisar que los nodos funcionan con la configuración por defecto.
Resultado esperado	1. Se elimina el archivo “usr_config.yaml”. 2. Se revisa internamente que los nodos funcionen con su configuración por defecto.

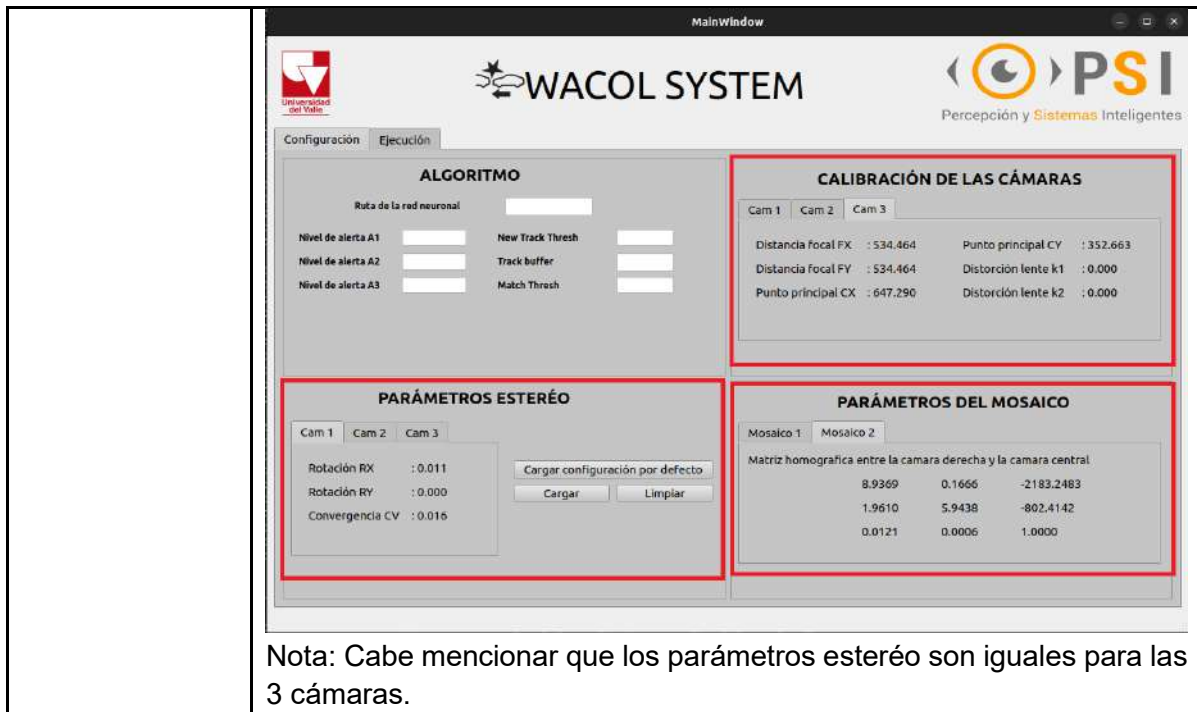
Resultados obtenidos	Se elimina el archivo “usr_config.yaml”  Se revisa que los nodos usan la configuración por defecto. 
Visualizar los parámetros extrínsecos e intrínsecos de las cámaras, las homografías y los parámetros estereo.	
Objetivo	Validar que en la pestaña configuración se pueden observar los parámetros extrínsecos e intrínsecos de las cámaras, las homografías y los parámetros estereo, despues de iniciar la ejecución del aplicativo.
Datos de la prueba	En esta prueba solo se inicializará el aplicativo para funcionar con un rosbag, de modo que se pueda observar todos los parámetros mencionados anteriormente.
Resultado esperado	1. Visualizar los parámetros extrínsecos e intrínsecos de las cámaras, las homografías y los parámetros estereo.
Resultados obtenidos	Se pueden visualizar los datos de calibración de cada cámara y los

parámetros del mosaico.

The image displays two screenshots of the WACOL SYSTEM software interface, showing the configuration and execution tabs. The interface is divided into four main panels, each with a red border. The top screenshot shows the 'Configuración' (Configuration) tab, and the bottom screenshot shows the 'Ejecución' (Execution) tab. The panels are:

- ALGORITMO**: Contains a 'Ruta de la red neuronal' (Neural network route) dropdown and three alert levels (A1, A2, A3) with corresponding 'New Track Thresh', 'Track buffer', and 'Match Thresh' values.
- CALIBRACIÓN DE LAS CÁMARAS**: Contains a table of camera calibration parameters for three cameras (Cam 1, Cam 2, Cam 3). The parameters include focal distance (FX, FY), principal point (CY, CX), and lens distortion (k1, k2).
- PARÁMETROS ESTEREO**: Contains a table of stereo parameters for three cameras (Cam 1, Cam 2, Cam 3). The parameters include rotation (RX, RY) and convergence (CV).
- PARÁMETROS DEL MOSAICO**: Contains a table of mosaic parameters for two mosaics (Mosaico 1, Mosaico 2). The parameters include homography matrix elements and lens distortion (k1, k2).

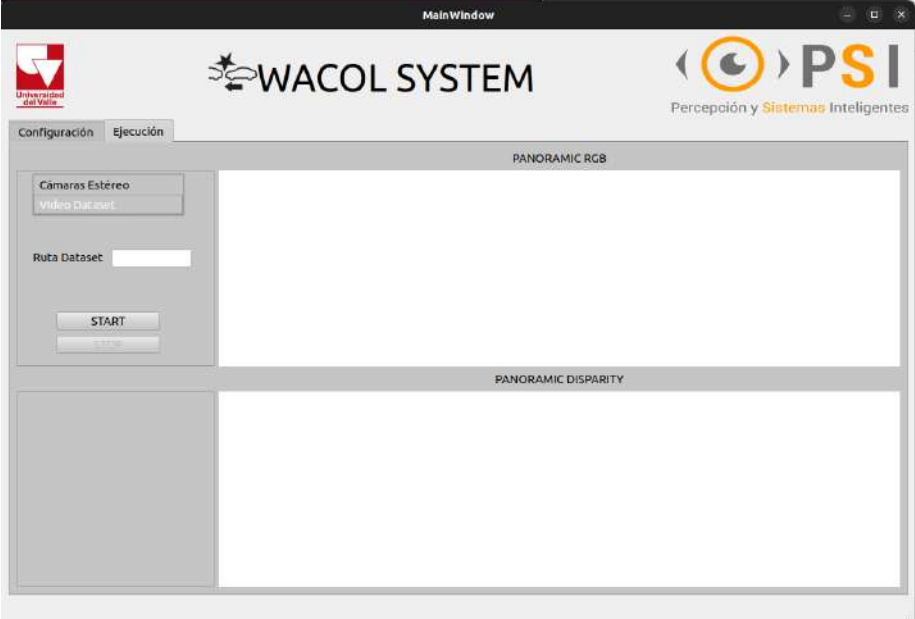
The 'CALIBRACIÓN DE LAS CÁMARAS' and 'PARÁMETROS DEL MOSAICO' panels are highlighted with red rectangles in both screenshots.

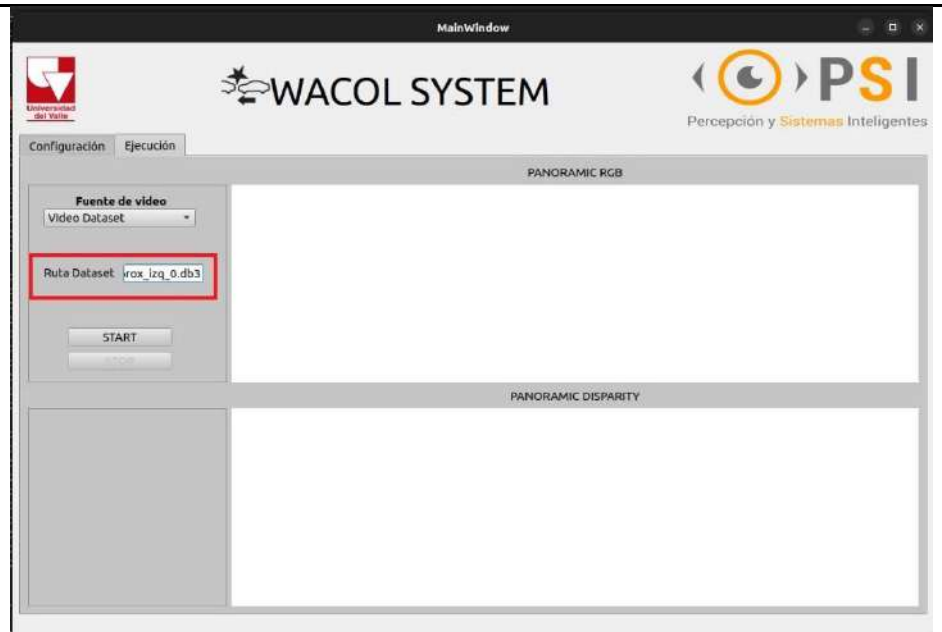


Nota: Cabe mencionar que los parámetros estereo son iguales para las 3 cámaras.

