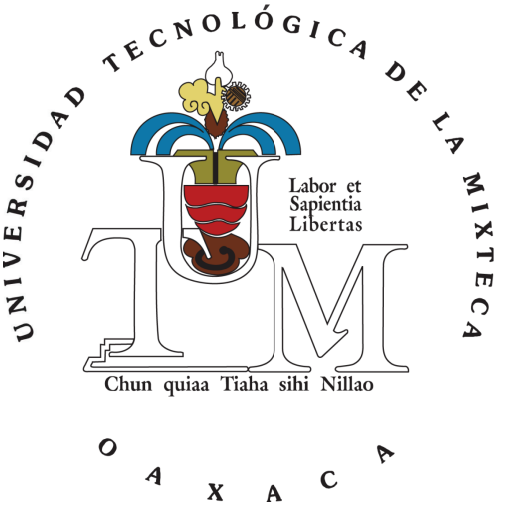


Universidad Tecnológica de la Mixteca

División de Estudios de Posgrado

Diseño de algoritmos para la navegación autónoma del vehículo "AutoNOMOS mini v2" con obstáculos estáticos y móviles en un entorno controlado

T E S I S

Para obtener el grado de
MAESTRO EN ROBÓTICA

Presenta:
Ing. Filiberto Eduardo Martínez Hernández

Director:
Dr. José Anibal Arias Aguilar
Codirector:
Dr. Edgar Macías García

H. Cd. de Huajuapan de León, Oaxaca, México, Enero del 2024

Tesis presentada en enero de 2024
ante los siguientes sinodales:

Dr. Oscar David Ramírez Cárdenas
Dr. Rosebet Miranda Luna
Dr. Fermin Hugo Ramírez Leyva
Dr. Christian Eduardo Millán Hernández

Director de tesis:
Dr. José Anibal Arias Aguilar

Codirector de tesis:
Dr. Edgar Macías García

Con mucho cariño a mis padres
porque siempre me enseñaron a
nunca rendirme, y a mi hermana,
por su claro ejemplo de
perseverancia.

Agradecimientos

A mi padre y a mi madre, Filiberto y Rocelia, por su amor, su apoyo, sus grandes sacrificios, sus regaños, pero sin duda, por la dedicación que me han otorgado a lo largo de mi educación para que nunca me falte nada. Porque gracias a todo eso y más, soy una persona íntegra, de principios y valores. Porque sin ustedes, jamás lo hubiese logrado.

A mi única hermana, Jazmín, por cuidar de todos en casa, por dejarme ser su ejemplo, y por ser mi ejemplo, por el apoyo incondicional que me otorga cada que lo necesito, por los momentos de alegría y diversión.

A Mariana, deseo expresar mi profundo agradecimiento por su inquebrantable apoyo durante el desarrollo de este proyecto de tesis. Por su presencia constante, la disposición que tiene para escucharme, por sus valiosos consejos. Indirecta y directamente ha formado parte de este proyecto; me ha alentado constantemente a no rendirme, su compañía ha sido un faro de luz en los momentos más difíciles, y sus habilidades destacadas en el diseño ha sido fundamental para la edición visual de este documento.

Agradezco profundamente a mis mejores amigos, Denes Luna (Denes), por guiarme hacia la serenidad en la toma de decisiones; a Daniel Rivera (Daimer), por su perseverancia y por demostrarme que los sueños son alcanzables; a Daniel Luna (Dani), por su habilidad para motivarme incluso en los momentos más desafiantes, a Rafael Castro (Anfa), por mostrarme el camino de la justicia y la libertad, y a Diego Ibarra (Diego), por su hábil persuasión e ideas innovadoras. A todos ustedes, les expreso mi agradecimiento por su apoyo incondicional a lo largo de mi formación académica. Agradezco sus consejos oportunos, su disposición para escucharme y la constante motivación que me brindaron. Los momentos compartidos, ya sean en concursos académicos o en el día a día, han sido invaluables. Gracias por alentarme a dar siempre el 100 % y por seguir siendo mis amigos.

A mi director de tesis, el Dr. Anibal Arias, por el inestimable respaldo proporcionado a lo largo de estos dos años. Su vasto conocimiento y la habilidad para transmitirlo a sus estudiantes, especialmente a mí, ha sido fundamental para guiar esta investigación. Además, valoro su integridad al siempre tomar decisiones correctas y ejercer su papel de líder justo como jefe de la División de Posgrado. De la misma manera, mi profundo agradecimiento al Dr. Edgar Macías García, mi codirector de tesis, por creer en mí, por orientarme desde la licenciatura en este mundo lleno de conocimientos, por tomarse siempre el tiempo y la dedicación de mi enseñanza, sin duda el Dr. Edgar aportó su vasto conocimiento en la tesis como en los artículos; y en los tiempos más difíciles personalmente de incertidumbre, siempre estuvo ahí para orientarme con mi futuro.

Al Dr. Oscar Ramírez, por su apoyo técnico en este proyecto, nunca antes había conocido a un profesor tan entusiasta y dedicado en sus enseñanzas; le agradezco por su disposición constante para recibirme y brindarme orientación en conceptos e ideas que en ocasiones resultaban complejas, su amabilidad va más allá de las obligaciones académicas, convirtiéndose en un amigo y guía valioso durante todo el proceso. Y a mis demás sinodales, el Dr. Rosebet, el Dr, Hugo Leyva, y el Dr. Christian Millán por su por su invalorable conocimiento en dirección de este proyecto.

A Canela y a Merlina, las dos más hermosas compañeras de cuatro patas de la familia, con la más sincera emoción de Canela siempre que la visito, y a Merlina, que llegó a la mitad de la maestría, y ya forma parte de la familia. Ambas se han convertido en parte esencial de mi vida, quienes esperan ansiosamente mi llegada a casa. El simple hecho de visitarlas se ha convertido en una fuente de felicidad y, de cierta manera, en una motivación adicional para abordar con determinación los desafíos de este proyecto.

Y por último, a la Universidad Tecnológica de la Mixteca, por sus instalaciones y su personal en general, pero principalmente a la División de Estudios de Posgrado, profesores y técnicos. Al Consejo Nacional de Humanidades, Ciencia Y Tecnología (CONAHCYT), puesto que sin el apoyo económico otorgado durante estos dos años el proyecto no hubiese sido posible, y por creer en las mentes de este país.

Resumen

En este proyecto de investigación se proponen técnicas de navegación autónoma en tiempo real basadas en redes neuronales convolucionales, implementadas en simulación utilizando ROS-Gazebo e implementando en un prototipo real. El prototipo en simulación e implementación real es un robot móvil de configuración Ackerman (AutoNOMOS Mini V2), y en ambos casos tiene una escala 1:10 en comparación a un auto convencional. La velocidad de desplazamiento es constante, sin embargo, la red neuronal convolucional controla y decide el ángulo de dirección, en ese sentido, la entrada de la red es la imagen observada por una cámara RGB colocada en la parte superior del prototipo y centrada, mientras que la salida es el ángulo de dirección.

El enfoque principal en el que está basada esta investigación, es la propuesta de los modelos para realizar ciertas actividades: navegación autónoma sin obstáculos, con obstáculos estáticos y con obstáculos móviles; estas tareas forman parte de la categoría AutomodelCar del TMR.

Para su implementación en tiempo real y sobre el prototipo real, es necesario utilizar un sistema de gran velocidad, por lo tanto, se aplicaron ciertas actualizaciones para que la navegación autónoma del AutoNOMOS Mini V2 sea posible. Por otro lado, crear una base de datos para cada una de las tareas es primordial para que el aprendizaje de los modelos pueda ser integrada. Tanto los modelos y entrenamiento, como las bases de datos etiquetadas para las tareas específicas, y el movimiento del prototipo, son las principales contribuciones de este proyecto.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Existen muchos avances dentro del área de la conducción autónoma, entre ellos se encuentran los Sistemas Avanzados de Asistencia a la Conducción (ADAS), los cuales con ayuda de sensores, cámaras y sistemas de navegación hacen la conducción más segura. Sin embargo, estos sistemas solo son una ayuda para el usuario. Empresas como Tesla o Google ya han emigrado de estos sistemas para convertirse en pioneros sobre la conducción totalmente autónoma. Pero el sector privado no es el único que ha comenzado a implementar sus estudios en estos temas; universidades como la de Parma en Italia (Broggi et al., 2015), la Universidad Libre de Berlín o la universidad Tecnológica de la Mixteca han desarrollado diferentes algoritmos basados en prototipos a escala de autos simulando uno real. Esto ha permitido que concursos en todo el mundo se comiencen a realizar, entre ellos el más importante en Alemania (Carolo-Cup), y destacando la reciente categoría implementada en el Torneo Mexicano de Robótica (TMR, por sus siglas) sobre autos autónomos, pero eso no es lo único, ya que empresas como Amazon también han desarrollado plataformas virtuales para el desarrollo de algoritmos por medio concursos (como AWS DeepRacer).

Es por ello que en el presente trabajo se busca desarrollar algoritmos que, basados en la información reunida por sensores en un vehículo a escala, hagan posible la navegación autónoma con obstáculos estáticos y móviles en un entorno controlado, tomando como referencia algunas tareas propuestas por los concursos del Carolo-Cup y del TMR y contemplando las métricas que utilizan como calificación.

1.1. Antecedentes

En esta sección se describen algunos de los algoritmos de navegación autónoma más relevantes encontrados en la literatura, así como plataformas, sistemas, y pruebas internacionales. Por lo tanto, las áreas analizadas se encuentran a nivel comercial y a nivel académico, en ambas se detallan las características, métodos de adquisición de datos y configuraciones de implementación, además del aporte que cada una realiza.

1.1.1. Proyectos comerciales

En el área empresarial existen dos organizaciones pioneras en el desarrollo de algoritmos para navegación autónoma, las cuales se describen a continuación.

Tesla. De acuerdo con (Tesla, 2021) esta es la empresa más importante en avances tecnológicos dentro de la conducción autónoma, la cual cuenta con su propio sistema *Autopilot* y quién asegura contar en el futuro con funciones de conducción completamente autónoma a través de actualizaciones en los vehículos.

El sistema de percepción de Tesla, denominado Tesla Vision, se compone de 8 cámaras periféricas que cubren los 360° de visibilidad y con un alcance de hasta 250 m. A esta visión la complementan 12 sensores ultrasónicos, lo que permite una mejor detección de objetos con diferentes formas. Este sistema está desarrollado sobre la base de una red neuronal profunda, la cual permite garantizar mayores niveles de confiabilidad que aquellos alcanzados con técnicas clásicas de visión artificial. La Tabla 1.1 explica cada sección de cámaras y sensores ubicados por todo el vehículo.

Waymo. Adicional a Tesla, otra empresa de gran notoriedad en el desarrollo de algoritmos para navegación autónoma es Waymo (Waymo, 2019), una división especializada de Google. Su sistema de percepción llamado *Waymo Driver* puede ser utilizado en diferentes tipos de vehículos desde camionetas, vehículos propios (Waymo one, Figura 1.1) hasta camiones de clase 8 (Waymo via).

A diferencia de Tesla y otras empresas, al ser Waymo parte de google, su sistema de tecnología se basa en mapas anteriormente descargados para generar un mapa con gran

Adquisición	Información	
Cámaras delanteras	Amplio	Lente de ojo de pez de 120° con capacidad de capturar las luces de tráfico, obstáculos que pueden aparecer en el camino y objetos cercanos. Útil en maniobras urbanas a bajas velocidades.
	Principal	Cubre una amplia gama de usos para diversos casos.
	Estrecho	Provee una vista enfocada de largo alcance. Útil en operaciones a velocidades altas.
Cámaras laterales	Visión al frente	Tienen una vista de 90° que permiten detectar autos que ingresan inesperadamente al carril de uso, además de proveer seguridad adicional cuando el auto ingresa a intersecciones con visibilidad limitada.
	Visión trasera	Monitorean los puntos ciegos de ambos lados del automóvil, un aspecto importante para el cambio de carril e incorporación de manera segura.
Cámaras de visión trasera	Toda la parte trasera	En versiones anteriores de los Tesla esta cámara apoyaba la visión trasera de manera segura, hoy en día, la cámara trasera con óptica mejorada es un elemento importante de la suite del Autopilot ya que es usada para las maniobras del estacionado autónomo.
Sensores ultrasónicos	Todo el auto	Estos sensores son útiles para detectar autos cercanos, en especial cuando se incorporan al carril del auto Tesla, además que brindan ayuda al estacionarse.

Tabla 1.1: Adquisición de datos en Tesla

nivel de detalle, el cual incluye desde marcadores de carriles, hasta señales de alto, bordillos y aceras. De esta manera, en vez de hacer uso únicamente de datos externos (como pueden ser aquellos provenientes del GPS), Waymo utiliza los mapas personalizados y los combina con los datos recabados de los sensores y las cámaras integradas en el vehículo con el fin de determinar la posición del vehículo en tiempo real.

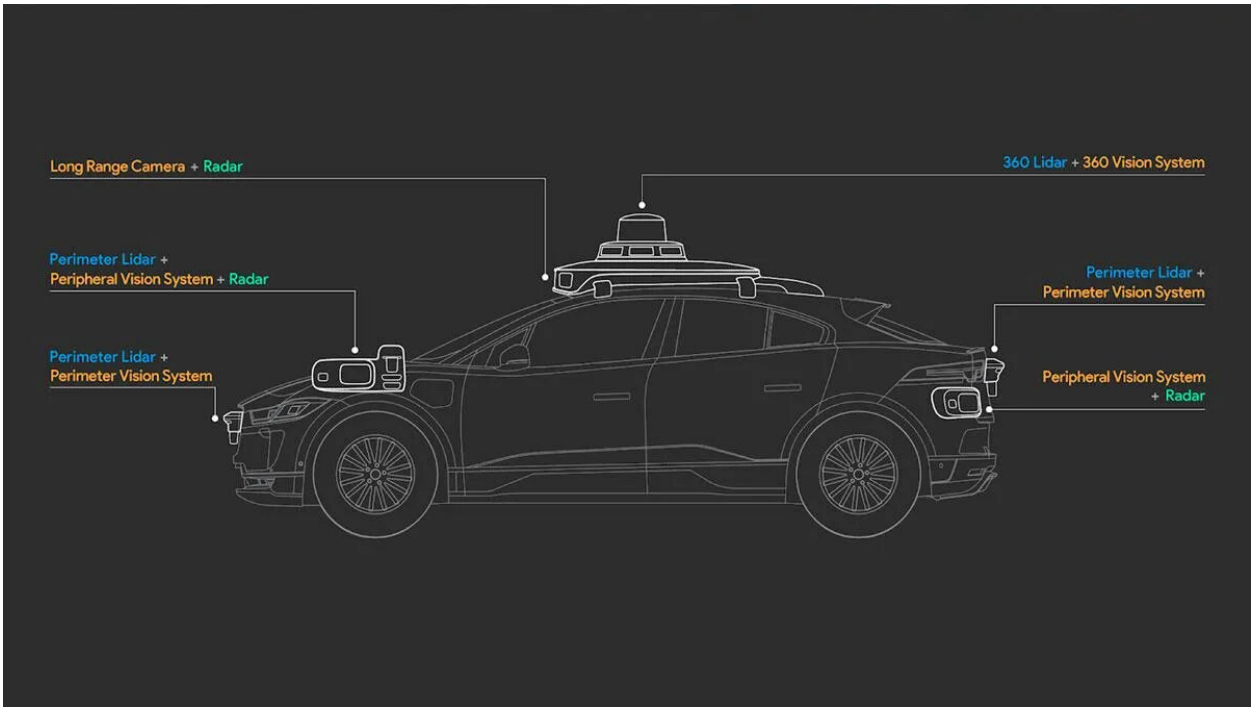


Figura 1.1: Vehículo Waymo one. Fuente: (Andrés, 2020)

Algunas características importantes que el sistema contempla son:

- Predicción ante situaciones:
  Los vehículos Waymo intentan predecir los comportamientos que están a punto de suceder alrededor del vehículo, de manera que el sistema sea lo más seguro posible.