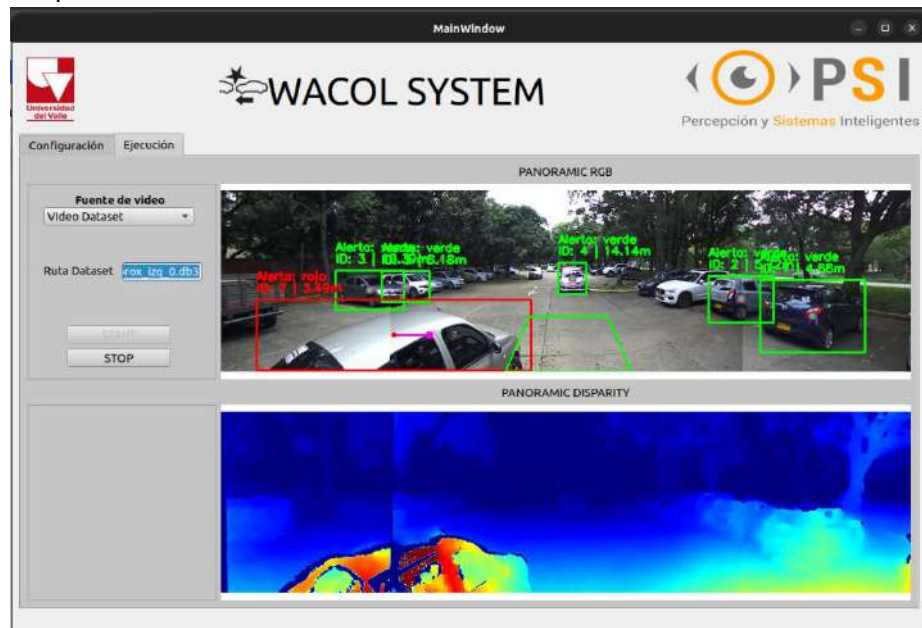
Tabla 4-2. Pruebas de integración Modulo de Ejecución.

PRUEBA DE INTEGRACIÓN – MÓDULO DE EJECUCIÓN	
• Seleccionar una fuente de información (cámaras o bagfile). • Iniciar y detener el proceso. • Visualizar la panorámica RGB resultante con los datos procesados y la panorámica de disparidad resultante.	
Tipo de prueba	Integración
Hardware Requerido	Portátil: • Procesador (CPU): Intel core i5 10300 H. • Tarjeta gráfica (GPU): Nvidia GTX 1650 Mobile. • Memoria RAM: 16 GB. • Memoria RAM de la GPU: 4 GB. 3 cámaras ZED2
Software requerido	Aplicativo de alerta de colisión con otros vehículos, (Python 3.10.12), ZED SDK, Ubuntu 22.04.5 LTS, Qt designer 5, CUDA, NVCC.
Objetivo	Validar el correcto funcionamiento del módulo de ejecución.
Seleccionar una fuente de información (cámaras o bagfile)	

Objetivo	Validar el correcto funcionamiento del aplicativo para seleccionar un video del tipo bagfile como fuente de información.
Datos de la prueba	Esta prueba se realiza seleccionando la opción de “Video Dataset” o en el comboBox que se encuentra en la pestaña “Ejecución” dentro del módulo de ejecución. Posteriormente ingresar el path al qlabel donde se indica la ruta del data set. Bagfile path: home/franklin/data/vehiculo_stop_1_aprox_izq/vehiculo_stop_1_aprox_izq_0.db3
Procedimiento	1. Seleccionar el comboBox de “Video Dataset” 2. Pegar el path al archivo bagfile que se quiere usar en el qlabel. 3. Presionar el boton “START”
Resultado esperado	1. Se carga el archivo. 2. Se visualiza el video
Resultados obtenidos	Se selecciona “Video Dataset” como fuente de información para la prueba.  Se ingresa el path del archivo bagfile para usarlo.



Se presiona el botón “Start”



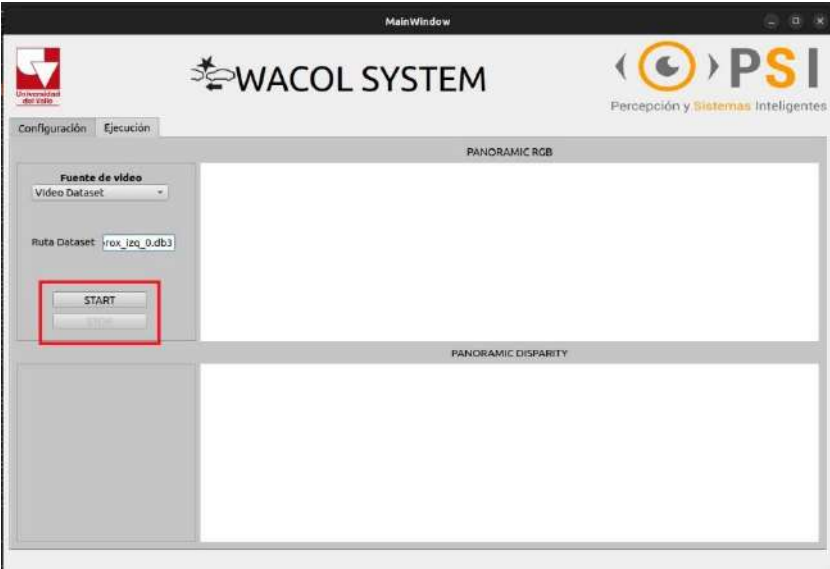
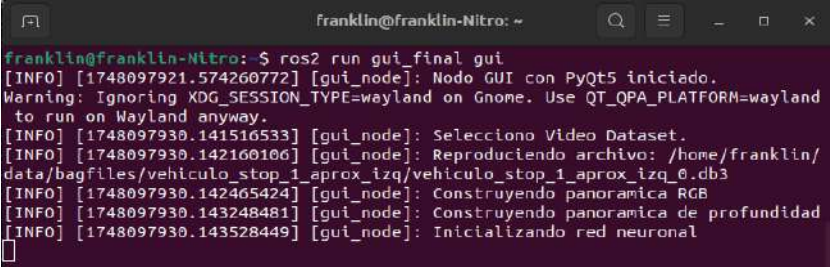
Iniciar y detener el proceso.

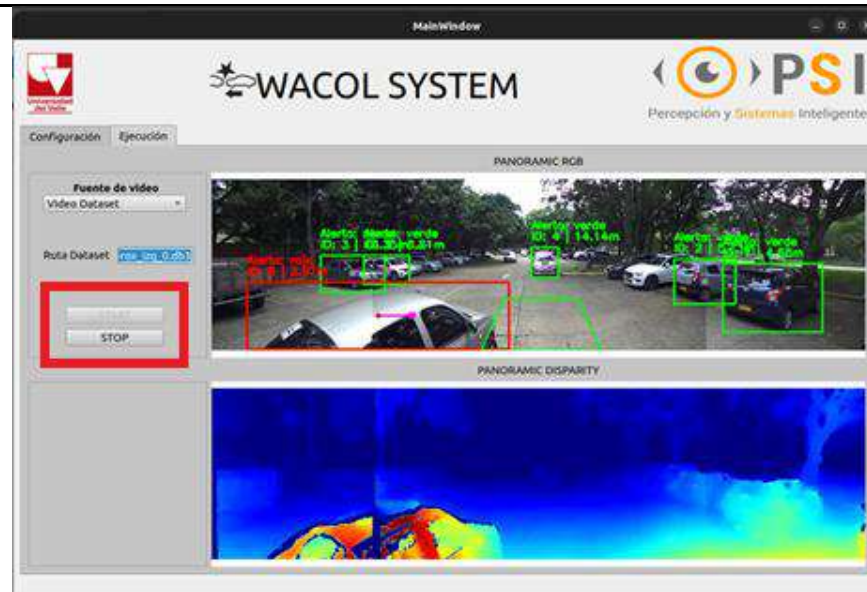
Objetivo

Validar el correcto funcionamiento del aplicativo para iniciar o detener el sistema.

Datos de la prueba

Esta prueba se realiza ya con la fuente de información seleccionada y presionando el botón de Start, antes de iniciar el sistema solo tiene habilitado el botón de Start, luego de presionarlo se inhabilita y se habilita el botón Stop para detener el proceso.

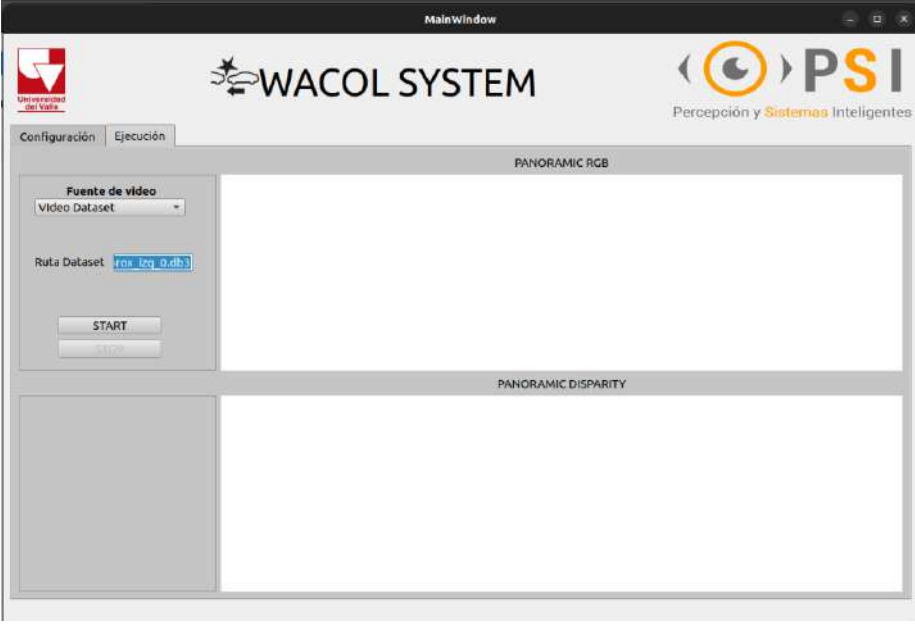
Procedimiento	1. Seleccionar la fuente de información. 2. Presionar el botón “START” para empezar. 3. Presionar el botón “STOP” para detener.
Resultado esperado	1. Se carga el archivo. 2. Se visualiza el video presionando el botón de Start. 3. Se deja de visualizar el video presionando el botón Stop.
Resultados obtenidos	Se presiona el botón “Start”  Se visualiza en la terminal que el proceso inicio.  Se presiona el boton “STOP”.



Se visualiza en la terminal que el proceso se detiene.

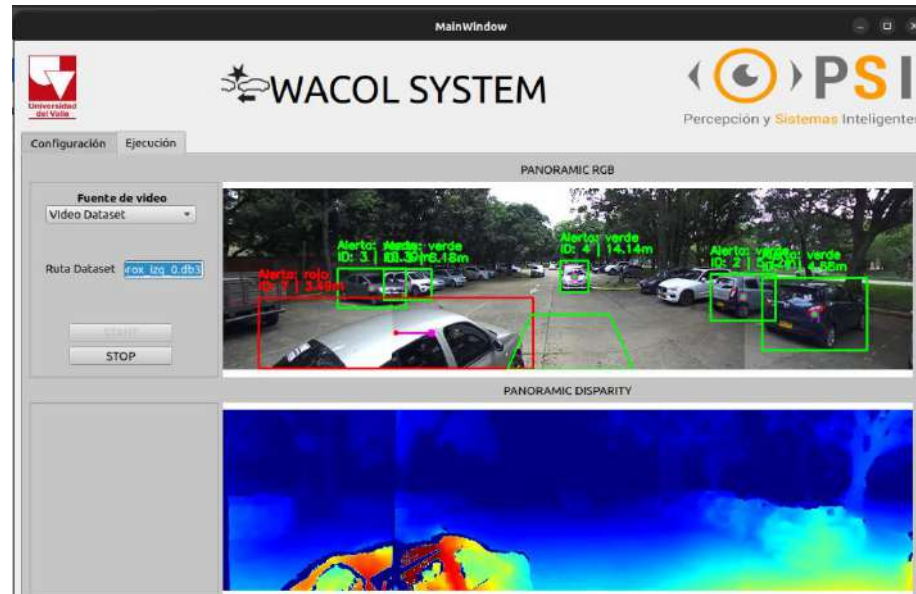
```
franklin@franklin-Nitro: ~
franklin@franklin-Nitro:~$ ros2 run gui_final gui
[INFO] [1748097921.574260772] [gui_node]: Nodo GUI con PyQt5 iniciado.
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland
to run on Wayland anyway.
[INFO] [1748097930.141516533] [gui_node]: Selecciono Video Dataset.
[INFO] [1748097930.142160106] [gui_node]: Reproduciendo archivo: /home/franklin/
data/bagfiles/vehiculo_stop_1_aprox_izq/vehiculo_stop_1_aprox_izq_0.db3
[INFO] [1748097930.142465424] [gui_node]: Construyendo panoramica RGB
[INFO] [1748097930.143248481] [gui_node]: Construyendo panoramica de profundidad
[INFO] [1748097930.143528449] [gui_node]: Inicializando red neuronal
[INFO] [1748097984.953738680] [gui_node]: Deteniendo reproducción del archivo ro
sbag.
[INFO] [1748097984.955400308] [gui_node]: Deteniendo todos los nodos.
[INFO] [1748097984.955936694] [gui_node]: -----
-----
[ ]
```

Se verifica que ya no se muestra nada en la GUI.

	
Visualizar la panorámica RGB resultante con los datos procesados y la panorámica de disparidad resultante.	
Objetivo	Validar que en la pestaña de ejecución el aplicativo es capaz de reproducir el video de la panorámica RGB y de disparidad según su fuente de video.
Datos de la prueba	En esta prueba solo se inicializará el aplicativo para funcionar con un rosbag, de modo que se pueda que los qlabels de la panorámica RGB y de disparidad se pueden visualizar sin problema.
Resultado esperado	1. Visualizar la panorámica RGB resultante con los datos procesados y la panorámica de disparidad resultante.

Resultados obtenidos

Se puede observar que la GUI muestra las respectivas panorámicas.



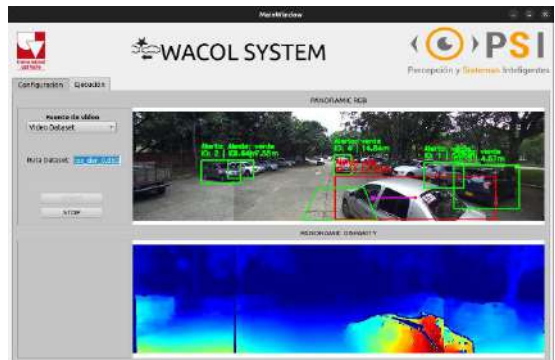
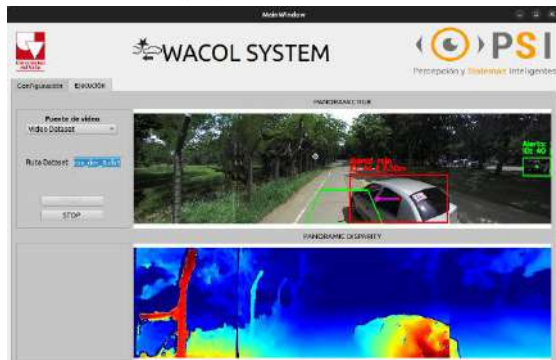
4.4 PRUEBAS DE CAMPO

Para el proceso de validación del sistema de alerta de colisión con otros vehículos, se realizaron dos pruebas de campo en diferentes escenarios representativos. Las pruebas se realizaron dentro de la Universidad del Valle con el carro eléctrico experimental de la Escuela de Ingeniería Mecánica (vehículo propio) que tenía el sistema montado en la parte superior y un Renault Symbol (vehículo externo). En la figura 4-1 se puede observar a los vehículos en cuestión. Las pruebas se dividen en dos grandes grupos, el vehículo propio quieto y en movimiento. En la tabla 4-3 se muestran los diferentes casos de prueba.



Figura 4-1. Vehículos usados para las pruebas de campo.

Tabla 4-3. Pruebas realizadas.