- Planificación del viaje:
  Una vez ingresada la ruta final del vehículo, este planificará la trayectoria a seguir y la que será de mayor conveniencia, a modo que el comportamiento del vehículo sea lo más seguro a lo largo del viaje.

El sistema de adquisición de datos de Waymo se compone de cuatro elementos principales, que se describen en la Tabla 1.2

Adquisición	Información
LiDAR	Está empleado en la parte más alta del vehículo, tiene una vista de todas las direcciones del vehículo.
Cámaras	Vista simultánea en los 360° alrededor del vehículo. Diseñados con un alto rango dinámico y estabilidad térmica, lo que permite la visibilidad en condiciones adversas y de poca luz. Las cámaras permiten a su vez detectar la información de la luz de tráfico, construcciones u otras escenas en el camino. Las diferentes versiones del sistema incluyen desde 19 hasta 29 cámaras.
Radar	Usa frecuencias milimétricas para enviar información importante al sistema, como por ejemplo distancia y velocidad de algún objeto. El sensor sigue operando de manera normal ante condiciones de lluvia, nieve o niebla.
Procesamiento	El sistema cuenta con un dispositivo ubicado en la parte posterior del vehículo, encargado de recopilar y procesar la información de los diferentes sensores.

Tabla 1.2: Adquisición de datos en Waymo

1.1.2. Plataformas y proyectos de investigación

Adicional a los sistemas desarrollados por empresas automotrices, en el contexto académico también se han desarrollado contribuciones en el área, a continuación se describen algunos proyectos y plataformas de relevancia dentro de la investigación sobre autos autónomos.

NVIDIA. PilotNet es una red neuronal profunda con aprendizaje End-to-End que permite guiar la navegación de un vehículo autónomo por medio del cálculo de su dirección de giro. Si bien esta es la principal característica de la red, también es capaz de reconocer objetos que se encuentran sobre el camino, como por ejemplo otros vehículos, postes e incluso objetos salientes de los costados del camino, como pueden ser arbustos, según detalla (Bojarski et al., 2016).

La Figura 1.2 muestra el diagrama a bloques del sistema de entrenamiento propuesto por NVIDIA. Las imágenes de entrenamiento son introducidas a una red neuronal vonvolucional (CNN, por sus siglas en inglés), el cual, durante el entrenamiento propone una dirección de giro que es comparada con una previamente calculada (o deseada), y cuya diferencia

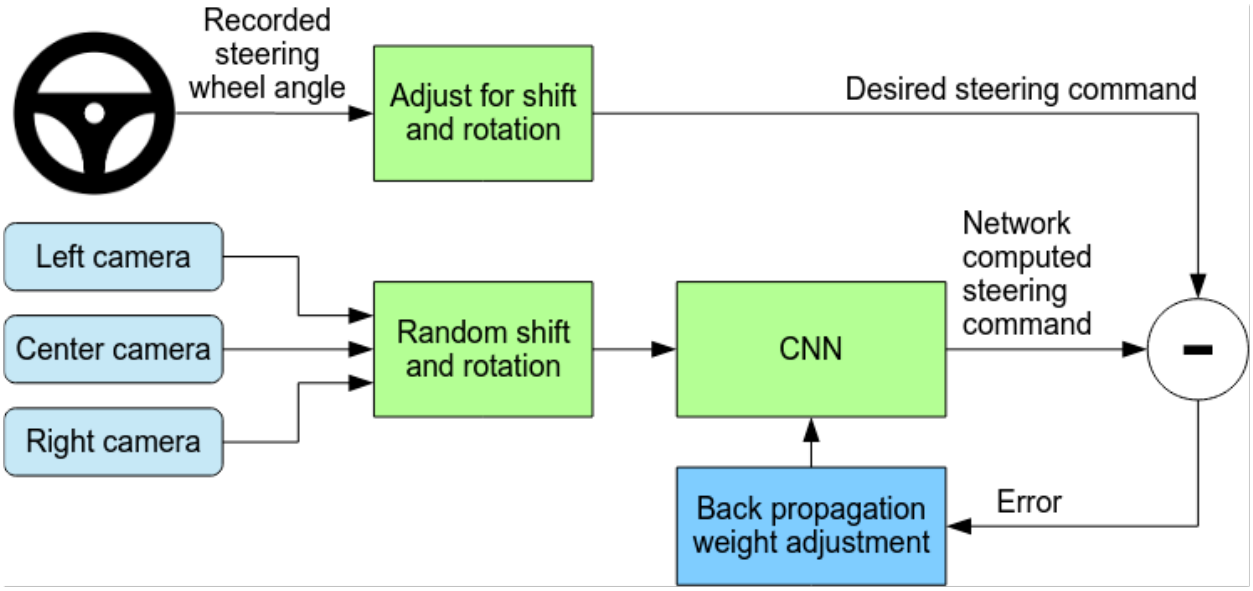


Figura 1.2: Diagrama a bloques del sistema PilotNet. Fuente: (Bojarski et al., 2016)

(empleando el error medio cuadrático) es utilizada para ajustar los parámetros del modelo.

La red entrenada es usada para obtener las direcciones del vehículo utilizando únicamente la cámara central. Esta red consta de 9 capas; 5 capas convolucionales (para extracción de características en las imágenes de entradas), una de normalización y 3 completamente conectadas (para el procesamiento de las características y el cálculo del ángulo de giro). La imagen de entrada se divide en planos YUV que son introducidos como entrada a la red neuronal (Figura 1.3).

BRAiVE. Uno de los principales trabajos académicos en el campo de la autonomía de vehículos se describe en (Broggi et al., 2015), donde en 2013 la Universidad de Parma, en Italia, probó por primera vez su vehículo autónomo denominado BRAiVE sobre una carretera, el cual pudo navegar una distancia de 13 Km. de forma completamente autónoma.

El sistema de adquisición de datos del vehículo contaba con dos escáneres láser Hokuyo UTM-30LX-EW montados en la parte delantera del vehículo, mientras que el sistema de posicionamiento empleaba un Topcon AGI3 con corrección RTK (Navegación Cinética Satelital) en tiempo real de GEOTOP (programa diseñado para que la computadora “Palm”se utilice como “colectora de datos”).

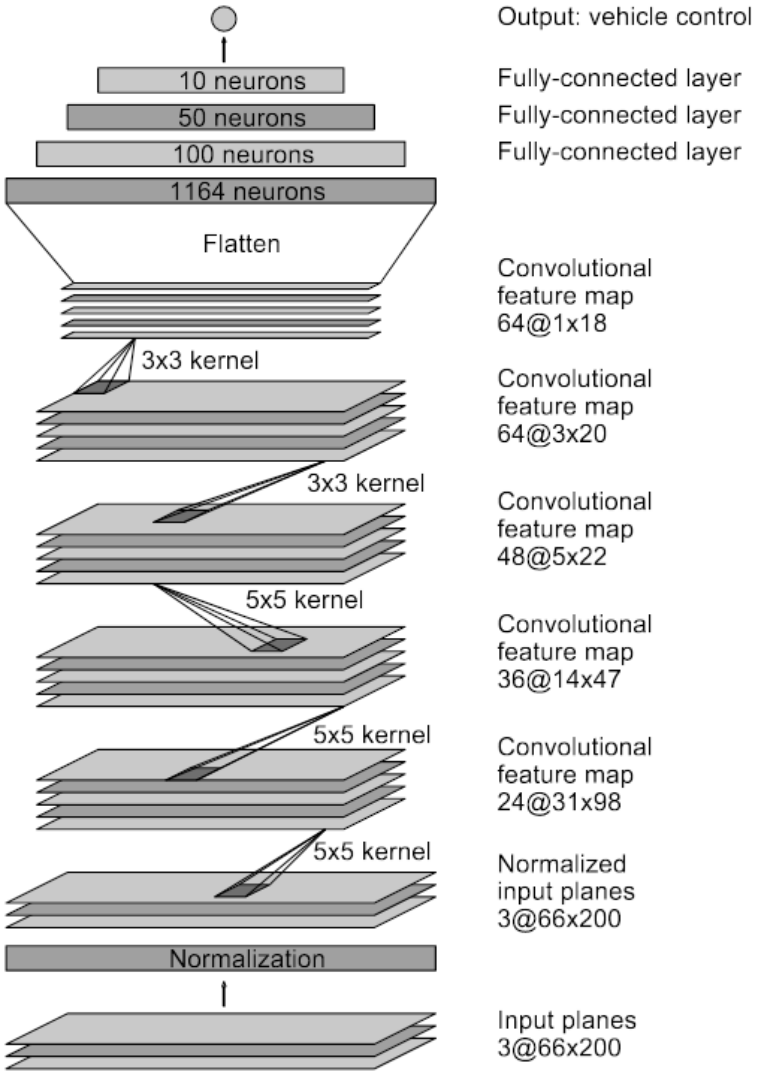


Figura 1.3: Red neuronal convolucional propuesta por NVIDIA. Fuente: (Bojarski et al., 2016)

El procesamiento del sistema de percepción se dividió en 3 computadores especializadas en las diferentes orientaciones del vehículo (Parte trasera, frontal y lateral), alcanzando una cobertura total de 360°, la Figura 1.4 representa el campo de visión de los sensores:

- Detección de obstáculos y carriles frontales mediante cuatro cámaras frontales y 4 láseres apuntando en esa dirección, siendo el central un láser IBEO (Sensor LiDAR).
- Detección de obstáculos laterales mediante dos escáneres láser de una sola capa y dos cámaras montadas en las esquinas del parachoques delantero.

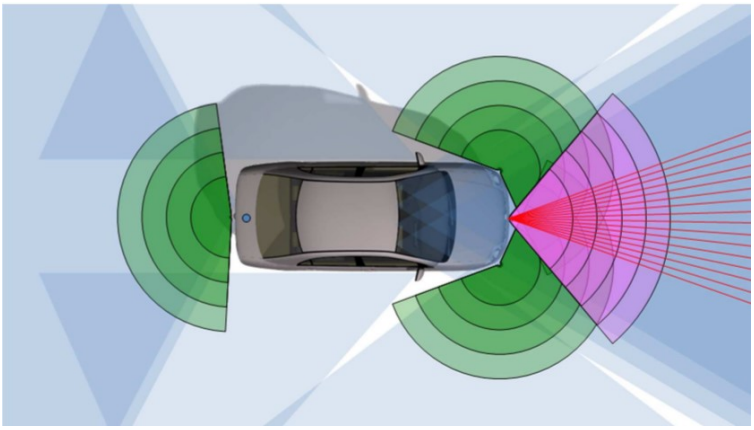


Figura 1.4: Campo de visión del BRAiVE. Fuente: (Grisleri y Fedriga, sf)

- Detección de obstáculos traseros con acercamiento a marcas viales (pasos de peatones, líneas de parada, etc.) y semáforos mediante un escáner láser y una cámara montados en el parachoques trasero, los espejos retrovisores externos también cuenta con cámaras montadas orientadas hacia atrás.

Las especificaciones de los sensores empleados en el sistema de percepción se describen en la Tabla 1.3 (Grisleri y Fedriga, sf) y la Figura 1.4. Donde se pueden observar dos cúmulos verdes frontales (generados por dos sensores Hokuyo), un tercer sensor en la parte trasera, y un láser Lux al frente (en color rosa). Adicionalmente, el sistema dispone de un sensor IDIS (rojo), que cuenta con un rango de adquisición mayor pero una menor apertura en su campo de visión.

Modelo del láser	FoV	Rango
Hokuyo UTM-30LX	270°	$0{,}1 - 30\ m$
IBEO Lux	85°	$0{,}3 - 80\ m$
Hella IDIS	16°	$0{,}7 - 110\ m$

Tabla 1.3: Especificaciones de los láseres

Si bien BRAiVE ya contaba con buena autonomía al momento de desplazarse de un lugar a otro, aún hacía falta regular algunos detalles, por ejemplo, este prototipo debía tener un conductor quién apoyaba en algunas actividades que el sistema no podía realizar.

A fin de facilitar el desarrollo de algoritmos de navegación a pequeña escala, se han desarrollado prototipos como el AutoNOMOS Mini, el cual en sus diferentes versiones ha sido la plataforma de desarrollo de diferentes centros de investigación alrededor del mundo.

Plataformas a escala de investigación. En México se han desarrollado proyectos en algunas Universidades y centros de investigación sobre la plataforma del AutoNOMOS Mini. Uno de ellos es (Bravo, 2018) del Instituto Politécnico Nacional, quién propuso una metodología a implementar en el AutoNOMOS Mini versión 2 para la navegación autónoma y evasión de obstáculos. De acuerdo a sus investigaciones, el prototipo AutoNOMOS es capaz de realizar diferentes tareas de navegación como lo es el seguimiento de carril y cruce de intersecciones, así como el seguimiento y rebase de obstáculos de manera satisfactoria. De la misma manera propone una metodología diferente para la detección de carriles, que consiste en la búsqueda de máximos locales para la detección de puntos de línea de carril sobre un histograma desarrollado a partir de la imagen, siendo eficaz para diferentes condiciones en la iluminación.

Así mismo (Diaz, 2020) desarrolló un algoritmo en la Universidad Tecnológica de la Mixteca para la reconstrucción de un mapa métrico generado a partir de la cámara Intel Real Sense a bordo del AutoNOMOS Mini V2. Esta reconstrucción forma parte de una tarea a realizar en un concurso nacional de robótica que se realiza en México año con año (Torneo Mexicano de Robótica). Los pasos que se siguieron para generar el mapa fueron los siguientes:

- Pre-procesamiento de las imágenes.
  - Corrección de perspectiva.
- Aplicación de técnicas para detección de líneas de carril, diferenciadas en su método y aplicación.
- Aplicación de un algoritmo para la asociación de componentes en imágenes consecutivas.
- Aplicación de un algoritmo para el cálculo de la matriz de transformación entre dos conjuntos de puntos.
- Reconstrucción de un mapa de una pista real y de un entorno virtual.