VEHÍCULO PROPIO QUIETO	VEHÍCULO PROPIO EN MOVIMIENTO
Invasión de carril por el lado izquierdo (5 pruebas)  Vehículo externo Vehículo propio 	Invasión de carril por el lado izquierdo (5 pruebas)  Vehículo externo Vehículo propio 
Invasión de carril por el lado derecho (5 pruebas)  Vehículo propio Vehículo externo 	Invasión de carril por el lado derecho (5 pruebas)  Vehículo propio Vehículo externo 

Aproximación frontal (10 pruebas)

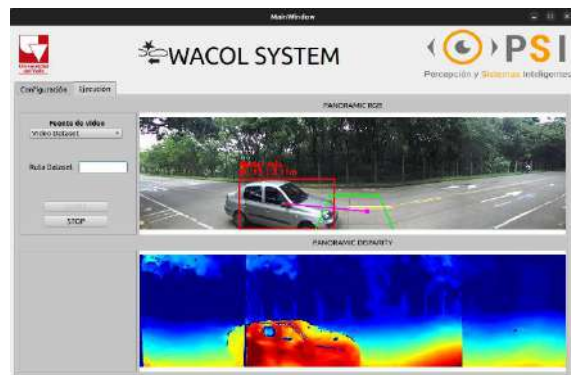


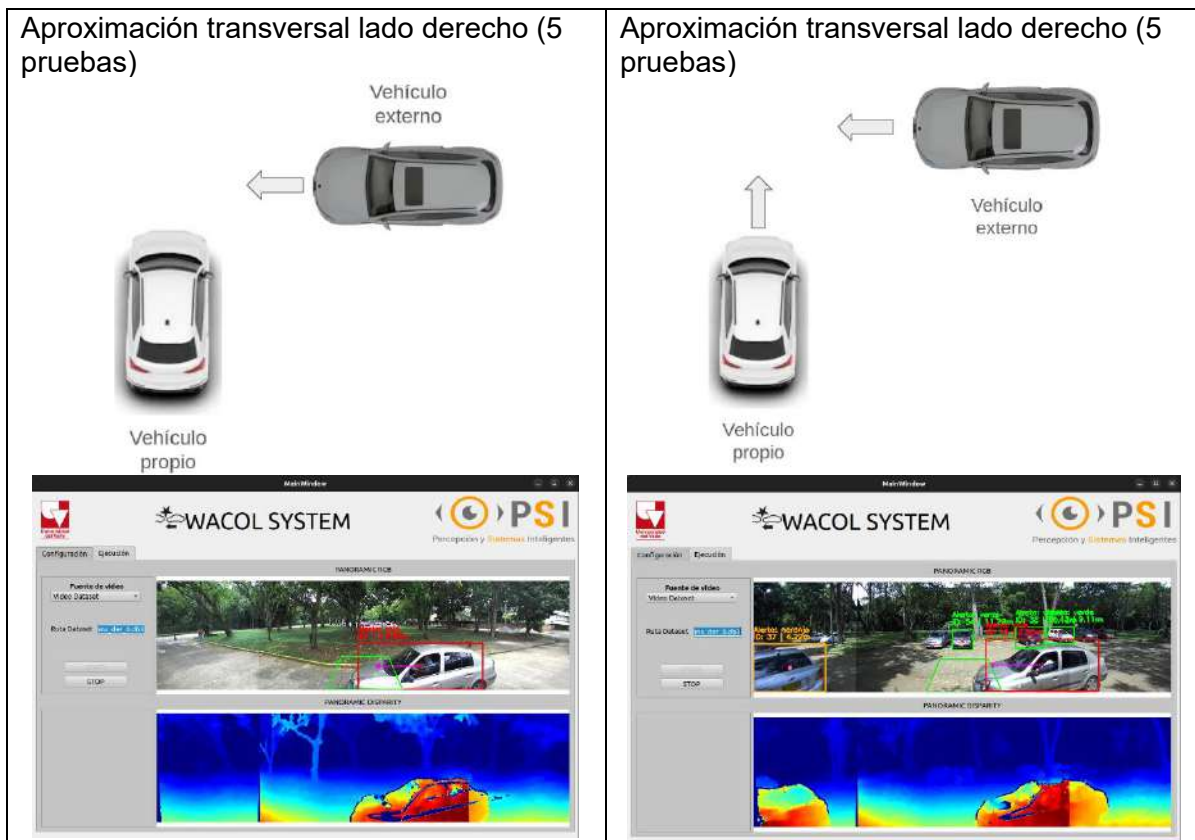
No se realizó este tipo de prueba en movimiento, ya que tiene un cierto grado de peligro cuando se están moviendo dos vehículos al tiempo uno detrás del otro.

Aproximación transversal lado izquierdo (5 pruebas)



Aproximación transversal lado izquierdo (5 pruebas)





Las pruebas se realizaron usando la red neuronal YOLOv8m entrenada con el data set 2 mencionada en el capítulo **3.3.2.3 Implementación de la red neuronal** pues la figura 3-21 muestra unos buenos resultados al evaluarla. Así mismo las pruebas se realizaron de día entre las 11 a.m. y las 2 p.m. (11 a.m. - 12 m. y 1 p.m.- 2 p.m.) teniendo una duración total de 2 horas, durante esas dos horas se grabaron 30 bagfiles para el vehículo quieto y 20 bagfiles para el vehículo en movimiento, dando un total de 50 bagfiles.

Para el escenario con el vehículo en reposo se registró una duración total de 355,39 segundos, mientras que para el vehículo en movimiento la duración fue de 313 segundos. A partir de cada archivo de registro (*bagfile*), se extrajeron 100 fotogramas distribuidos de manera uniforme a lo largo de su ejecución, obteniendo así un total de 5000 imágenes para el proceso de análisis. Dicho análisis se realizó considerando los parámetros presentados en la Tabla 4-4, con los cuales fue posible construir la matriz de confusión correspondiente.

Tabla 4-4. Parámetros para obtener la matriz de confusión.

Nombre	¿Qué significa?
Verdadero Positivo (TP)	Había realmente riesgo de colisión y el sistema activó la alerta.
Verdadero Negativo (TN)	No había riesgo de colisión y el sistema no activó la alerta.
Falso Positivo (FP)	No había riesgo de colisión, pero el sistema activó una alerta (falsa alarma).

Falso Negativo (FN)	Sí había riesgo de colisión, pero el sistema no activó alerta. Esto es lo más peligroso.
TPR (True Positive Rate) o Sensibilidad/Recall	De todos los positivos reales , ¿qué proporción identificó correctamente el modelo?
TNR (True Negative Rate) o Especificidad	De todos los negativos reales , ¿qué proporción identificó correctamente el modelo?
FPR (False Positive Rate)	De todos los negativos reales , ¿qué proporción fue incorrectamente clasificada como positiva?
FNR (False Negative Rate)	De todos los positivos reales , ¿qué proporción fue incorrectamente clasificada como negativa?

4.4.1 VEHÍCULO QUIETO

El sistema de alerta de colisión se evaluó inicialmente con el vehículo en estado estacionario. Este escenario resulta relevante porque, aunque el vehículo propio apenas se mueva, pueden presentarse colisiones por descuido del conductor o de otros vehículos, como ocurre en trancones, donde son frecuentes los choques frontales o laterales a baja velocidad. Si bien este tipo de accidentes no suelen implicar un alto riesgo para la seguridad de los ocupantes, sí generan daños materiales y pérdidas económicas.

Para estas pruebas se realizaron 30 ejecuciones (30 *bagfiles*), de las cuales se extrajeron 100 fotogramas cada una, distribuidos uniformemente a lo largo de la duración del archivo. En total se obtuvieron 3000 imágenes, de las cuales 2996 fueron útiles para el análisis, ya que 4 no contenían información relevante. Los resultados se presentan en la **Tabla 4-5**, donde los valores fueron ajustados ligeramente para que la suma total corresponda al 100%.

Tabla 4-5. Resultados generación de alerta para el vehículo quieto.

GENERACIÓN DE ALERTA VEHÍCULO QUIETO	
Verdadero Positivo (VP): 2084 (69.6%)	Falso Positivo (FP): 0
Falso Negativo (FN): 289 (9.6%)	Verdadero Negativo (VN): 623 (20.8%)

- **Verdaderos positivos:** Este valor indica que en 2084 de las 2996 muestras, el sistema generó correctamente la alerta esto indica que cumple adecuadamente con la generación de alertas cuando hay peligro.
- **Falsos negativos:** Este valor crítico indica que en 289 de las 2996 muestras, el sistema no generó alertas aun cuando existía un riesgo de colisión. Esto corresponde al 9.65% indicando que el sistema es bastante confiable, pues la no generación de alertas ante un escenario de riesgos es baja. La causa principal de estos falsos negativos radica en que la red neuronal **YOLO**, en ciertos fotogramas, tuvo dificultades para detectar correctamente al vehículo. Este problema se presentó, principalmente, cuando el vehículo transitaba por la **zona de empalme de las imágenes en la panorámica**, donde la detección se ve afectada por la fusión de información visual. Como resultado, el sistema no emitía la alerta correspondiente, al no identificar adecuadamente al objeto en riesgo.
- **Falsos positivos:** Este valor crítico indica que en ninguna de las 2996 muestras, el sistema generó alertas que en realidad no correspondían a ningún riesgo. Esto es un buen indicador de la precisión del sistema para generar alertas.

- **Verdaderos negativos:** Este valor indica que en 623 de las 2996 muestras, el sistema reconoció que no había riesgo y no generó ninguna alerta de colisión, ya que realmente no había riesgo.

Al calcular la precisión, la exactitud (accuracy), TPR, TNR, FPR y FNR se obtiene los siguientes resultados:

$$\text{Precisión} = \frac{VP}{VP+FP} = \frac{2084}{2084+0} = 1 \text{ es decir, un } 100\%$$

$$\text{Accuracy} = \frac{VP+VN}{VP+VN+FP+FN} = \frac{2084+623}{2084+623+0+289} = 0.9035 \text{ es decir, un } 90.35\%$$

$$\text{TPR} = \frac{VP}{VP+FN} = \frac{2084}{2084+289} = 0,8782 \text{ es decir, un } 87.82\%$$

$$\text{TNR} = \frac{VN}{VN+FP} = \frac{623}{623+0} = 1 \text{ es decir, un } 100\%$$

$$\text{FPR} = \frac{FP}{VN+FP} = \frac{0}{623+0} = 0 \text{ es decir, un } 0\%$$

$$\text{FNR} = \frac{FN}{VP+FN} = \frac{289}{2084+289} = 0,1217 \text{ es decir, un } 12.17\%$$

Sensibilidad (TPR = 87.82%) El sistema logra identificar correctamente la mayoría de los casos donde realmente existe un riesgo de colisión. Esto significa que en casi **9 de cada 10 situaciones de peligro**, se genera la alerta correspondiente. Es un resultado sólido, aunque el **12.17% de falsos negativos** (casos de riesgo no detectados) debe considerarse como una limitación crítica, dado que en un sistema de seguridad vehicular la omisión de una alerta puede tener consecuencias graves.

Especificidad (TNR = 100%) El modelo clasifica perfectamente los casos en los que **no existe riesgo**, lo que se confirma con el **FPR = 0%**. En otras palabras, **no se generan falsas alarmas**. Este resultado es muy positivo, ya que para el aplicativo es crucial evitar que el sistema genere advertencias innecesarias, las cuales podrían inducir al conductor a ignorar alertas reales.

Falsos negativos (FNR = 12.17%) Aunque la proporción de falsos negativos es relativamente baja, representa el punto más débil del sistema. Los **289 casos sin alerta pese a haber riesgo** se explican principalmente por limitaciones en la detección de vehículos por parte de la red YOLO, en especial en la zona de empalme de las imágenes panorámicas. Esto sugiere que la arquitectura de fusión de imágenes podría optimizarse para mejorar la detección en esas áreas críticas.

Falsos positivos (FPR = 0%) La ausencia de falsos positivos es un punto fuerte del sistema. En ninguna de las 2996 muestras se generó una alerta sin motivo, lo que significa que la **confiabilidad del sistema frente a falsas alarmas es total**.

4.1.2 VEHÍCULO EN MOVIMIENTO

El sistema de alerta de colisión también se evaluó con el vehículo en movimiento, lo que permite analizar escenarios más cercanos a la conducción real. En estas condiciones, pueden ocurrir colisiones por descuido del conductor o de otros vehículos, como sucede en maniobras de cambio de carril o en intersecciones. Estas pruebas se realizaron cuidadosamente, simulando posibles situaciones de riesgo de forma controlada.

Para este escenario se efectuaron 20 pruebas, descritas en la **Tabla 4-3**. De ellas se obtuvo un total de 2000 imágenes, de las cuales se analizaron 1586, ya que 414 no contenían

información útil. Los resultados se presentan en la **Tabla 4-6**, donde los valores fueron ajustados ligeramente para garantizar que la suma total corresponda al 100%.

Tabla 4-6. Resultados generación de alerta para el vehículo en movimiento.

GENERACIÓN DE ALERTA VEHÍCULO EN MOVIMIENTO	
Verdadero Positivo (VP): 610 (38.5%)	Falso Positivo (FP): 82 (5.1%)
Falso Negativo (FN): 184 (11.6%)	Verdadero Negativo (VN): 710 (44.8%)

- **Verdaderos positivos:** Este valor indica que, en 610 de las 1586 muestras, el sistema generó correctamente la alerta esto indica que cumple adecuadamente con la generación de alertas cuando hay peligro.
- **Falsos negativos:** Este valor indica crítico que, en 184 de las 1586 muestras, el sistema no generó alertas aun cuando existía un riesgo de colisión. Esto corresponde al 11.78% indicando que el sistema es bastante confiable, pero tiene margen de mejora. Al igual que en el caso anterior la causa principal de estos falsos negativos radica en que la red neuronal **YOLO**, en ciertos fotogramas, tuvo dificultades para detectar correctamente al vehículo. Este problema se presentó, principalmente, cuando el vehículo transitaba por la **zona de empalme de las imágenes en la panorámica**, más aún para este caso ya que el vehículo está en movimiento. Como resultado, el sistema no emitía la alerta correspondiente, al no identificar adecuadamente al objeto en riesgo.
- **Falsos positivos:** Este valor crítico indica que, en 80 de las 2996 muestras, el sistema generó alertas que en realidad no correspondían a ningún riesgo. Esto es un indicador de que movimiento puede generar un poco de ruido en sus predicciones para la generación de alertas. Esto puede deberse al **movimiento del vehículo propio y de los objetos del entorno** generando ruido tanto en las detecciones como en las distancias obtenidas. Este ruido puede provocar variaciones momentáneas en las posiciones y trayectorias estimadas de los objetos, llevando al sistema a interpretar erróneamente que existe una situación de riesgo. Además, este efecto se intensifica cuando hay cambios rápidos en la perspectiva de la cámara o en la iluminación, lo que puede afectar la precisión de la red neuronal y del algoritmo de seguimiento.
- **Verdaderos negativos:** Este valor indica que, en 710 de las 1586 muestras, el sistema reconoció que no había riesgo y no generó ninguna alerta de colisión, ya que realmente no había riesgo.

Al calcular la precisión, la exactitud (accuracy), TPR, TNR, FPR y FNR se obtienen los siguientes resultados:

$$\text{Precisión} = \frac{VP}{VP+FP} = \frac{610}{610+82} = 0.8815 \text{ es decir, un } 88.15\%$$

$$\text{Accuracy} = \frac{VP+VN}{VP+VN+FP+FN} = \frac{610+710}{610+710+82+184} = 0.8322 \text{ es decir, un } 83.22\%$$

$$\text{TPR} = \frac{VP}{VP+FN} = \frac{610}{610+184} = 0.7682 \text{ es decir, un } 76.82\%$$

$$\text{TNR} = \frac{VN}{VN+FP} = \frac{710}{710+82} = 0.8964 \text{ es decir, un } 89.64\%$$

$$FPR = \frac{FP}{VN+FP} = \frac{82}{710+82} = 0.1035 \text{ es decir, un } 10.35\%$$

$$FNR = \frac{FN}{VP+FN} = \frac{184}{610+184} = 0,2317 \text{ es decir, un } 23.17\%$$

Sensibilidad (TPR = 76.82%) El sistema identifica correctamente aproximadamente **3 de cada 4 situaciones de riesgo**, lo cual es aceptable, pero **inferior al caso anterior (87.82% vehículo estático)**. La disminución del TPR refleja una mayor dificultad para detectar todos los casos de peligro, probablemente por la complejidad añadida del movimiento (del vehículo propio y de los objetos circundantes).

Especificidad (TNR = 89.64%) La tasa de verdaderos negativos se mantiene relativamente alta: en casi 9 de cada 10 escenarios sin riesgo, el sistema no genera falsas alarmas. Sin embargo, aparece un **FPR = 10.35%**, esto señala que, en alrededor de 1 de cada 10 casos sin riesgo, el sistema sí emitió una alerta innecesaria. Esto introduce un problema de **falsas alarmas** que puede afectar la confianza del usuario en el sistema si se repite con frecuencia.