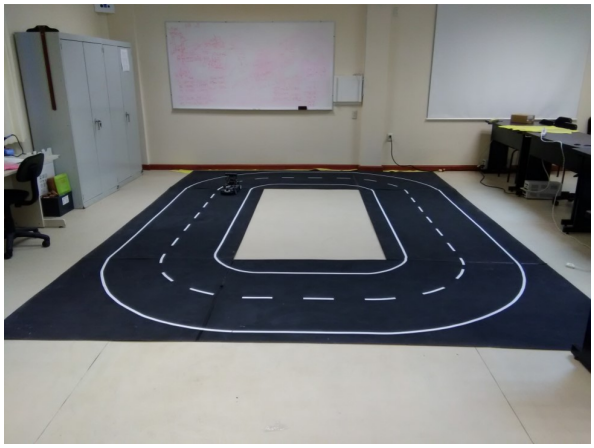


Figura 1.5: Pista real empleada para generar un mapa métrico. Fuente: (Diaz, 2020)

La Figura 1.5 representa la pista real de la cual se reconstruyó el mapa, y la Figura 1.6 muestra el resultado obtenido a partir de las técnicas usadas. Como puntos a resaltar en los resultados, la pista real tiene errores de segmentación debido al fondo en el que esta montada la pista, cuyo color es muy cercano al blanco. Estos errores agregan pixeles erróneos a la reconstrucción, sin embargo son pocos en comparación a los correctos.

La Figura 1.6 cuenta con partes de las líneas derecha e izquierda donde hay discontinuidades, estas son debidos a errores de clasificación en líneas rectas y a perdida de información por el campo de visión de la cámara. Sin embargo, basados en los resultados experimentales y medidas de error cuantitativas la respuesta de los algoritmos fue considerada como favorable, cumpliendo así los objetivos propuestos.

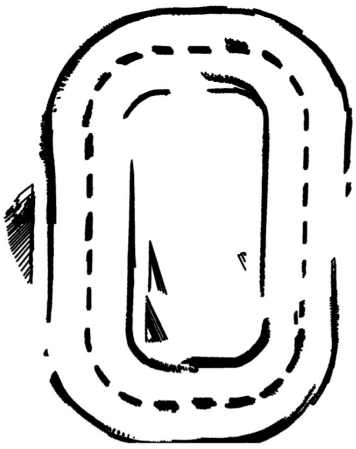


Figura 1.6: Reconstrucción del mapa a partir de la pista real. Fuente: (Diaz, 2020)

Basado en los primeros prototipos de autos autónomos, se ha desarrollado el concurso llamado DARPA Challenge, el cual tiene por objetivo llegar de un punto de Estados Unidos a otro pasando por puntos de control sin intervención humana. Sin embargo, participar en estos concursos involucra una elevada inversión al desarrollar un auto autónomo, por esta razón la Universidad Libre de Berlín ha desarrollado el primer prototipo que tiene por nombre AutoNOMOS Mini, y a su vez ha creado el primer concurso de coches autónomos denominado Carolo-Cup, con la única regla de que los prototipos cuenten con una escala 1:10 en comparación a un auto normal, descrito en (Nolte et al., 2018). La Figura 1.7 representa algunos momentos del Carolo-Cup.

Esto abre las posibilidades para que diferentes universidades y centros de investigación puedan desarrollar las habilidades en el tema de los autos autónomos sin la necesidad de dotar a un coche de escala 1:1 con sistemas de procesamiento y percepción.

El concurso Carolo-Cup califica los siguientes aspectos:

- Arquitectura de hardware y aspectos de construcción.
- Arquitectura de software y soluciones generales de algoritmos.
- Consumo de energía.
- Aspectos financieros.
- Gestión del proyecto y gestión del conocimiento.
- Soluciones de algoritmos específicos para las tareas individuales (estacionamiento, seguimiento de carril, evitación de obstáculos, reconocimiento de señal de tráfico).
- Control del vehículo (Longitudinal, lateral, así como el control para estacionarse).

A partir de las primeras versiones del prototipo AutoNOMOS, se han desarrollado en México diferentes concursos año con año, estos concursos contemplan utilizar cualquier versión del prototipo para implementar algoritmos que puedan realizar navegación autónoma en un espacio controlado, tal es el caso del Torneo Mexicano de Robótica (TMR, 2018), quién a partir del 2022 ha optado por una nueva categoría basada en simulación, donde se evalúa la respuesta de los algoritmos bajo un ambiente virtual. Así mismo, Amazon ha lanzado su propio concurso para América latina denominado AWS DeepRacer (AWS, 2021), donde el prototipo es un coche de carreras autónomo hecho a escala, impulsado en un circuito virtual diseñado en la nube bajo su plataforma AWS (Servicios web de Amazon, por sus siglas eb inglés), este concurso se realiza año con año con la única condición de utilizar sus sistemas de desarrollo.

Figura 1.7: Fotografías en el Carolo-Cup durante el entrenamiento y competencia. Fuente: (Nolte et al., 2018)

1.2. Planteamiento del problema

Dentro del área de la conducción autónoma en autos, aún se requiere de investigación para hacer realidad la visión de un coche totalmente autónomo. Los Sistemas Avanzados de Asistencia a la Conducción (ADAS, por sus siglas en inglés), permiten realizar aspectos individuales de una tarea en particular, como pueden ser el sistema de detección de peatones y freno autónomo, la detección de ángulo muerto, o el control de crucero adaptativo (detallado en (Stöckle et al., 2020)). Del mismo modo, según (Brown, 2017), si bien al realizar conducción automática los coches tienden a mejorar su mecánica de conducción, el problema se encuentra en que la conducción siempre busca estadísticamente evitar una colisión, la cual puede no ser siempre la mejor respuesta; por ejemplo, al ser rebasado por otro auto, irónicamente se podrían incrementar los riesgos de colisión debido a que el algoritmo podría interpretar un caso erróneo, como la anticipación de un accidente, en consecuencia, aún hace falta de propuestas en la búsqueda de un sistema completamente robusto y autónomo.

Dejando de lado esta variable, una de las más importantes constantes fundamentales para realizar navegación autónoma radica en obtener información sobre el entorno que rodea al vehículo, la cual es necesaria para la navegación y localización, de tal manera que los sistemas de conducción asistida, así como la planeación de trayectorias, requieren conocer el entorno para generar una ruta y con ello una decisión, explicado en (Diaz, 2020). De acuerdo con (Rojas, 2021), dentro del área de la investigación académica, existen diferentes prototipos, entre ellos se encuentran los modelos de autos AutoNOMOS Mini, los cuales cuentan con dos cámaras, una con propiedades para percibir la profundidad y una segunda ojo de pescado, un sensor LiDAR, un sensor de orientación (IMU), así como capacidades en telecomunicaciones en redes 2.4 G.

Con la finalidad de promover el desarrollo de algoritmos de navegación autónoma sin el costo que un auto a tamaño real requiere para resolver problemas de conducción, organizaciones como (de Robótica, 2021) desarrollan concursos año con año en México, los cuales consisten en usar vehículos a escala 1:10 no tripulados y cuyo objetivo es realizar una serie de pruebas que se desarrollan sobre una superficie similar a una autopista de dos carriles, las principales pruebas a cumplir son las siguientes: Navegación autónoma sin obstáculos, navegación autónoma con obstáculos estáticos y navegación autónoma con obstáculos móviles. En ese mismo orden de ideas, los criterios a evaluar se basan en cumplir completamente cada una de estas tareas, y las puntuaciones se realizan de acuerdo al tiempo que se requiere para completar el circuito, en orden ascendente y a disposición de la cantidad de concursantes, del mismo modo hay penalizaciones en el caso en el que un prototipo no complete el circuito, se salga, o choque.

Por otro lado, estas tareas son realizadas con el uso del prototipo AutoNOMOS en sus diferentes versiones, y con ello se emplea la cámara RGB con la que cuenta para realizar las tareas con el uso principalmente de operaciones morfológicas a partir de lo captado por la cámara, lo cual genera un alto grado de dependencia por lo que esta percibe, por otro lado, la computadora con la que cuenta (Odroid XU4) permite realizar los cálculos en un campo limitado de operaciones, por lo que realizar tareas complejas resulta complicado, del mismo modo, esto limita la velocidad del prototipo, puesto que de ello dependerá la velocidad con la que la computadora realiza los cálculos para determinar la salida (ángulo de dirección).

En ese mismo sentido, la Universidad Tecnológica de la Mixteca cuenta con la versión 2 del AutoNOMOS, sin embargo, esta versión tiene algunas limitantes, como por ejemplo la comunicación exclusiva en redes 2.4G, software (sistema Ubuntu 14.04) y hardware (computadora Odroid XU4) atrasados en comparación con las nuevas tecnologías, y una locomoción que falla en algunos casos.

1.3. Justificación

De acuerdo con un estudio realizado por (IMCO, 2019) en México, la movilidad en las ciudades más grandes impacta directamente a sus habitantes, su calidad de vida, productividad laboral, salud física, mental y hasta en la vida familiar. Así mismo es un elemento clave para la competitividad de las ciudades, pues afecta de manera directa los costos de transportación y desplazamiento de sus habitantes. Una movilidad urbana competitiva significa ofrecer opciones de transporte que sean atractivas y deseables, de tal forma que se reduzcan las horas-persona que se pierden durante los traslados, así como el impacto en la salud. Es importante seguir optimizando y desarrollar el área automotriz a fin de encontrar un balance entre el usuario y el equipo que cumplan las necesidades que este requiere.

(Ladero, 2019) hace mención al gran número de fabricantes de automóviles que han apostado ampliamente por la conducción autónoma. Empresas como Audi, Mercedes Benz, BMW, General Motors y Tesla han desarrollado sus propios algoritmos y sistemas de conducción lo cual, dificulta en gran medida el acceso libre a la investigación en el tema.

Dicho lo anterior y citando a (Collado, 2009), el objetivo de los Sistemas Inteligentes de Transporte (SIT) es incrementar la seguridad, eficiencia y confort del transporte mejorando la funcionalidad de los coches y las carreteras.

En nuestro país, como ya se hizo mención, el Torneo Mexicano de Robótica (TMR, (TMR, 2018)) ha lanzado año con año un concurso que promueve el desarrollo de algoritmo de conducción autónoma, teniendo como única restricción que el prototipo sea un robot con configuración Ackerman (el mismo que un auto 1:1). Por tal motivo, la pertinencia de esta investigación se centra en el desarrollo, optimización e implementación de algoritmos que permitan proveer a un vehículo de las capacidades para desarrollar tareas como la navegación autónoma (libre de obstáculos, con obstáculos estáticos y con objetos en movimiento) bajo un entorno controlado con luz artificial y usando una superficie similar a una autopista de dos carriles.

Así mismo es relevante desarrollar una metodología propia y de dominio público que contribuya al crecimiento de la investigación de automóviles autónomos a escala en México, tomando como referencia las actividades planteadas por el TMR e implementando los algoritmos sobre el prototipo AutoNOMOS Mini V2. En este sentido, no solamente se desea cumplir los retos, sino que también realizarlos utilizando redes neuronales convolucionales (CNN), cambiando la computadora en el prototipo y además, actualizando el software, con la finalidad de que sea posible emplear CNN, además de que los cálculos que se requieran se cumplan en un menor tiempo que utilizando la computadora con la que ya cuenta, posibilitando que la velocidad del prototipo sea mayor, aunado a lo anterior, la actualización servirá también para poder emplear redes 5G en la comunicación, con la finalidad de mover información entre la computadora del prototipo y otra externa en tiempo real. En este mismo contexto, actualizar la zona de locomoción ayudará en gran medida a realizar la navegación, puesto que se tiene un error de diseño en la transmisión de la velocidad.

Al cumplir las tareas anteriores, no solamente se habrán cumplido las tareas de navegación, sino que también se tendrá al prototipo funcional para poder proponer actividades futuras y que den seguimiento a lo planteado en este proyecto.

1.4. Hipótesis

Utilizando la información obtenida por los sensores del vehículo AutoNOMOS mini V2, es posible desarrollar un sistema de percepción con redes neuronales convolucionales para la navegación autónoma considerando obstáculos estáticos y en movimiento sobre un ambiente controlado, así como cumplir con los retos establecidos por el Torneo Mexicano de Robótica.

1.5. Objetivos

1.5.1. Objetivo general

Diseñar algoritmos de navegación autónoma para el vehículo AutoNOMOS Mini V2 que cumplan con tareas específicas: Navegación autónoma sin obstáculos, con obstáculos estáticos y con obstáculos móviles, usando visión e inteligencia artificial en un entorno controlado.

1.5.2. Objetivos específicos

1. Identificar los trabajos previamente desarrollados en la visión e inteligencia artificial relacionados con la conducción autónoma.

2. Identificar las técnicas y plataformas de desarrollo más importantes en la navegación autónoma.

3. Elegir las técnicas de mayor relevancia en la navegación autónoma, visión artificial que podrían ser de utilidad para este proyecto.

4. Habilitar un sistema de adquisición de datos sobre los sensores del vehículo AutoNOMOS Mini v2, en tiempo real y simulación.

5. Establecer diferentes criterios de pre-procesamiento sobre los datos adquiridos de los sensores del vehículo, que faciliten la creación de las bases de datos.

6. Bajo el enfoque de aprendizaje supervisado, crear tres bases de datos de imágenes que permitan el aprendizaje de diferentes tipos de tareas como la conducción sobre un carril, el rebase y la detección de obstáculos.

7. Diseñar e implementar un sistema de percepción sobre el vehículo que permita la navegación autónoma sobre un ambiente controlado en tiempo real y en simulación, considerando obstáculos estáticos y en movimiento.

1.6. Metas

El desarrollo de este proyecto alimenta la investigación y el desarrollo de la navegación autónoma en los vehículos en México, es importante que el prototipo AutoNOMOS Mini V2 realice vueltas a la pista sin complicaciones, de la misma manera es necesario que evada obstáculos estáticos y obstáculos móviles, así mismo los algoritmos a implementar deben ser lo suficientemente robustos para realizar dichas tareas. Un listado de tópicos importantes a cumplir se mencionan a continuación:

1. Analizar las técnicas de mayor relevancia usados para la navegación autónoma, así como técnicas de visión e inteligencia artificial.
2. Relacionar los componentes del robot AutoNOMOS Mini V2:
   - Analizar los componentes con los que cuenta el prototipo AutoNOMOS mini V2.
   - Analizar los diagramas de las relaciones entre los sensores, actuadores y conexiones.
   - Elaborar un criterio para dar inicio con las pruebas en el orden de menor complejidad a mayor complejidad.
3. Realizar procesamiento de imágenes de la cámara del robot AutoNOMOS Mini V2 y elegir un área de interés en la visión del robot.
4. Crear tres bases de datos con las imágenes obtenidas y ángulo de dirección del prototipo de acuerdo con el área de interés elegida, así también como la información del sensor LiDAR.
5. Diseñar y emplear redes neuronales convolucionales para la predicción de movimiento del AutoNOMOS Mini v2 en un entorno controlado con obstáculos estáticos y dinámicos, y obtener buenos resultados en su ejecución (que el prototipo no se salga de su carril y evada obstáculos sin chocar).
6. Obtener y elegir las redes neuronales convolucionales con sus hiperparámetros que obtengan resultados favorables y resuelvan las tareas de navegación autónoma del vehículo AutoNOMOS Mini V2 en un entorno controlado.