Falsos negativos (FNR = 23.17%) El **23% de los riesgos reales no fueron detectados**, lo que es un valor preocupante. Como se describe, esto se debe principalmente a limitaciones en la detección con YOLO en zonas de empalme y a la mayor complejidad del escenario dinámico. El movimiento introduce problemas de consistencia en la detección y en el cálculo de trayectorias, lo que incrementa la probabilidad de que el sistema ignore amenazas reales.

Falsos positivos (FPR = 10.35%) En este caso, a diferencia del anterior, sí se generan falsas alarmas. Se identificaron **82 casos de riesgo inexistente señalados como colisión**. Este ruido puede atribuirse al movimiento del vehículo propio y de los objetos del entorno, que afecta tanto las detecciones de YOLO como las estimaciones de distancia y trayectoria. Factores adicionales, como cambios rápidos en la perspectiva de la cámara y variaciones de iluminación, intensifican estos errores.

4.5 CONCLUSIONES

Las pruebas realizadas con el sistema de alerta de colisión utilizando la red neuronal YOLOv8m entrenada con el dataset 2 mencionado en el capítulo **3.3.2.2 Implementación de la red neuronal**, demostraron un desempeño sólido tanto en condiciones de vehículo quieto como en movimiento. Se analizaron un total de 50 bagfiles (30 con el vehículo quieto y 20 con el vehículo en movimiento), extrayendo 100 fotogramas por bagfile, para un total de 5000 imágenes evaluadas.

En el caso del **vehículo quieto**, el sistema mostró una **precisión del 100%** y una **exactitud del 90.35%**, sin generar falsos positivos y con una tasa de falsos negativos del 12.17%. Esto refleja una alta confiabilidad en la detección de riesgos reales sin producir falsas alarmas, lo cual es ideal en escenarios de bajo dinamismo como trancones o maniobras de parqueo, donde los accidentes pueden ser frecuentes, pero no necesariamente graves.

Para el **vehículo en movimiento**, el sistema mantuvo una **precisión del 88.4%** y una **exactitud del 83.22%**, con una tasa de falsos negativos del 23.17% y falsos positivos del 10.35%. Aunque estos valores siguen indicando un buen desempeño general, también evidencian que el movimiento introduce desafíos adicionales que afectan levemente la fiabilidad del sistema. Este comportamiento sugiere que la dinámica del entorno genera mayor ruido en las predicciones y en las mediciones de distancia, lo que puede llevar a alertas erróneas o la omisión de algunas situaciones de riesgo.

En conjunto, los resultados validan el funcionamiento del sistema de alerta de colisión bajo distintos escenarios, demostrando su capacidad para detectar de manera efectiva situaciones potencialmente peligrosas. Sin embargo, se identifican oportunidades de mejora en escenarios dinámicos, donde una optimización del modelo o un ajuste de los umbrales de alerta podría incrementar la robustez del sistema.

5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1 Conclusiones generales

La siniestralidad vial continúa siendo uno de los problemas más persistentes y alarmantes en la agenda pública. Según datos de la Organización Mundial de la Salud, cada año fallecen aproximadamente 1.3 millones de personas en accidentes de tráfico [2]. Ante esta realidad, el presente trabajo abordó la problemática desde una perspectiva tecnológica, con el objetivo de desarrollar una solución orientada a la prevención de accidentes mediante el uso de sistemas inteligentes, partiendo de la combinación de diferentes campos, como lo es la visión por computadora, las redes neuronales convolucionales y el hardware adecuado para realizar una función en específico.

Partiendo de la comprensión de la magnitud del problema, se llevó a cabo una exhaustiva revisión bibliográfica que permitió analizar enfoques, tecnologías y modelos previos enfocados en la detección y prevención de colisiones. Este análisis, desarrollado en el capítulo 2, condujo a la identificación de tres pilares fundamentales para el diseño de sistemas de alerta de colisión efectivos: en primer lugar, la calidad de los sensores, que fue determinante para obtener una percepción precisa del entorno; en segundo lugar, los métodos de detección y seguimiento de objetos; y, por último, la necesidad de contar con un sistema embebido de alto rendimiento, capaz de procesar datos y emitir alertas de manera ágil, con una latencia mínima, requisito importante para reaccionar oportunamente ante situaciones de riesgo.

Este planteamiento se consolidó en el marco teórico, donde se analizó en profundidad el funcionamiento de técnicas como la detección basada en redes neuronales convolucionales, el seguimiento de objetos y la estimación de colisiones. Esta comprensión permitió establecer un diseño robusto que integraba de manera coherente diversas disciplinas, como la visión por computador, el aprendizaje profundo y la predicción de trayectorias, lo que dio lugar a una arquitectura tecnológica orientada a mejorar la seguridad vial.

A partir de esta base conceptual, el capítulo tres detalló el proceso de desarrollo técnico del sistema, desde la fase de diseño conceptual y la elaboración de diagramas estructurales hasta la construcción de la interfaz gráfica. El aplicativo final se organizó en dos módulos principales: el módulo de configuración y el módulo de procesamiento. El primero permitió al usuario ajustar parámetros clave del sistema, como los niveles de alerta, la red neuronal empleada, los umbrales del algoritmo de seguimiento y la visualización de datos de calibración, tanto del mosaico panorámico como del sistema estéreo.

Por su parte, el módulo de procesamiento se encargó de ejecutar las funciones esenciales del sistema: detección de vehículos, seguimiento, estimación de distancias y generación de alertas. La integración de tecnologías como el algoritmo ByteTrack para el seguimiento de objetos y los filtros de Kalman para la predicción de trayectorias dotó al sistema de una capacidad inteligente para anticipar riesgos y emitir alertas oportunas, brindando al conductor tiempo suficiente para reaccionar ante posibles situaciones de peligro.

La validación práctica, documentada en el cuarto capítulo, demostró la eficacia del sistema bajo condiciones reales, tanto con el vehículo estático como en movimiento. Los resultados obtenidos fueron satisfactorios, alcanzando niveles de precisión y exactitud que avalaron su funcionalidad. Las pruebas realizadas abordaron diferentes escenarios, tanto para el

vehículo quieto como para el vehículo en movimiento, simulando colisiones frontales, transversales o de invasión de carril. La tabla 4-3 detalló las pruebas efectuadas al sistema. Para el caso del vehículo quieto, el sistema mostró una **precisión del 100%** y una **exactitud del 90.35%**, sin generar falsos positivos y con una tasa de falsos negativos del 12.17%. Para el vehículo en movimiento, el sistema mantuvo una **precisión del 88.4%** y una **exactitud del 83.22%**, con una tasa de falsos negativos del 23.17% y falsos positivos del 10.35%. De forma general se podría decir que el sistema tuvo una precisión del 94,2% y una exactitud de 86.78%.

Sin embargo, también se identificaron áreas de mejora, especialmente en escenarios dinámicos, donde el entorno cambiante y las condiciones de iluminación o movimiento introdujeron incertidumbre en las predicciones y en los valores de distancia. Esto abrió la puerta a futuras líneas de investigación centradas en la optimización de modelos, la fusión de sensores y la adaptación del sistema a contextos aún más complejos, dado que el sistema se probó en un entorno controlado dentro de la universidad con pocos vehículos en escena y un solo vehículo en movimiento.

En suma, este trabajo entregó una posible solución al problema de la siniestralidad vial. La combinación entre rigor científico, innovación tecnológica y compromiso social fue el motor de este proyecto, y sentó las bases para desarrollos futuros que integraran más funcionalidades, como el reconocimiento de peatones, ciclistas u obstáculos imprevistos.

De esta manera, se cerró un ciclo de investigación que abarcó desde la formulación del problema hasta la implementación y evaluación de una solución concreta. Se demostró que fue posible diseñar, construir y validar un sistema ADAS funcional y accesible, con potencial real para salvar vidas. El desafío que quedó fue continuar su evolución para optimizarlo y adaptarlo a nuevas plataformas y tecnologías, ya que cada vida que se salvó en la vía gracias a la tecnología representó un paso más hacia un futuro más seguro y justo para todos.

5.2 Trabajos futuros

A partir de los resultados obtenidos y las limitaciones observadas en el presente trabajo, se identifican múltiples oportunidades de mejora y expansión que pueden guiar el desarrollo de futuras investigaciones o implementaciones del sistema propuesto.

Una primera línea de trabajo recomendable consiste en **la incorporación de módulos de preprocesamiento distribuidos**, capaces de realizar tareas como compresión, normalización de iluminación y envío sincronizado de las imágenes de cada cámara. Estos módulos, integrados físicamente cerca de cada cámara, permitirían **reducir la carga computacional del nodo central** y garantizar una entrega más eficiente y sincronizada de los datos visuales, algo especialmente útil en arquitecturas multivista o multicámara.