1.7. Alcances y limitaciones de la tesis

Los alcances y limitaciones que enfrenta este proyecto se listan a continuación:

- El prototipo AutoNOMOS Mini v2 estará limitado a desplazarse dentro de una pista, la cual se encuentra trazada en el edificio de la División de Estudios de Posgrado de la Universidad Tecnológica de la Mixteca, el cual es un circuito con fondo negro y señalamientos blancos, con y sin obstáculos, ya sean estáticos o móviles. Dentro de las limitantes de proyecto, las siguientes tareas no serán consideradas:
  - Responder ante señalamientos de transito como lo son: Reducir la velocidad, cruce de peatones, cambios de carril, etc.
  - Realizar recorridos largos que excedan los límites de velocidad y autonomía de la batería.
  - Responder ante una situación de crucero con semáforos y tráfico.
  - Maniobras de estacionamiento.
  - La tercer tarea, en la que la evasión se realiza con un obstáculo móvil, únicamente se realizará en simulación.
- El prototipo estará limitado a ciertas condiciones durante su trayectoria en la pista.
  - El prototipo busca no salirse del perímetro que forma el carril, sin importar que este no vaya centrado.
  - Avance del vehículo a velocidad constante.
  - El movimiento resultante está sujeto al manejo y control del prototipo durante la creación de las bases de datos.
  - El algoritmo considerará obstáculos únicamente en el carril del vehículo, por lo que comenzará a evadir al encontrar el obstáculo, lo rodeará en el carril contiguo, se reincorporará, y seguirá con su camino.

Capítulo 2

Marco teórico

2.1. Plataforma AutoNOMOS Mini V2

Como lo hace notar (AutoNOMOS, 2017) , describe la plataforma de desarrollo AutoNOMOS Mini V2 el cual es un modelo vehicular a escala 1:10 diseñado con propósitos educativos en la Universidad Libre de Berlín, el sistema de percepción se enlista a continuación y las Figuras 2.1 y 2.2 representan su locación en el modelo:

Componente	Información
Computadora Odroid XU4 de 64 GB	Computadora principal.
Plataforma Arduino NANO	Protocolo I2C, PWM, control de batería.
Módulo usb-serial FT4232H	Para la comunicación.
Servomotor XciteRC KLS-19s	Motor para la dirección.
Motor Faulhauber sin escobillas	Motor para la tracción.
Sensor IMU MPU5060 de 6 ejes	Orientación actual del robot.
Cámara Intel 3D SR300	Visión.
Escáner RP-LIDAR 360s	Visión.
Batería SLS X-Cube 14.5v	5000mAh de autonomía.

Tabla 2.1: Componentes de importancia en el AutoNOMOS mini V2

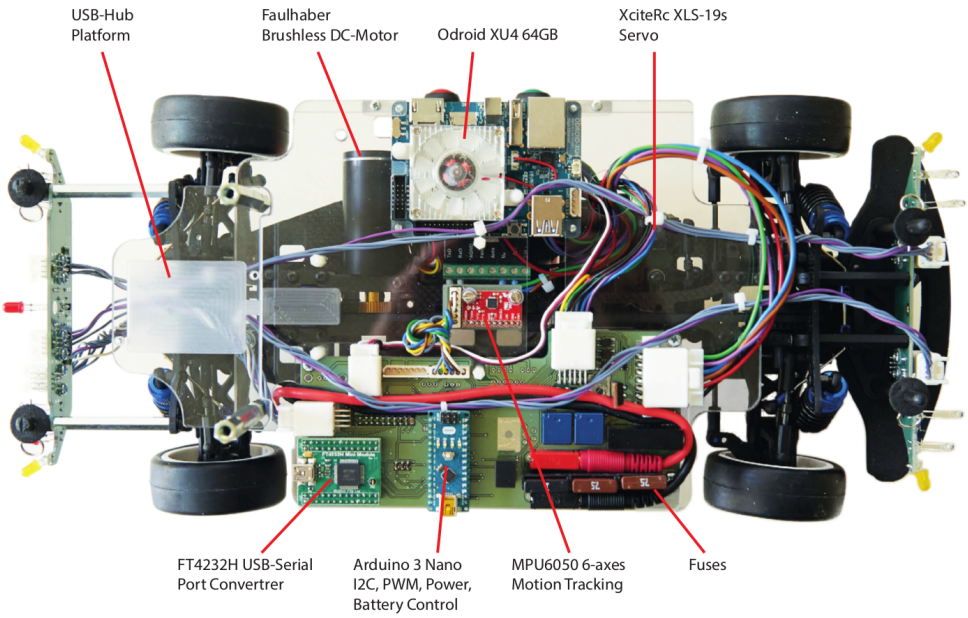


Figura 2.1: Vista superior del AutoNOMOS Mini V2. Fuente: (AutoNOMOS, 2017).

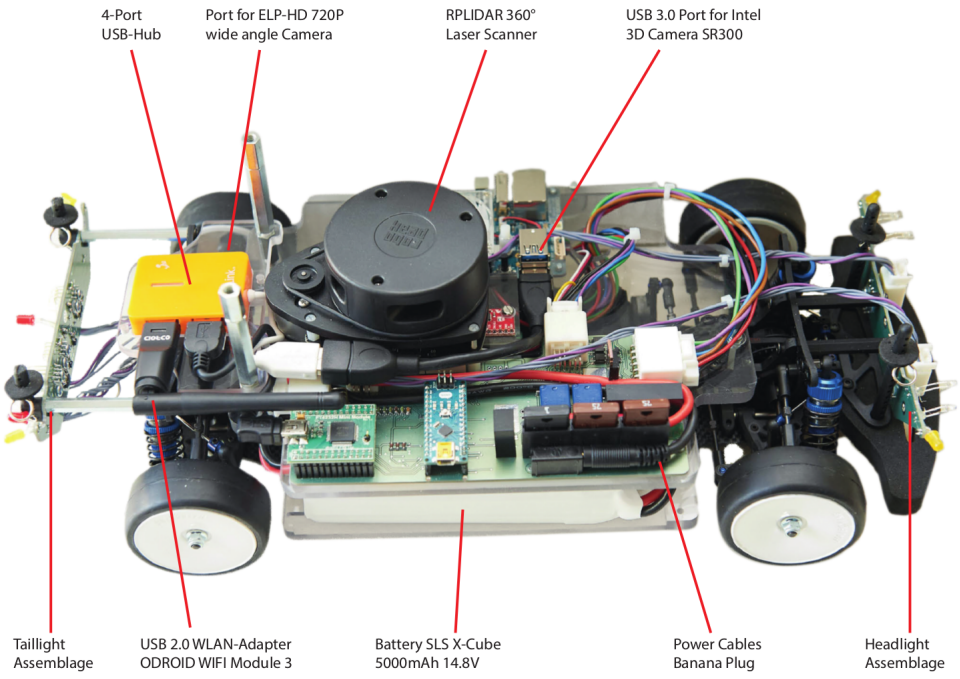


Figura 2.2: Vista lateral del AutoNOMOS Mini V2. Fuente: (AutoNOMOS, 2017)

En cuanto al software, este prototipo utiliza el sistema operativo Linux para los procesos (en la computadora Odroid), mientas que Ubuntu es la distribución empleada en la computadora, la versión con la que el prototipo cuenta es la 14.04, que incluye principalmente software libre y de código abierto, y en este caso, ya cuenta con las librerías y los programas necesarios para tener a todos los componentes comunicados entre sí. Y para este caso, utiliza ROS principalmente como fuente de comunicación.

2.2. Probabilidad

Según (Allen, 2012), la probabilidad representa un grado de creencia en un hecho o predicción. Una probabilidad muy alta representa la certeza de que un hecho es cierto, o que una predicción se hará realidad. Un valor de probabilidad bajo representa certeza de que el hecho es falso. Los valores intermedios representan que un resultado predicho es tan probable que suceda como de que no. Comúnmente se representa en una escala de 0 a 1 o de 0 % a 100 %. Por otro lado, un problema de regresión se basa en resultados muy dinámicos en sus pruebas, ya que se busca ajustar y encontrar una relación entre los datos a los que se desee ajustar, de manera que al no ser completamente exacto el resultado, se desea que la probabilidad de acercarse al resultado verdadero sea lo más alta posible.

2.3. Aprendizaje máquina

De acuerdo con (Sandoval, 2018), el aprendizaje máquina es una rama de la inteligencia artificial, donde se busca generar algoritmos los cuales tienen la capacidad de aprender sin tener que programarse de manera explícita. Sin embargo, para que esto suceda, hace falta proveer de información al algoritmo con datos necesarios para cumplir las tareas que se desean, evidentemente, la carga de información será proporcional a la capacidad de aprendizaje. Existen dos tipos de aprendizajes según (Calvo et al., 2018):

- Aprendizaje supervisado: La información usada para desarrollar el algoritmo contiene información sobre las características de lo estudiado. Es decir, la información requerida y que se desea conocer se encuentra implícitamente dentro de la información suministrada para crear el modelo. De acuerdo con (Roman, 2019), existen dos principales

aplicaciones de aprendizaje supervisado:

- Clasificación: La clasificación es una categoría dentro del aprendizaje supervisado donde el objetivo es predecir y clasificar valores discretos de acuerdo a las clases categóricas que puedan existir. Un ejemplo sencillo de clasificación binaria se puede observar en la Figura 2.3, donde se tiene dos clases de objetos, dos características de dichos objetos (x_1 y x_2). Una vez se tiene cada punto de datos con su clase, como resultado se tiene una línea divisora y cada que se ingrese un nuevo dato, se clasificará en alguna clase de acuerdo a esta línea.

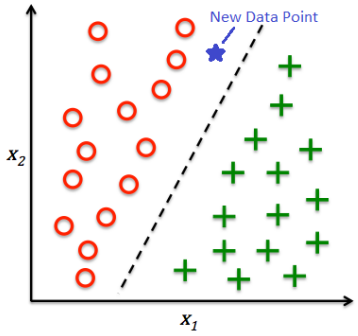


Figura 2.3: Ejemplo de clasificación binaria. Fuente: (Roman, 2019)

- Regresión: Es usada para asignar categorías a datos que no se encuentran etiquetados. En esta categoría se tiene un número de variables predictoras o explicativas y una variable de respuesta continua o resultado, se desea encontrar relación entre las variables para hallar un resultado continuo. Como ejemplo, la Figura 2.4 ilustra una regresión lineal donde se establece una línea recta que minimiza la distancia entre los puntos de muestra y la línea ajustada. Por último, se utilizan las desviaciones obtenidas en la formación de la línea para predecir nuevos datos de salida.

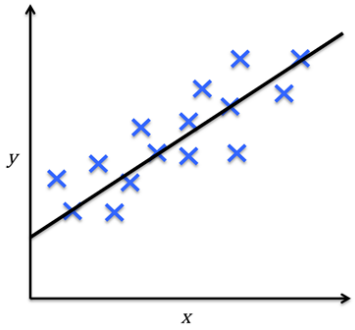


Figura 2.4: Ejemplo de regresión lineal. Fuente: (Roman, 2019)

- Aprendizaje no supervisado: A diferencia del aprendizaje supervisado, este aprendizaje no cuenta con la información de la variable a predecir, en este caso, el objetivo se centra en encontrar patrones o relaciones en los datos. La principal categoría es el agrupamiento.

  - Agrupamiento: También conocido como clustering, y tal como señala (Moya, 2016), es un método que consiste en agrupar un conjunto de objetos que no están etiquetados en subconjuntos de objetos denominados Clusters. Cada Cluster está formado por un conjunto de objetos que son similares entre sí, por lo tanto son distintos respecto a los objetos de otros Clusters.

    Se puede partir de una hipótesis, como por ejemplo algunos tipos de cabello, donde x es el color: $f(x) = Moreno, Rubio, Castaño, Canoso$ seguido se obtiene información sobre la estructura o dominio de salida en base a los datos del color de cabello que se tiene en el data set. Para ello se debe utilizar alguna técnica de Clustering e indicar que el agrupamiento será en 4 Clusters. Una vez finalizada la técnica de Clusterización se podrá identificar los individuos que son Morenos, Rubios, Castaños o Canosos observando el valor medio (o Centroide) de cada Cluster. La Figura 2.5 corresponde al modelo gráfico de un problema parecido.

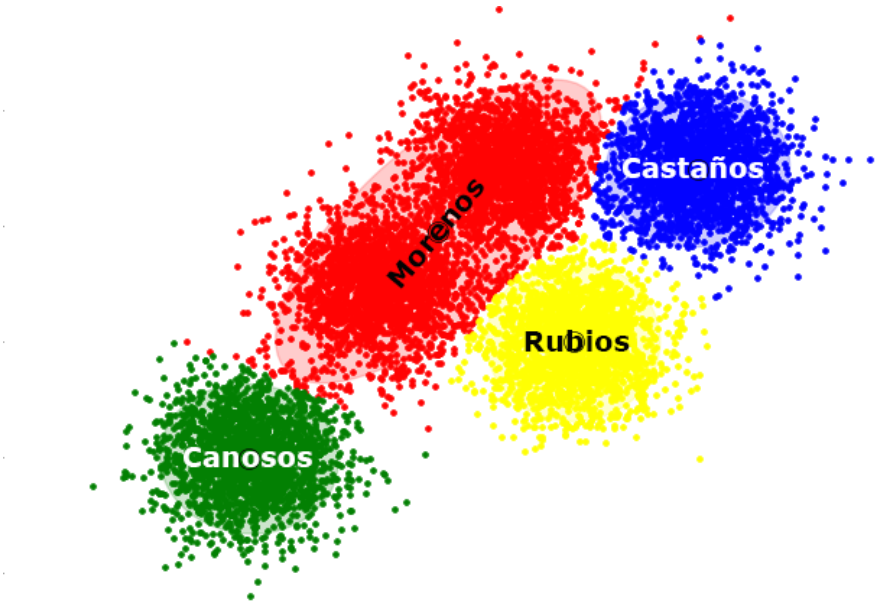


Figura 2.5: Ejemplo de clustering con 4 grupos. Fuente: (Moya, 2016)

2.3.1. Redes neuronales

Según lo descrito en (Hagan et al., s.f.), las redes neuronales artificiales están altamente relacionadas con las redes neuronales biológicas, por lo tanto, dichas redes artificiales son inspiración de las redes biológicas y es por tal motivo que cuentan con los mismos nombres identificadores, la Figura 2.6 muestra detalladamente una neurona y sus partes más importantes usadas en una neurona artificial son:

- Dendrita: Las dendritas son redes de fibra nerviosa en forma de árboles que transportan señales eléctricas al cuerpo celular. El cuerpo celular suma dichas señales y establece un umbral.
- Axón: El axón es la parte de fibra más larga de una neurona que transporta señales del cuerpo celular a otras neuronas. El contacto entre el axón de una neurona y la dendrita de otra neurona se denomina sinapsis.