En este mismo sentido, se plantea como desafío técnico la **limitación de ancho de banda en plataformas embebidas**, particularmente en el caso de la **NVIDIA Jetson AGX Orin**, donde la simultaneidad de flujos de video a alta resolución puede provocar cuellos de botella. Como propuesta de mejora, se sugiere explorar la **integración de dispositivos complementarios de expansión de ancho de banda**, como módulos de transferencia por PCIe externos, adaptadores de red de alto rendimiento o buses de comunicación

personalizados, que permitan aumentar el rendimiento del sistema sin comprometer su latencia ni su capacidad de respuesta en tiempo real.

Asimismo, como trabajo futuro se sugiere implementar una alarma visual que pueda ser percibida directamente por el conductor sin necesidad de desviar la atención hacia la pantalla, ya que dicha acción puede representar un riesgo durante el proceso de conducción. Esta funcionalidad podría integrarse mediante indicadores luminosos en el tablero o en el parabrisas (HUD), contribuyendo a una interacción más segura y efectiva con el sistema.

Finalmente, se considera promisorio investigar la **escalabilidad del sistema hacia entornos distribuidos o en red**, como flotas de vehículos conectados o sistemas de transporte público, donde la información recolectada por un nodo pueda alimentar modelos colaborativos o sistemas centralizados de alerta urbana.

Estas líneas de trabajo no solo fortalecerían la viabilidad técnica del sistema desarrollado, sino que permitirían proyectarlo hacia aplicaciones reales con impacto social, económico y tecnológico en el campo de la seguridad vial.

6. REFERENCIAS

- [1] NHTSA (National Highway Traffic Safety Administration), “Vehículos Automatizados para Mayor Seguridad.” Accessed: May 21, 2023. [Online]. Available: <https://www.nhtsa.gov/es/tecnologia-e-innovacion/vehiculos-automatizados-para-mayor-seguridad>
- [2] H. S. Lee and K. Kim, “Simultaneous Traffic Sign Detection and Boundary Estimation Using Convolutional Neural Network,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 5, pp. 1652–1663, May 2018, doi: 10.1109/TITS.2018.2801560.
- [3] World Health Organization, “Traumatismos causados por el tránsito.” Accessed: Apr. 02, 2023. [Online]. Available: <https://www.who.int/es/news-room/fact-sheets/detail/road-traffic-injuries>
- [4] Diana Nova, “Boletín estadístico Colombia - Fallecidos y lesionados por siniestros viales,” Agencia Nacional de Seguridad Vial. Accessed: Apr. 04, 2023. [Online]. Available: https://ansv.gov.co/sites/default/files/2023-03/Boletin_Nacional_Diciembre_2022.pdf
- [5] Secretaria de tránsito y transporte municipal Santiago de Cali, “Accidentes de tránsito Cali 2012,” Estadística de accidentalidad, Cali, Colombia. Accessed: Apr. 09, 2023. [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjqpfaUgof_AhUzl2oFHWLyBVoQFnoECAgQAQ&url=https%3A%2F%2Fwww.cali.gov.co%2Floader.php%3FIServicio%3DTools%26ITipo%3Ddescargas%26IFuncion%3Ddescargar%26idFile%3D9598&usg=AOvVaw1ZjQsPChSn87rq_cqR8qCc
- [6] A. Simic, O. Kocic, M. Z. Bjelica, and M. Milosevic, “Driver monitoring algorithm for advanced driver assistance systems,” in *24th Telecommunications Forum, TELFOR*

- 2016, Institute of Electrical and Electronics Engineers Inc., Jan. 2017. doi: 10.1109/TELFOR.2016.7818908.
- [7] H. Noori, Dānishgāh-i Firdawsī. Department of Computer Engineering, and Institute of Electrical and Electronics Engineers, *Object Tracking Methods: A Review - 2019 9th International Conference on Computer and Knowledge Engineering (ICCKE) : October 24-25, 2019, Ferdowsi University of Mashhad*. 2019.
 - [8] S. K. Soni, Madan Mohan Malaviya University of Technology, North Dakota State University, Institute of Electrical and Electronics Engineers. Uttar Pradesh Section, IEEE Industry Applications Society, and Institute of Electrical and Electronics Engineers, *FCW: A Forward Collision Warning System Using Convolutional Neural Network - ICE3-2020 : International Conference on Electrical and Electronics Engineering : February 14-15, 2020*.
 - [9] C.-H. Hsieh, D.-C. Lin, C.-J. Wang, Z.-T. Chen, and J.-J. Liaw, "Real-Time Car Detection and Driving Safety Alarm System With Google Tensorflow Object Detection API; Real-Time Car Detection and Driving Safety Alarm System With Google Tensorflow Object Detection API," 2019.
 - [10] Forbes staff, "En Colombia, el 63,8% de las personas no ganan más de un salario mínimo," Forbes Colombia. Accessed: Apr. 12, 2023. [Online]. Available: <https://forbes.co/2020/12/07/economia-y-finanzas/en-colombia-el-638-de-las-personas-no-ganan-mas-de-un-minimo/>
 - [11] J. B. Kim, "Efficient vehicle detection and distance estimation based on aggregated channel features and inverse perspective mapping from a single camera," *Symmetry (Basel)*, vol. 11, no. 10, Oct. 2019, doi: 10.3390/sym11101205.
 - [12] A. M. Ibrahim, R. M. Hassan, A. E. Tawfiles, T. Ismail, and M. S. Darweesh, "Real-Time Collision Warning System Based on Computer Vision Using Mono Camera," in *2nd Novel Intelligent and Leading Emerging Sciences Conference, NILES 2020*, Institute of Electrical and Electronics Engineers Inc., Oct. 2020, pp. 60–64. doi: 10.1109/NILES50944.2020.9257941.
 - [13] Guo li Taiwan ke ji da xue and Institute of Electrical and Electronics Engineers, *Real-Time Forward Collision Alert System using Raspberry Pi - 2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS 2019) : proceedings : Dec. 3-6, 2019, Taipei, Taiwan*.
 - [14] J. Cao, Y. Wang, and B. Li, "Real-time vehicle back-up warning system with a single camera," in *Proceedings - International Conference on Image Processing, ICIP*, IEEE Computer Society, Dec. 2015, pp. 2275–2279. doi: 10.1109/ICIP.2015.7351207.
 - [15] W. Song, Y. Yang, M. Fu, Y. Li, and M. Wang, "Lane Detection and Classification for Forward Collision Warning System Based on Stereo Vision," *IEEE Sens J*, vol. 18, no. 12, pp. 5151–5163, Jun. 2018, doi: 10.1109/JSEN.2018.2832291.

- [16] H. Y. Lin, J. M. Dai, L. T. Wu, and L. Q. Chen, "A vision-based driver assistance system with forward collision and overtaking detection," *Sensors (Switzerland)*, vol. 20, no. 18, pp. 1–19, Sep. 2020, doi: 10.3390/s20185139.
- [17] W. Farhat, C. Souani, O. Ben Rhaïem, and H. Faïedh, "Cooperative Forward Collision Avoidance System Based on Deep Learning," in *Proceedings - International Conference on Developments in eSystems Engineering, DeSE*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 515–519. doi: 10.1109/DESE54285.2021.9719448.
- [18] N. Albarella, F. Masuccio, L. Novella, M. Tufo, and G. Fiengo, "A forward-collision warning system for electric vehicles: Experimental validation in virtual and real environment," *Energies (Basel)*, vol. 14, no. 16, Aug. 2021, doi: 10.3390/en14164872.
- [19] Guo li zhong xing da xue, IEEE Consumer Electronics Society, Institute of Electrical and Electronics Engineers, and IEEE Signal Processing Society, *Intelligent Vehicle Collision System Based on A Deep Learning Approach - 2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW) : 19-21 May 2018*.
- [20] M. F. Kabir and S. Roy, "Real-time vehicular accident prevention system using deep learning architecture," *Expert Syst Appl*, vol. 206, Nov. 2022, doi: 10.1016/j.eswa.2022.117837.
- [21] D. Kim, J. Choi, H. Yoo, U. Yang, and K. Sohn, "Rear obstacle detection system with fisheye stereo camera using HCT," *Expert Syst Appl*, vol. 42, no. 17–18, pp. 6295–6305, May 2015, doi: 10.1016/j.eswa.2015.04.035.
- [22] A. Leu, D. Aiteanu, and A. Gräser, *A novel stereo camera based collision warning system for automotive applications - 2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics*. IEEE, 2011.
- [23] IEEE Robotics and Automation Society and Institute of Electrical and Electronics Engineers, "Real-Time Forward Collision Warning system using nested Kalman filter for monocular camera - IEEE ROBIO 2018 : 2018 IEEE International Conference on Robotics and Biomimetics : December 12-15, 2018, Kuala Lumpur, Malaysia."
- [24] V. G. Krishnan, P. V. Rao, M. Navaneethakrishnan, V. Prabhu, D. Parkavi, and T. Tamilarasan, "Improvement of Safety in Road Environment by Designing an Advanced Driver Assistance System," in *International Conference on Automation, Computing and Renewable Systems, ICACRS 2022 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 907–911. doi: 10.1109/ICACRS55517.2022.10029157.
- [25] IEEE Staff, *Current Challenges in Modelling Advanced Driver Assistance Systems: Future Trends and Advancements - 2017 2nd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*. IEEE, 2017.
- [26] J. L. Trincado Alonso, B. Torres Salcedo, and A. Pérez Alonso, "Sistema de Visión Estereoscópica para Navegación Autónoma de vehículos no tripulados," 2010.