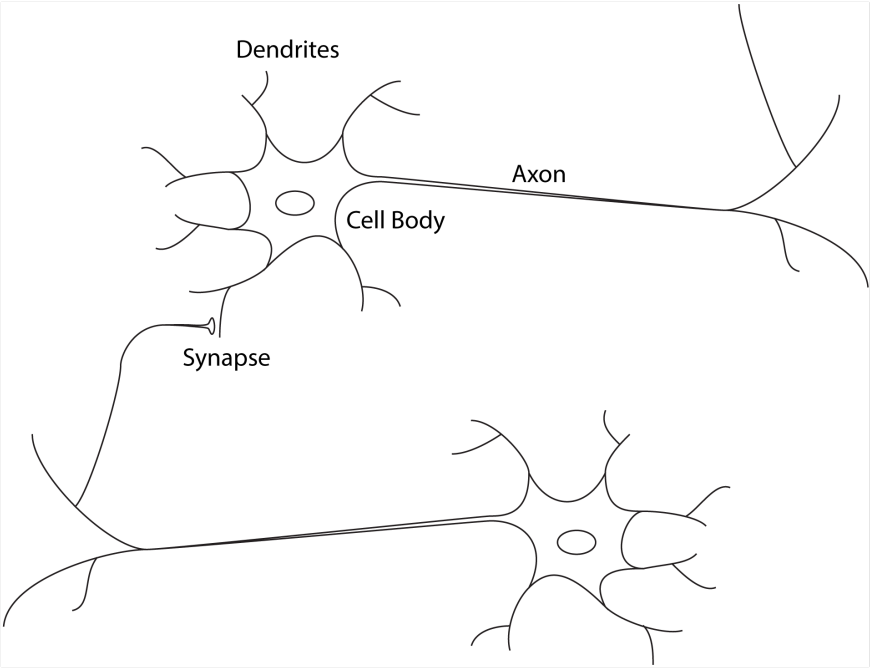


Figura 2.6: Partes de una neurona biológica. Fuente: (Hagan et al., s.f.)

Tal como señala (Gershenson, 2012), al modelar la neurona en una red neuronal artificial, su complejidad se abstrae en gran medida. La abstracción se generaliza en entradas (quienes simulan la sinapsis) y que se multiplican por pesos, seguido de una función matemática la cual

determina si la neurona se activa, y por último la salida, quién se conecta con otras entradas de otras neuronas formando la red neuronal, una de las representaciones más acertadas se observa en la Figura 2.7.

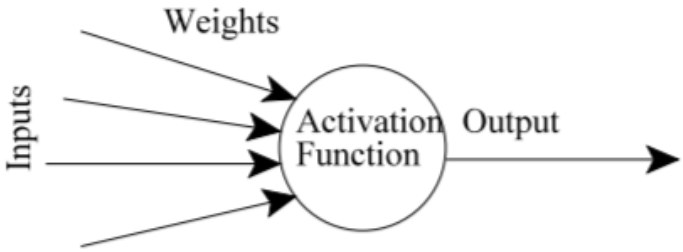


Figura 2.7: Partes de una neurona artificial. Fuente: (Gershenson, 2012)

2.3.1.1. Perceptrón multicapa

Reafirmando lo mencionado anteriormente y citando a (Popescu et al., 2009), el perceptrón multicapa es la red neuronal más conocida y utilizada en la práctica. En este caso las señales se transmiten dentro de la red en un solo sentido: de entrada a salida. No hay bucle, la salida de cada neurona no afecta a la propia neurona. Esta arquitectura se conoce como feed-forward, la representación que se le da a esta red puede observarse en la Figura 2.8. Las capas que no están directamente conectadas con las entradas o salidas se llaman capas ocultas.

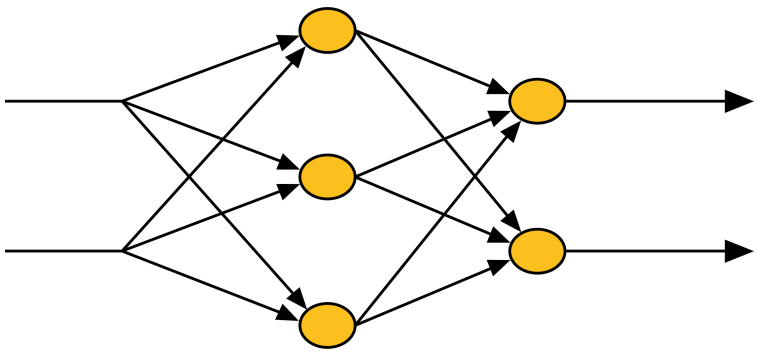


Figura 2.8: Representación de un ejemplo de red neuronal Feed-forward. Fuente: (Popescu et al., 2009)

Un perceptrón con una sola capa y una entrada genera regiones de decisión bajo la forma de semiplanos. Al agregar otra capa, cada neurona actúa como un perceptrón estándar para las salidas de las neuronas de la capa anterior, por lo tanto, la salida de la la red puede estimar regiones de decisión convexas, resultante de la intersección de los semiplanos generada por las neuronas. Del mismo modo, un perceptrón tricapa puede generar áreas de decisión arbitrarias, como lo muestra la Figura 2.9.

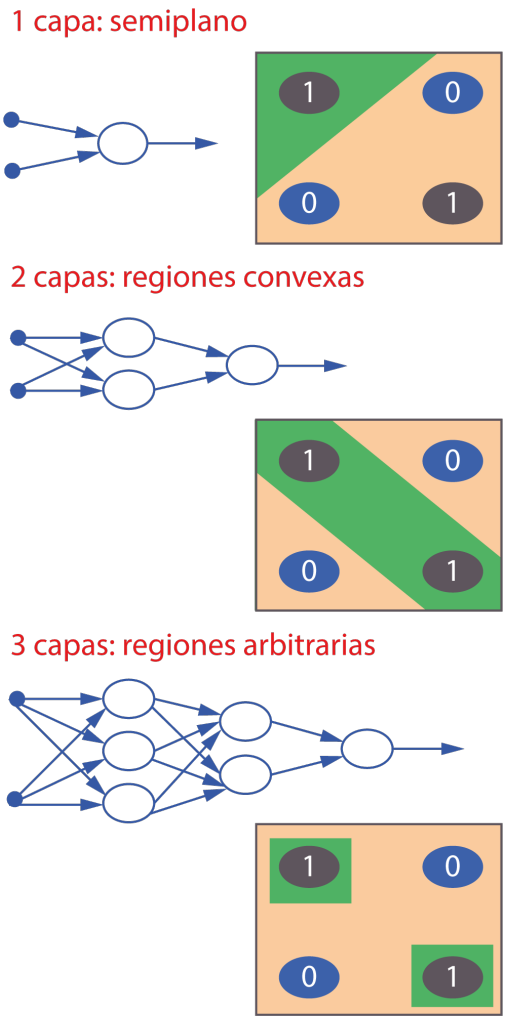


Figura 2.9: Diferencias de usar N capas en un mismo problema XOR. Fuente: (Popescu et al., 2009)

2.3.2. Aprendizaje profundo

El aprendizaje profundo surge como una necesidad debido al error de actividades más complejas con el uso de algoritmos tradicionales (Silva, 2021). Debido a que generalizar ejemplos o muestras se vuelve complejo cuando se tienen datos de dimensiones mas altas. Por lo tanto, el aprendizaje busca modelar abstracciones de datos con altas dimensiones utilizando arquitecturas compuestas de transformaciones no lineales múltiples. En consecuencia, al agregar más capas, la red puede representar funciones con complejidad reciente, y a su vez se genera una red profunda.

2.3.2.1. Red neuronal convolucional

La característica principal de las redes neuronales convolucionales es la aplicación a datos de imágenes, debido a que estos datos pueden ser vistos como cuadrículas bidimensionales y estas redes neuronales se especializan en estas configuraciones de información.

El objetivo entonces, es el resolver un problema sobre lo observable, intentado que este sea resuelto lo más parecido a como lo haría un cerebro humano, evidentemente no al mismo grado de complejidad, pero pueden ser utilizados para problemas especializados.

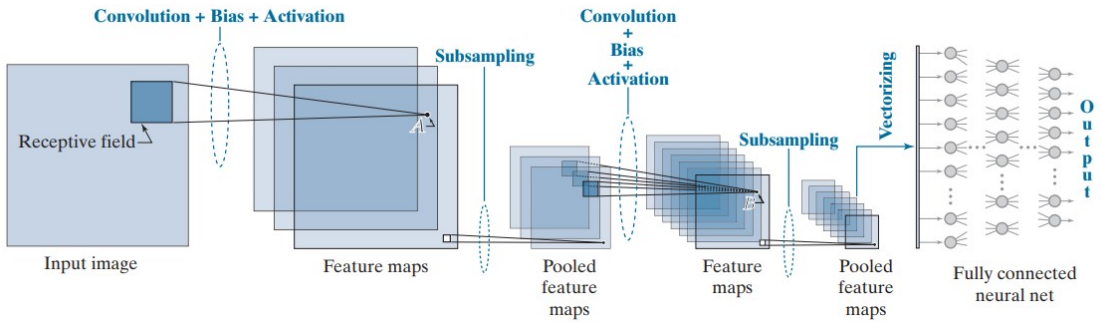


Figura 2.10: Estructura de una red neuronal convolucional (CNN) Fuente: (Gonzalez y Woods, 2018)

La estructura de una red neuronal convolucional se muestra en la Figura 2.10, donde se observa que en la capa convolucional las unidades se organizan en planos,y dichos planos se denominan mapas de características.

La explicación mecánica de la convolución espacial se describe en la ecuación 2.1, donde (x, y) es cualquier punto, ω denota el kernel a utilizar, $a_{x,y}$ denota la imagen o la agrupación de características dependiendo de la capa, y l, k definen las dimensiones del kernel.

$$\begin{aligned} \omega * a_{x,y} &= \sum_{l} \sum_{k} \omega_{l,k} a_{x-l,y-k} \\ &= \omega_{1,1} a_{x-1,y-1} + \omega_{1,2} a_{x-1,y-2} + \cdots \end{aligned} \tag{2.1}$$

Por lo tanto la Ecuación 2.1 puede ser escrita de la siguiente manera (donde n es el tamaño del kernel):

$$\begin{aligned} \omega * a_{x,y} &= \omega_1 a_1 + \omega_2 a_2 + \cdots + \omega_n a_n \\ &= \sum_{i=1}^{n} w_i a_i \end{aligned} \tag{2.2}$$

Y si se agrega el bias y a la ecuación se denota como z, entonces la convolución puede verse como:

$$\omega * z = \sum_{i=1}^{n} w_i a_i + b \tag{2.3}$$

Por último, se debe tomar en cuenta que este resultado es lineal, por lo que a la Ecuación 2.3 se le aplica una función de activación.

Gráficamente, la convolución toma una región de pixeles de la imagen de entrada, la cual se llama campo receptivo, y los valores de la convolución se generan moviendo el campo receptivo sobre la imagen y, en cada posición, se forma una suma de productos del conjunto de pesos del kernel con los píxeles contenidos en el campo receptivo, la Figura 2.11 muestra un ejemplo del resultado de aplicar la convolución en un espacio de la imagen.

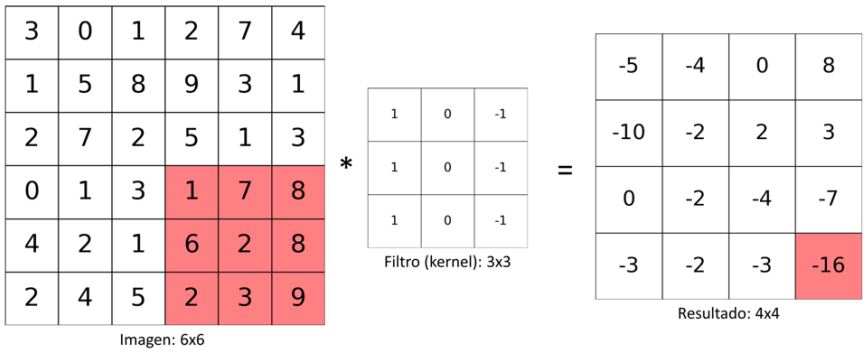


Figura 2.11: Ejemplo del proceso de convolución en imágenes Fuente: (Sotaquirá, 2019)

Siguiendo con la Figura 2.10, se observan 3 mapas de características en la primera capa, y cada mapa fue generado por un kernel para generar diferentes características, el conjunto de estos mapas se llama capa convolucional, y esta red cuenta con 2 capas.

En consecuencia, los filtros pueden verse como identificadores de características, tales como pueden ser bordes, colores, curvas, etc.

Exiten dos parámetros a modificar en una CNN, los cuales son (1) el paso (stride en inglés) y (2) el relleno (padding). El paso controla la manera en que el kernel se va a mover sobre la imagen, lo común es que el paso sea configurado a uno, por otro lado, el relleno agrega con ceros el volumen perdido a causa de la convolución, intentado preservar la mayor información posible.

Lo siguiente dentro de la red es el submuestreo o pooling, y no son más que mapas de características de resolución espacial reducida. Por lo que aplicar pooling a la capa convolucional es un set más pequeño (típicamente de 2x2 pixeles) de regiones. Los métodos más comunes de pooling son: Average pooling, donde los valores en cada vecindad se reemplazan por el promedio de los valores en dicha vecindad, Max pooling, que reemplaza los valores en un vecindario por el valor máximo de sus elementos y por último, L_2 pooling, en el cual el valor combinado resultante es la raíz cuadrada de la suma de los valores de la vecindad al cuadrado. Se debe tomar en cuenta que el uso de este método ayuda al costo computacional, sin embargo el precio que se paga por usarlo se ve reducido en la calidad de las características, puesto que se pierde información, en muchos casos, este paso ya no se realiza. La Figura 2.10 cuenta con 2 procesos de pooling, en donde el segundo cuenta con el menor tamaño posible para ser procesado.

Por último, se deben vectorizar los mapas de características en 2D que se encuentran agrupados en la última capa. Se utiliza la indexación lineal, es decir, cada matriz en la última capa de la CNN pasa a ser un vector, después de concatenarse y formar un vector. Este vector es el que se propaga a través de la red neuronal, mismo que se muestra al final de la Figura 2.10. Para una mejor visualización de todo el proceso, la Figura 2.12 representa un ejemplo para clasificar números utilizando la red neuronal convolucional de la Figura 2.10.

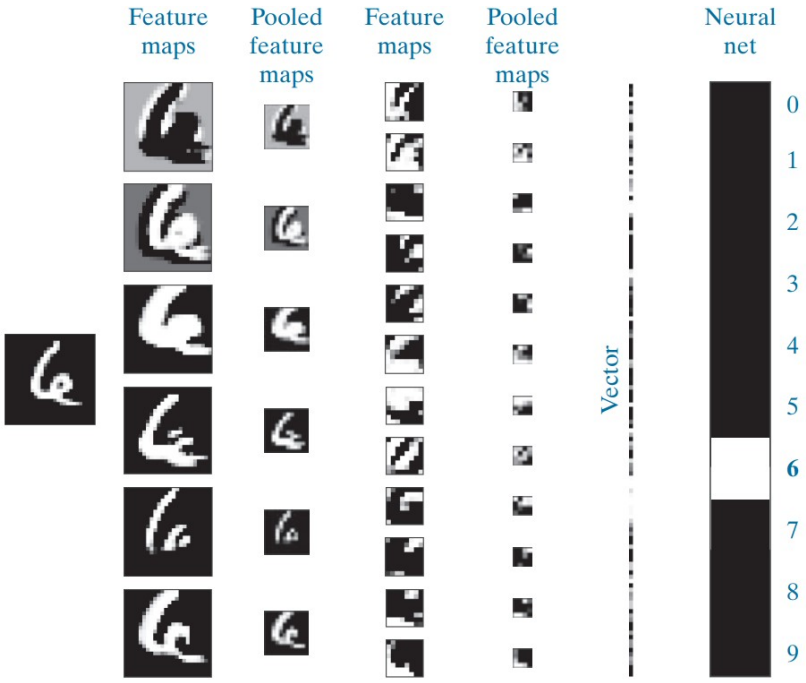


Figura 2.12: Aplicación visual de una red neuronal convolucional (CNN) Fuente: (Gonzalez y Woods, 2018)

2.3.2.2. Hiperparámetros

No existe una metodología escrita sobre cuantas capas utilizar: cuantas convolucionales y cuantas completamente conectadas, así también como el tamaño del kernel, el paso, la función de activación; puesto que la arquitectura de la red dependerá en gran medida de los datos de entrada. Pero lo que sí se puede realizar es analizar el estado del arte y readaptar arquitecturas previamente establecidas y partir de ellas. El controlar toda esta información se le conoce como ajuste de los hiperparámetros, con la finalidad de buscar y encontrar los mejores resultados.

2.3.2.3. Capas ELU

Como ya se ha comentado, después de la capa convolucional y de una capa completamente conectada, se aplica una capa de activación. La razón es implementar la no linealidad

a las operaciones que una CNN ya realiza, puesto que dichas operaciones son lineales. Existen diferentes funciones de activaciones, y cada una de ellas son empleadas para problemas específicos, como por ejemplo la tanh, la sigmoide, o incluso las ReLU, sin embargo, investigadores han encontrado que para los problemas de regresión y más específicamente sobre navegación autónoma las capas ELU (Unidad lineal exponencial, por sus siglas en inglés) funcionan mejor, por la razón de que la CNN es capaz de entrenar mucho más rápido que utilizando otras funciones. La capa ELU puede expresarse matemáticamente de la siguiente forma:

$$f(x) = \begin{cases} x & si \quad x \geq 0 \\ \\ \alpha(e^x - 1) & si \quad x \leq 0 \end{cases} \tag{2.4}$$

Donde α puede adoptar cualquier valor positivo, y para este caso particular se utiliza como $\alpha = 1$, de este modo, se observa que para valores negativos la salida se encuentra entre -1 y 0, mientras que para valores positivos la salida toma el mismo valor de entrada. La Figura 2.13 muestra gráficamente esta función.

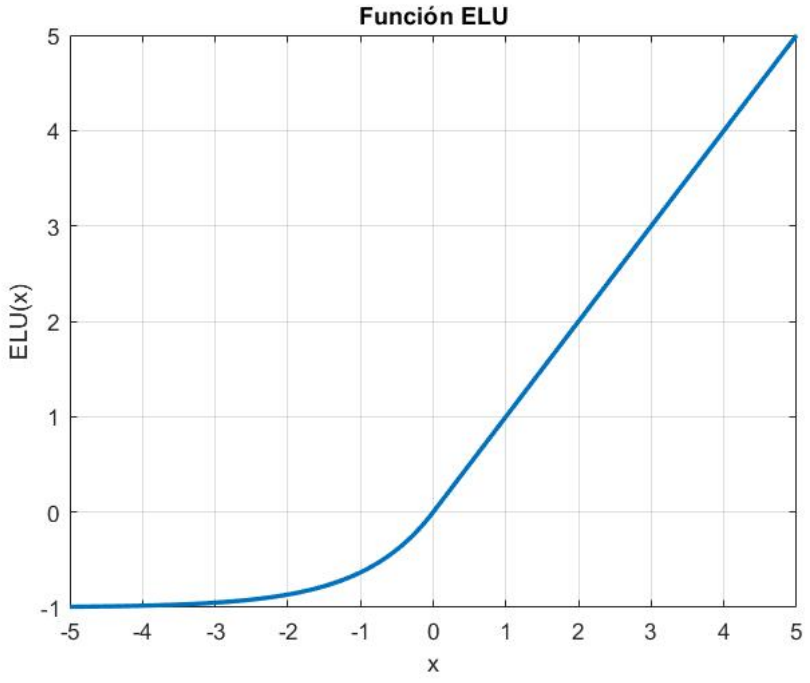


Figura 2.13: Función de activación ELU

2.3.2.4. Entrenamiento

En entrenamiento es la manera en la cual una CNN aprende, más específicamente con un método llamado backpropagation (propagación hacia atrás).

Cuando se genera una CNN, lo primero que se inicializan son los pesos aleatoriamente. Es decir, los kernel, el bias, los valores con los que está conectada cada neurona tiene valores al azar. Por lo tanto, se puede señalar al proceso de entrenamiento en cuatro etapas importantes: 1) el paso hacia adelante, 2) cálculo de la función de costo, 3) paso hacia atrás, y por último, 4) actualización de los pesos. Este proceso se lleva a cabo de manera iterativa y cíclica:

1. **Paso hacia adelante.** Se toma una imagen del conjunto de entrenamiento y se pasa por toda la CNN (tomando en cuenta que los pesos poseen valores aleatorios), lo común es que el resultado obtenido no sea el esperado, sin embargo, dado que es un problema de aprendizaje supervisado, es conocido lo que debería tener como resultado la CNN. En ese sentido, la función de costo juega un papel muy importante para la evaluación.

2. **Función de costo.** Existen muchas funciones de costo, sin embargo, para problemas de regresión, en la literatura se prefiere utilizar el MSE (Error cuadrático medio, por sus siglas en inglés), que está dado como:

$$L = MSE = \frac{1}{n}\sum_{i=1^n}^{n} (valor_{deseado} - valor_{predicho})^2 \tag{2.5}$$

   Donde n es el número total de datos evaluados, i corresponde a cada uno de los datos contenidos en n, el $valor_{deseado}$ es lo que se desea que la CNN prediga y $valor_{predicho}$ es lo que la CNN a través del método hacia adelante predijo. La Ecuación 2.5 puede ser reescrita L, donde L no es más que la función de costo.

3. **Paso hacia atrás.** El objetivo del paso hacia atrás es basarse en el error para intentar disminuirlo, gráficamente este problema puede ser visto como una partícula en el espacio, la cual se desea que esta partícula llegue al fondo de un espacio cóncavo (un mínimo), y así reduciendo el error, por lo tanto, matemáticamente puede ser visto como dL/dW, donde W son los pesos en una capa específica y L la función de costo. Como es sabido, la derivada puede observarse como la dirección, en este caso de la partícula, la cual apunta hacia afuera, sin embargo, se desea que este decrezca.

4. **Actualización de los pesos.** Una vez calculada la derivada anterior, se actualizan los pesos de cada capa en el mismo paso hacia atrás, la expresión siguiente muestra el peso actualizado:

$$W = W_i - \eta \frac{dL}{dW} \tag{2.6}$$

Donde W es l nuevo peso, W_i es el peso inicial anterior y η el factor de aprendizaje (usualmente este factor es pequeño).

El proceso descrito se denomina iteración, por lo que las iteraciones se repetirán dependiendo del criterio tomado por el programador. Una vez entrenada la CNN, deberá ser capás de evaluar imágenes nuevas y obtenerse el resultado que se desea.

2.3.2.5. Pruebas

Para comprobar el buen funcionamiento de esta CNN, puesto que el proyecto es un problema de regresión, el modelo se evalúa con imágenes nuevas y el prototipo AutoNOMOS funcionando, por lo que se esperaría que el robot realice lo que se desea.

2.4. Visión artificial

En la opinión de (Cognex, 2018), la visión artificial incluye las aplicaciones ya sea industriales y no industriales, como puede ser el ámbito académico, en donde se combina las secciones de hardware y software para realizar una actividad, en el caso más común, la captura y procesamiento de la imagen. Evidentemente, la robustez tanto del hardware como del software definirá las limitantes de la ejecución en sus funciones. Los sistemas de visión artificial se basan en sensores digitales protegidos dentro de cámaras con óptica especializada para adquirir imágenes, de modo que se pueda procesar, analizar, medir y ejecutar una toma de decisiones.

En el procesamiento de imágenes se utiliza información pictórica para un mejor manejo de los datos que se desean obtener de las imágenes. Dicho de otra manera y en palabras de (Valentín, 2018), manipula la información que se tiene en las imágenes para eliminar el ruido o cualquier irregularidad que no se desea captar y analizar, estas imágenes son provenientes de

dispositivos de la vida cotidiana como cámaras. Dentro del análisis de imágenes una imagen se puede definir como una función bidimensional $f(x, y)$ donde x, y son las coordenadas y el valor que toma f en cualquier coordenada se denomina intensidad o nivel de gris en ese punto.

(Gonzalez y Woods, 2019) describe en la Figura 2.14, la generación de una imagen, se observa una fuente de energía, la cual es reflejada por un elemento del escenario, seguido a esto se tiene el sistema de adquisición de la imagen, la cual recolecta la energía y la enfoca en un plano de imagen, la lente que cuenta el sistema de adquisición de imagen produce salidas proporcionales a la intensidad de la luz recibida en cada sensor. Los circuitos digitales y analógicos barren estas salidas y la convierten a una señal de video, que luego es digitalizada en otra sección del sistema de imagen. Por lo tanto, salida es una imagen digital.

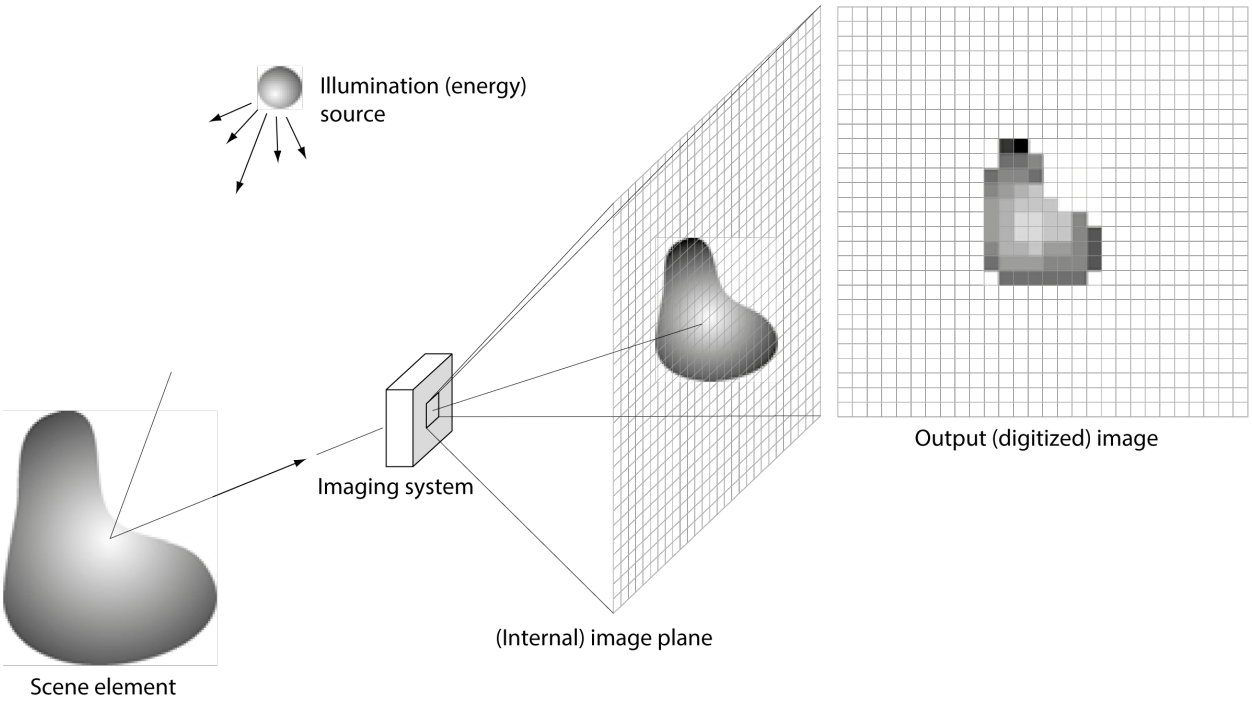


Figura 2.14: Proceso para la generación de una imagen. Fuente: (Gonzalez y Woods, 2019)

Dicho lo anterior y citando a (González et al., 2006), se reafirma que en el proceso para la creación de una imagen consiste básicamente en lo siguiente: el objeto, la fuente radiante (iluminación), y el sistema de formación de imagen el cual está formado por: un sistema óptico, un sensor y un digitalizador. La imagen puede ser representada por una matriz f de dimensiones NxM vista en la ecuación 2.7, donde cada pixel es un elemento de intensidad.

$$f = \begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,M) \\ f(2,1) & f(2,2) & \cdots & f(2,M) \\ \vdots & \vdots & \ddots & \vdots \\ f(N,1) & f(N,2) & \cdots & f(N,M) \end{bmatrix} \tag{2.7}$$

2.5. OpenCV

OpenCV es una librería dirigida fundamentalmente a la visión por computadora en tiempo real según lo detalla (Arevalo et al., 2005b), quién además explica que únicamente OpenCV proporciona las bibliotecas de tipos de datos estáticos y dinámicos (Matrices, grafos, etc.), por otra parte también cuenta con un amplio repertorio de herramientas para trabajar con la mayoría de capturadoras/cámaras que se emplean para estos ejercicios, de la misma manera cuenta con un entorno fácil e intuitivo y se encuentra disponible para Linux. Los elementos de alto nivel que proporciona OpenCV se observan en la Figura 2.15.