- [27] M. Peris Martorell, "Aproximacion del Mapa de Disparidad Estereo Mediante Tecnicas de Aprendizaje Automatico," Sep. 2011.
- [28] V. H. Velasco, E. González Juárez, I. Robledo Vega, A. Pacheco González, and R. Baray Arana, "Comparación de Mediciones de Profundidad entre un Sistema de Visión Estéreo y un Sensor RGBD," *Congreso Internacional en Ingeniería Electrónica. Mem. Electro 2015, Vol. 37, pp 187-192, Chihuahua, Chih. México*, pp. 187–192, 2015.
- [29] J. M. Guerrero Hernández, "Técnicas de procesamiento de imágenes estereoscópicas," 2009.
- [30] M. M. Martínez, "Técnicas de visión estereoscópica para determinar la estructura tridimensional de la escena," 2010.
- [31] Baeldung, "What Is Image Stitching?," <https://www.baeldung.com/cs/image-stitching#bd-what-is-image-stitching>. Accessed: Aug. 21, 2025. [Online]. Available: <https://www.baeldung.com/cs/image-stitching#bd-what-is-image-stitching>
- [32] Geeksforgeeks, "Image Stitching with OpenCV," <https://www.geeksforgeeks.org/computer-vision/image-stitching-with-opencv/>. Accessed: Aug. 21, 2025. [Online]. Available: <https://www.geeksforgeeks.org/computer-vision/image-stitching-with-opencv/>
- [33] J. Sebastian and G. Prieto, "Redes neuronales convolucionales y redes neuronales recurrentes en la transcripción automática," 2019, doi: 10.13140/RG.2.2.10855.39843.
- [34] IBM, "¿Qué son las redes neuronales convolucionales?" Accessed: Mar. 26, 2024. [Online]. Available: <https://www.ibm.com/es-es/topics/convolutional-neural-networks>
- [35] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," Dec. 25, 2006. doi: 10.1145/1177352.1177355.
- [36] M. Abouelyazid, "Evaluation of SORT, DeepSORT, and ByteTrack for Multiple Object Tracking in Highway Videos - 42 International Journal of Sustainable Infrastructure for Cities and Societies Comparative."
- [37] F. Lotfi, K. Virji, and G. Dudek, "Robust Scuba Diver Tracking and Recovery in Open Water Using YOLOv7, SORT, and Spiral Search," in *Proceedings - 2023 20th Conference on Robots and Vision, CRV 2023*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 233–240. doi: 10.1109/CRV60082.2023.00037.
- [38] F. Za. A. H. Aadi, A. Sadiq, and Z. Labd, "Comparing Object Tracking Algorithms for Real-Time Applications: Performance Analysis and Implementation Study," in *Proceedings - 10th International Conference on Wireless Networks and Mobile Communications, WINCOM 2023*, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/WINCOM59760.2023.10322976.
- [39] N. Aharon, R. Orfaig, and B.-Z. Bobrovsky, "BoT-SORT: Robust Associations Multi-Pedestrian Tracking," Jun. 2022, [Online]. Available: <http://arxiv.org/abs/2206.14651>

- [40] M. C. Munuera Raga, "Filtro de Kalman y sus aplicaciones," Barcelona, Jun. 2018.
- [41] D. Pampliega Ruiz, "Algoritmo de SLAM para el ROMEO-4R," Sevilla, 2008.
- [42] IEEE Industrial Electronics Society., Sichuan da xue., and Keisoku Jidō Seigyo Gakkai (Japan), *Study on Vehicle Safety Distance Warning System - 2008 IEEE International Conference on Industrial Technology : IEEE ICIT 2008 : 21-24 April 2008, Sheraton Chengdu Lido Hotel, Chengdu, China : conference proceedings*. IEEE Xplore, 2008.
- [43] W. Wang, J. Xi, and D. Zhao, "Learning and inferring a driver's braking action in car-following scenarios," *IEEE Trans Veh Technol*, vol. 67, no. 5, pp. 3887–3899, May 2018, doi: 10.1109/TVT.2018.2793889.
- [44] X. Wang, M. Chen, M. Zhu, and P. Tremont, "Development of a Kinematic-Based Forward Collision Warning Algorithm Using an Advanced Driving Simulator," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 9, pp. 2583–2591, Sep. 2016, doi: 10.1109/TITS.2016.2528508.
- [45] "Capítulo 5 Proceso Unificado Rational Aplicado." Accessed: May 28, 2023. [Online]. Available: <http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.100/175/A8%20Capítulo%205.pdf?sequence=8>
- [46] V. Ferrer Gutiérrez, "Calibración de una cámara por seguimiento de características en una secuencia de imágenes," Sep. 2004.
- [47] Stereo Labs, "What is the calibration file?" Accessed: May 27, 2024. [Online]. Available: <https://support.stereolabs.com/hc/en-us/articles/360007497173-What-is-the-calibration-file>
- [48] Pomares Jorge, G. J. García, and F. Torres, "Control visual para el seguimiento intemporal de trayectorias empleando el espacio de normas de la matriz de homografía 2D." [Online]. Available: <https://www.researchgate.net/publication/39438086>
- [49] Stereo Labs, "Camera Calibration." Accessed: May 27, 2024. [Online]. Available: <https://www.stereolabs.com/docs/video/camera-calibration>
- [50] Timothy M., "What is Scale-Invariant Feature Transform (SIFT)?," <https://blog.roboflow.com/sift/>.
- [51] Stereolabs Inc, "Performance Benchmark of YOLO v5, v7 and v8." Accessed: Feb. 04, 2025. [Online]. Available: <https://www.stereolabs.com/blog/performance-of-yolo-v5-v7-and-v8>
- [52] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, and C. Tomasi, "Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking," Sep. 2016, [Online]. Available: <http://arxiv.org/abs/1609.01775>

- [53] J. Luiten *et al.*, “HOTA: A Higher Order Metric for Evaluating Multi-Object Tracking,” Sep. 2020, doi: 10.1007/s11263-020-01375-2.
- [54] Y. Zhang *et al.*, “ByteTrack: Multi-Object Tracking by Associating Every Detection Box,” Oct. 2021, [Online]. Available: <http://arxiv.org/abs/2110.06864>
- [55] V. Tadic *et al.*, “Perspectives of RealSense and ZED Depth Sensors for Robotic Vision Applications,” Mar. 01, 2022, *MDPI*. doi: 10.3390/machines10030183.

7. ANEXOS

En este apartado se deja la constancia de los documentos referentes a la metodología RUP del desarrollo del sistema. Estos se encuentran alojados en el siguiente enlace a GoogleDrive:

<https://drive.google.com/drive/folders/1yxLmgLPu8G02d61FAySaklrk4vVYSQ0q?usp=sharing>

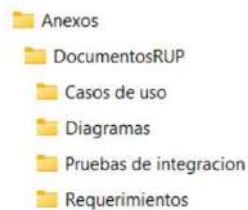


Figura 7-1. Estructura Anexos.

- En el directorio “Casos de uso” se encuentra el archivo PDF referente a casos de uso real.
- En el directorio “Diagramas” se encuentran los archivos referentes a los diagramas hechos.
- En el directorio “Requerimientos” se encuentran los archivos PDF referentes a requerimientos funcionales y no funcionales.
- En el directorio “Pruebas de integración” se encuentra en archivo PDF referente a las pruebas de integración.