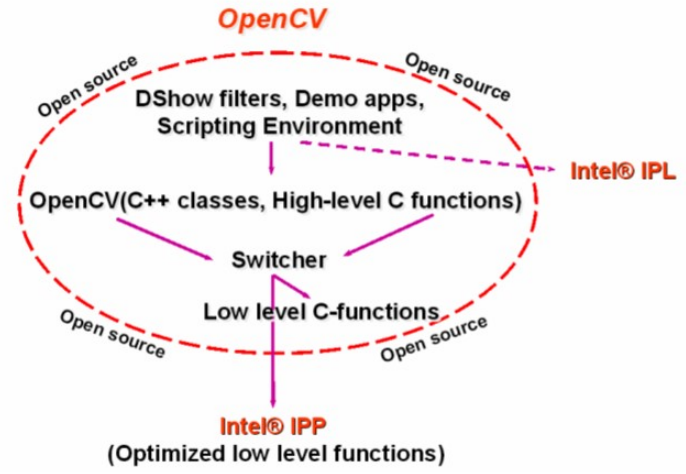


Figura 2.15: Estructura de la librería OpenCV. Fuente: (Arevalo et al., 2005a)

Según (Arevalo et al., 2005a), la librería OpenCV proporciona numerosas aplicaciones para emplear las distintas funciones de la librería.

2.6. ROS (Robot Operating System)

De acuerdo con (Riva, 2021), ROS (Robot Operating System, por sus siglas en inglés) proporciona las librerías y herramientas necesarias para asistir a los desarrolladores de software a crear aplicaciones para robots. Por lo tanto, ROS provee abstracción de hardware que facilita una plataforma consistente sobre la cual corren las aplicaciones, de igual manera cuenta con controladores de dispositivos para conectarse fácilmente entre otros dispositivos o computadoras, librerías para agilizar el uso de programación al proveer funcionalidades comunes en el manejo de robots, también cuenta con herramientas de visualización para observar gráficamente conexiones, mensajes, estructuras de datos, así como un simulador llamado Gazebo, la comunicación se da por mensajes y la administración de paquetes es sencilla de crear y configurar.

En palabras de (Ortego, 2017), ROS está basado en una arquitectura de grafos, donde el procesamiento se encuentra en los nodos de la red, quienes pueden recibir, mandar y multiplexar mensajes de sensores, control, estados, planificaciones, y actuaciones a la frecuencia que se desee.

ROS cuenta con varias distribuciones, entre ellas se encuentra ROS Melodic Morenia que es la doceava distribución lanzada y una de las más estables, su desarrollo es principalmente usado sobre Ubuntu 18.04 para computadoras x64 bits, así también como dispositivos ARM de 64 bits (Acorn RISC Machines, por sus siglas en inglés), que no son más que computadoras con reducción de instrucciones.

La Figura 2.16 representa en un esquema las partes más importantes que conforman la comunicación en ROS, así como la relación que se genera entre estas zonas, a continuación se detallan estos conceptos:

- Roscore: Es la suma del ROS Master, rosout y un servidor de parámetros, es decir, lleva registro de los mensajes y los tipos de mensaje que se realizan cuando ROS es ejecutado.
- Nodo: Un nodo de ROS es un ejecutable que genera o recibe información en forma de mensajes. Esta estructura es importante ya que ayuda a a disminuir las fallas debido a que cada nodo es independiente de los demás nodos, por lo que si alguno deja de funcionar, los demás nodos siguen creando mensajes o están a la espera del siguiente mensaje.

- ROS Master: Este nodo proporciona los servicios de registro y nombre al resto de nodos en el sistema, todos los nodos publicadores y suscriptores se registran al nodo maestro, también proporciona el servidor de parámetros.
- Tópico: Son los buses de datos usados por los nodos para enviar mensajes a otros nodos. El mensaje viaja de un nodo publicador (Publisher node) a un nodo suscriptor (Subscriber node) a cierta frecuencia que se decide en ambos nodos, a que frecuencia se envía el mensaje, y por consiguiente.
- Mensaje: El mensaje es enviado por el nodo publicador a través del tópico (creado en este mismo nodo publicador), el nodo suscriptor elige a que nodo o nodos suscribirse.

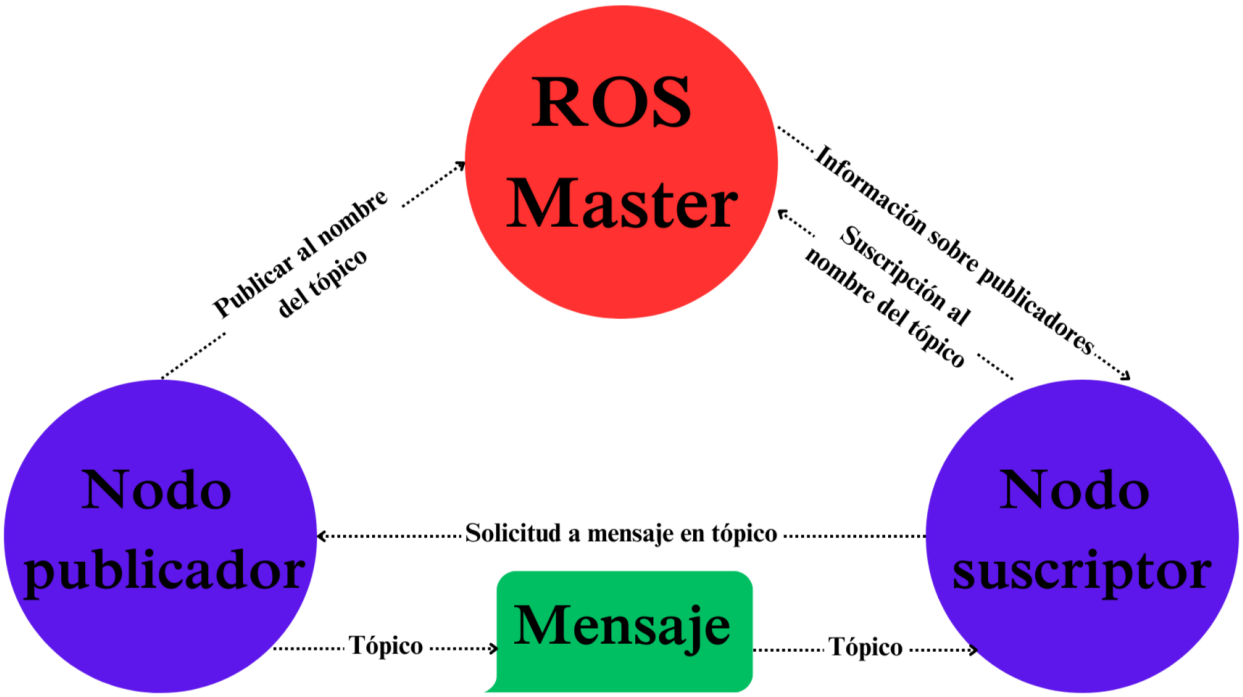


Figura 2.16: Partes más importantes de la comunicación en ROS. Fuente: (Robert, 2020)

2.6.1. Simulador Gazebo

Gazebo es un simulador con una mesa de trabajo que cuenta con librerías y un servicio en la nube, en él se pueden crear diseños físicos y entornos realistas con una alta fidelidad para el uso de sensores. Este simulador forma parte de las librerías de ROS, por lo que crear nodos y utilizarlos en Gazebo para el manejo de mensajes es una tarea sencilla, por lo tanto,

busca que la operación de los robots de manera simulada e implementada de manera real, sea lo más parecido posible.

2.6.2. Gráfico rqt

El gráfico rqt es una librería que proporciona ROS para visualizar gráficos, en ese sentido, rqt tiene la capacidad de proporcionar gráficamente la topología que se genera al hacer uso de ROS, por lo que se mostrará en un gráfico las conexiones que se generen entre nodos, así como tópicos creados, frecuencias de actualización y tipos de mensaje. Es útil en casos en los que la abstracción de la información sea muy compleja.

2.6.3. Bags

Un bag es un formato de archivo utilizado en ROS para almacenar mensajes con datos (la extensión de este archivo es .bag). Los bags son creados a partir de la palabra reservada rosbag, el cual se suscribe a uno o más tópicos de ROS y almacena los datos de los mensajes en un archivo (.bag) conforme son recibidos. Estos archivos también pueden ser reproducidos en ROS con los mismos tópicos que fueron grabados, o incluso puede ser reasignado a tópicos nuevos. La herramienta de rosbag también incluye una opción para publicar un reloj simulado que corresponde a la hora en la que se registran los datos en el archivo, por lo que el manejo de los archivos resulta sencillo si se desea emplear dos o más tópicos al mismo tiempo.

Los bags son el mecanismo principal en ROS para el registro de datos, lo que significa que tienen una variedad de usos: Se pueden registrar conjuntos de datos para luego visualizarlos, además de ser etiquetados y almacenados, también puede ser utilizado para realizar registros de diagnóstico de hardware a largo plazo, las herramientas como rqt_bag permiten visualizar los datos en el bag, así como el trazado de gráficas y la visualización de imágenes, también se puede inspeccionar rápidamente los datos del archivo bag desde la consola (rostopic list), el formato de archivo bag es muy eficiente tanto para la grabación como para la reproducción, ya que los mensajes se almacenan en la misma representación utilizada en la capa de transporte de red de ROS.

Capítulo 3

Metodología

En esta investigación se pretende desarrollar e implementar algoritmos de navegación autónoma tomando como referencia la investigación del Dr. Sampieri (descrito en (Hernández et al., 2010)), a continuación se muestran los pasos a seguir para la realización de este proyecto:

1. Identificación: Analizar y comprender el funcionamiento de cada componente en el AutoNOMOS Mini V2, así como la interconexión entre ellos y la utilidad de su uso en este proyecto.

2. Supervisión y actualización de hardware: La supervisión consta de identificar el buen funcionamiento de cada componente, así como la adaptación de la posición de cada sensor y actuador, es decir, la posición de estas piezas se colocarán en función a las necesidades requeridas. Del mismo modo, si un componente no funcionara, reparar o cambiar.

3. Actualización de software: Adaptar el sistema para que las CNN sean posibles implementar.

4. Adquisición: La adquisición comprende de las técnicas a implementar para la adquisición de imágenes e información del sensor LiDAR a través de los sensores y cámara de las que dispone el prototipo. Para ello es necesario colocar la cámara en una posición adecuada para obtener la mejor resolución y eliminar el mayor ruido posible de la imagen.

5. Procesamiento: Tiene por objetivo el procesamiento de los datos de entrada a fin de identificar aquellos que tengan mayor relevancia para la realización de las tareas propuestas.

6. Base de datos: Bajo el enfoque de aprendizaje supervisado se pretende crear cinco bases de datos (3 para simulación, 2 para implementación real) de imágenes etiquetadas que permitan el aprendizaje de los diferentes tipos de tareas propuestos.

7. Entrenamiento: Las bases de datos serán utilizadas para entrenar diferentes modelos basados en Redes Neuronales Convolucionales (CNN) especializadas en tres condiciones de navegación:

   - Navegación autónoma sin obstáculos: Se utilizará como base la CNN propuesta por NVIDIA en (Bojarski et al., 2016).
   - Navegación autónoma con obstáculos: Se utilizará como base la CNN propuesta para la tarea uno.
   - Navegación autónoma con obstáculos móviles: Se utilizará como base la CNN propuesta para la tarea dos.

8. Actuación: Una vez obtenida una respuesta satisfactoria de los modelos entrenados, se procederá a su implementación sobre el vehículo. A fin de evaluar su rendimiento en el prototipo simulado y real. En el caso de implementación real, la actuación se realizará por medio de una segunda computadora la cual contiene el modelo CNN de acuerdo a la tarea a cumplir, por lo que el robot enviará a través de una red 5G la imagen, la computadora externa decidirá el ángulo de dirección y este será devuelto al prototipo.

9. Observación: Es la supervisión de los procesos anteriores, en especial la actuación, por lo que hace falta observar el comportamiento del vehículo ante las 3 tareas a cumplir y determinar si se han cumplido los objetivos, en caso contrario se deberá regresar al paso **7** y considerar realizar cambios en los hiperparámetros, y evaluar las gráficas del error con lo obtenido anteriormente.

10. Validación: Si los resultados en las tres tareas resuelven los problemas planteados, solo en ese caso el prototipo ha cumplido con lo planteado y las CNN implementadas son correctas.

La Figura 3.1 muestra lo antes descrito de manera gráfica, en ella se observan todos los puntos. Después del paso cinco, se divide en dos los pasos, cuando no hay CNNs entrenadas se crean las bases de datos, se entrenan, se observan los errores, y vuelven al paso cinco, donde ahora se prueban en el robot y se observa si este cumple con sus tareas, por lo que la metodología se vuelve un proceso iterativo.

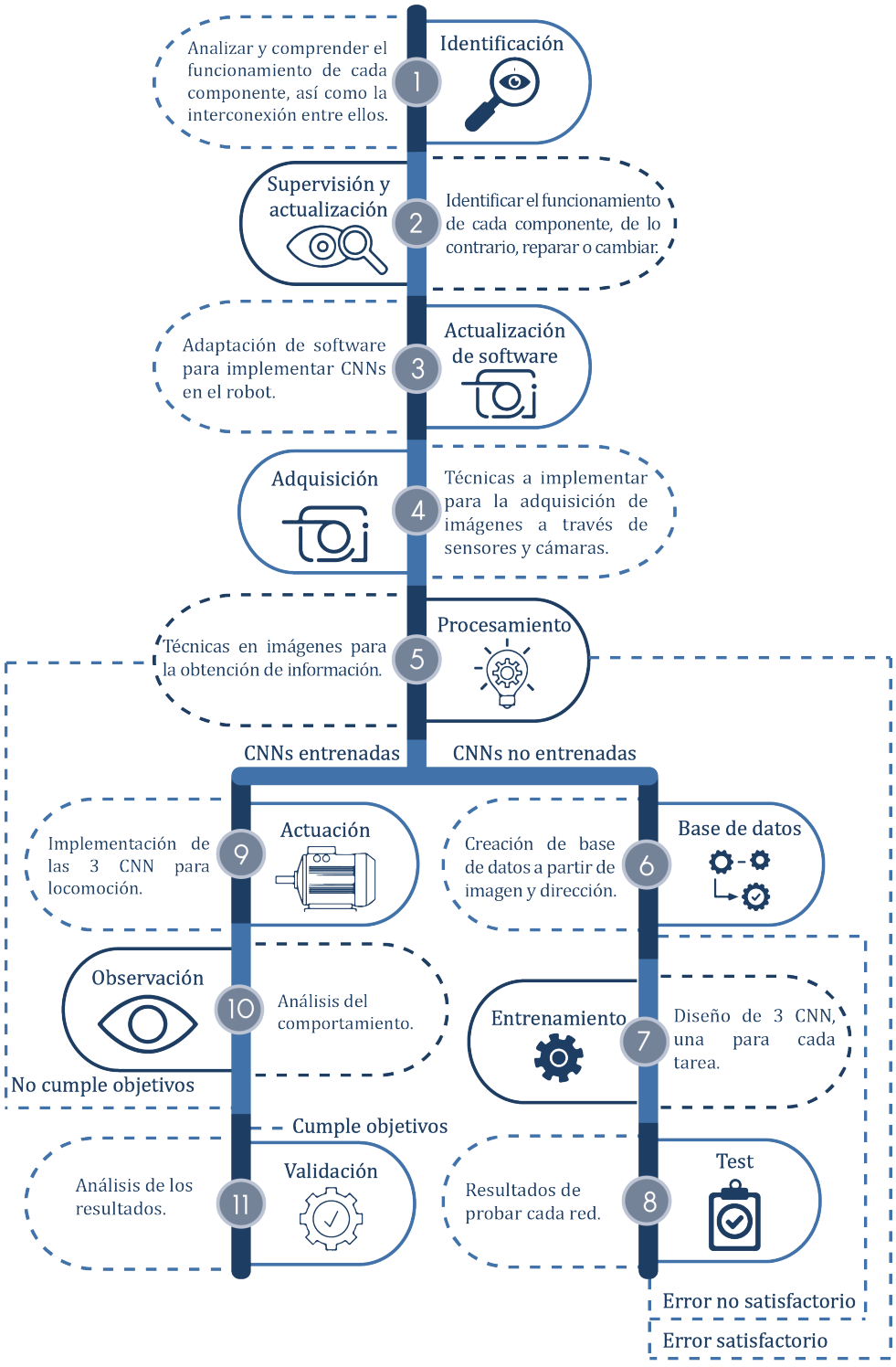


Figura 3.1: Metodología.

Capítulo 4

Desarrollo

En este capítulo se presenta el desarrollo de los algoritmos para el movimiento autónomo de la plataforma AutoNOMOS Mini V2. La información del proceso se divide y describe en tres secciones:

- Acondicionamiento de la plataforma.
- Simulación.
- Implementación física.

Lo anterior con la finalidad de cumplir con los objetivos previamente planteados haciendo uso de lo mencionado en la Metodología.

4.1. Acondicionamiento de la plataforma

El acondicionamiento corresponde a algunos cambios y actualizaciones realizadas en el prototipo AutoNOMOS Mini V2, estos cambios están divididos en dos partes: acondicionamiento de hardware y acondicionamiento de software. A continuación se describen estos

cambios para acondicionar al prototipo AutoNOMOS Mini V2, y de esta manera, desarrollar programas o algoritmos para las actividades posteriores.

4.1.1. Hardware

El hardware tiene 3 principales actualizaciones que se desarrollaron sobre el prototipo físico del AutoNOMOS Mini V2, en este sentido, la computadora principal fue reemplazada, así como la cámara RGB, del mismo modo el engrane de transmisión para tracción del movimiento de las 4 ruedas también fue reemplazado. Estas actualizaciones físicas se explican a continuación:

Computadora principal: La computadora con la que el prototipo contaba era una Odroid XU4 de 2 GB, esta computadora fue reemplazada debido a que solo admitía redes 2.4G en su conexión WiFi, siendo complicado intercambiar datos tales como imágenes y datos de los sensores a otras computadoras, esto debido a la pérdida de información causada por el ancho de banda, por lo que se requería de una conexión alámbrica de por medio limitando que el prototipo tenga un mayor nivel de autonomía. Por otro lado, la capacidad de procesamiento de esta computadora estaba limitada en cuanto al procesamiento requerido para realizar cálculos de los datos de imágenes y sensores, por lo que el cambio fue necesario.

Figura 4.1: Computadora Jetson Nano

En consecuencia, la computadora que el prototipo tenía era de arquitectura ARM (Odroid XU4 de 2 GB), reemplazada por una computadora de misma arquitectura la cual es una Jetson Nano de 4 GB de RAM (Figura 4.1), y aunado al doble en capacidad de memoria RAM, esta computadora está adicionada con una GPU con mayor velocidad de procesamiento lo que equivale a mayor cantidad de cálculos en un menor tiempo, necesaria para el cálculo de las redes neuronales, además, cuenta con un módulo dual de antenas WiFi y tecnología 5G, útil para la transferencia de datos sin pérdida. La Tabla 4.1 representa las características más importantes de las dos computadoras.

	Odroid	Jetson Nano
CPU	Mali-T628 6 núcleos	ARM Cortex-A57 de 4 núcleos Procesador MPCore
GPU	-	NVIDIA Maxwell con 128 núcleos
RAM	2 GB	4 GB

Tabla 4.1: Características de las computadoras ARM Odroid y Jetson.

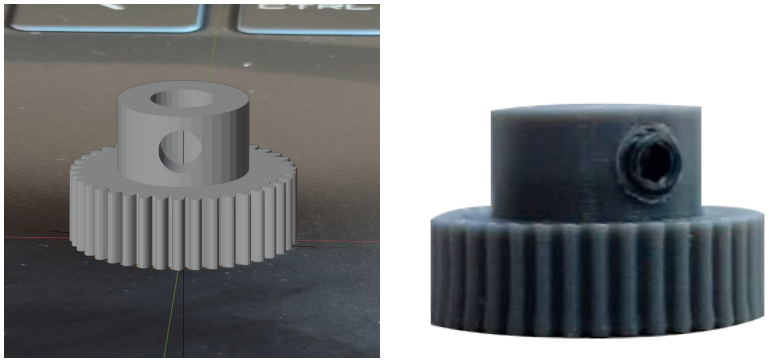
Cámara de profundidad: La cámara situada sobre el modelo AutoNOMOS Mini V2 y con la que este contaba es una cámara RGB Intel RealSense SR300 con profundidad, sin embargo, esta cámara contaba con algunos detalles en el funcionamiento, por un lado tenía fallas ante cambios rápidos de iluminación, ya que no ajustaba el brillo de la imagen de acuerdo al brillo percibido por la cámara, provocando imágenes oscuras en ambientes con poca iluminación e imágenes claras en ambientes con mucha iluminación, por otro lado el software controlador para esta cámara no tiene soporte en la computadora Jetson Nano, por lo que un cambio en esta cámara fue necesario.

Figura 4.2: Cámara utilizada en el modelo AutoNOMOS

Como ya se mencionó, el prototipo contaba con una cámara de profundidad Intel RealSense SR300, y resolución de 640x480 px a 30 FPS, la cual fue reemplazada por una cámara

RGB de profundidad Orbeec Astra Pro, esta cámara cuenta con las mismas características que la RealSense, sin embargo, se elimnan los errores de iluminación y debido a que el software es libre, resulta de fácil instalación. La Figura 4.2 muestra la nueva cámara que se instaló sobre el modelo físico.

Engrane de transmisión para tracción: El prototipo contaba con un engrane para generar la tracción montado en el eje del motor, no obstante, este engrane tenía un error en su maquinado ya que el eje del engrane se encontraba desfasado del centro del engrane, provocando un rozamiento extra en cada vuelta del eje, y viéndose reflejado cuando el prototipo tenía movimiento con una velocidad lenta, este problema provocaba la detención del prototipo.



(a) Modelo CAD (b) Resultado de impresión

Figura 4.3: Modelo y resultado de impresión del engrane de transmisión

Como resultado, se modeló un nuevo engrane con las características que el original tenía y se imprimió en una impresora 3D (Ender 3 Pro) con una calidad de extrusión de 0.12 mm y 80 % de relleno, la Figura 4.3(a) representa el modelo CAD, mientras que la Figura 4.3(b) representa el resultado de imprimir en 3D.

Conexiones: Finalmente, los cambios anteriormente mencionados dan como resultado el esquema de la Figura 4.4, en la cual se puede observar que se requieren de dos fuentes de alimentación, esto debido a la mayor cantidad de corriente que consume la computadora Jetson. Estas fuentes consisten en dos baterías de polímero de litio, una conectada a la parte lógica (computadora, arduino, cámara, LiDAR), mientras que la segunda alimenta únicamente al driver del motor y por ende, la tracción.

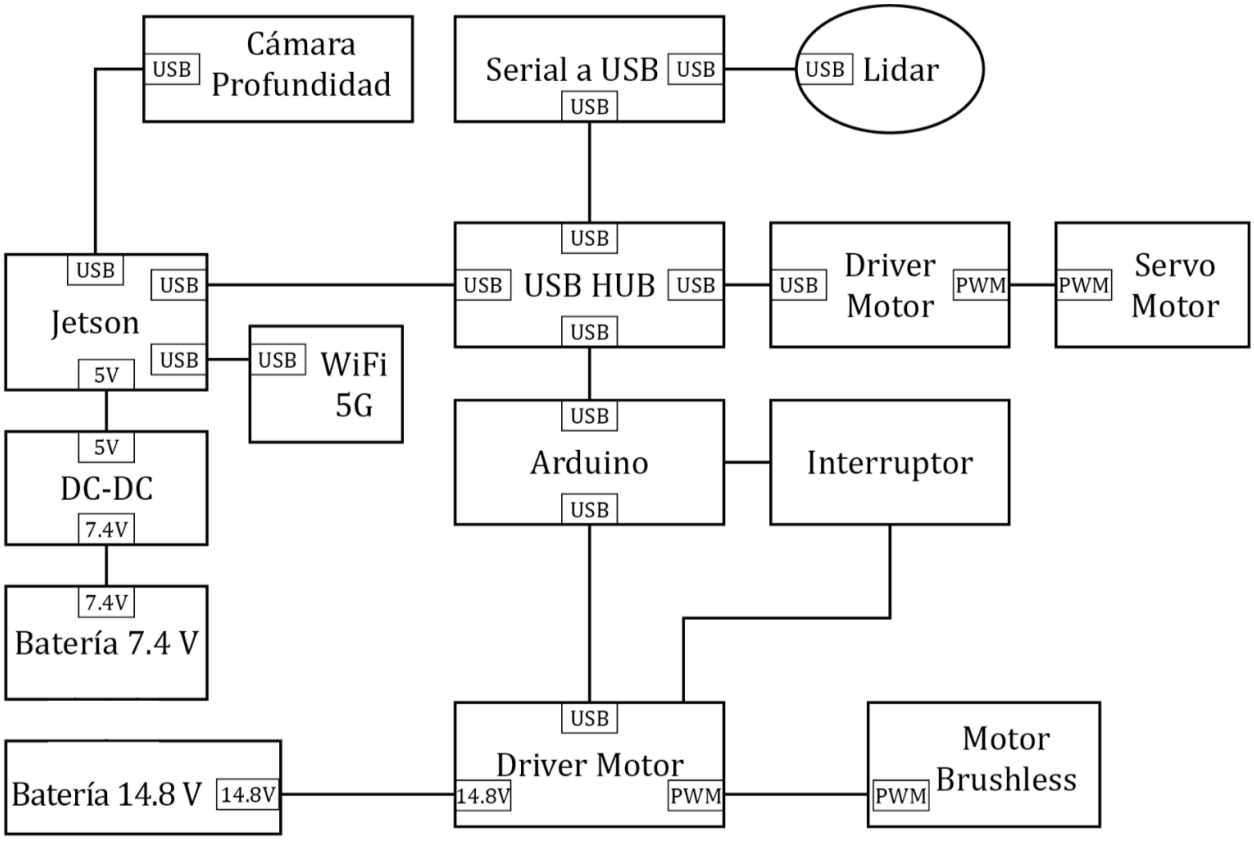


Figura 4.4: Conexiones físicas en el hardware

4.1.2. Software

La computadora Odroid anteriormente utilizada contaba con la versión de Ubuntu 16.04 sin interfaz gráfica, por lo que cualquier instrucción era dada únicamente por línea de comando, mientras que la Jetson Nano actualmente instalada cuenta con una versión de JetPack 4.6 (versión de la imagen para instalar) que contiene Ubuntu 18.04, y a diferencia de la Odroid, esta sí tiene una interfaz gráfica, por lo que además de poder ver información en la línea de comandos, también se puede observar lo que la cámara captura en tiempo real. Por otro lado la actualización de sistema ayudó a hacer uso de librerías especializadas en redes neuronales, como lo es la versión de Python 3, ya que Ubuntu 16.04 solamente cuenta con Python 2, así como la versión de numpy para operaciones matemáticas, y h5py, el cual es la extensión de los modelos exportados de redes neuronales ya entrenados.

El software representa el contenido de la computadora ARM Jetson Nano en cuanto al sistema que es utilizado para dar movimiento al prototipo, y se explica a continuación:

ROS: La versión de ROS correspondiente a Ubuntu 18.04 es Melodic Morenia (ya que cada versión de Ubuntu tiene su propia versión de ROS), disponible para arquitecturas ARM como es el caso de la Jetson, a diferencia de la computadora Odroid, al tener una versión gráfica, es posible instalar la versión completa de ROS, con todas las librerías incluidas, como lo es rviz, la cual tiene lo necesario para observar gráficamente lo que los sensores perciben, Gazebo, en el cual se pueden realizar ciertas simulaciones si así se desea, y además, generar bags, los cuales son útiles si se desea guardar información extra de algún sensor. Si bien, estas librerías son útiles para observar lo que sucede en el robot, fueron de ayuda para realizar el proceso de adquisición de datos más rápido en comparación a que no se hubiesen utilizado.

Tensorflow: La versión instalada de Tensorflow compatible con el sistema de la Jetson Nano es la versión 2, útil para implementar redes neuronales, la anterior computadora (Odroid XU4) no contaba con alguna versión de Tensorflow, y para su uso hace falta instalar las siguientes librerías con las respectivas versiones, compatibles con la versión de Ubuntu y de Jetson:

- python → 3.6
- numpy → 1.19.4
- future → 0.18.2
- mock → 3.0.5
- keras_preprocessing → 1.1.2
- keras_applications → 1.0.8
- gast → 0.4.0
- h5py → 3.1.0

La lista anterior dependerá y cambiará de acuerdo a la versión de computadora, y para la Jetson Nano 4GB es lo mencionado en dicha lista, ya que si no se instala alguna librería compatible es posible que la GPU no haga el cálculo y únicamente la CPU realizará estos procesos.

Auto-inicio: Al encender la computadora Jetson es pertinente ejecutar un archivo de inicialización, cuya labor es iniciar el AutoNOMOS Mini V2 de manera automática, levantar procesos y activar sensores, así como la ejecución de archivos de programa creados para

el modelo. En este sentido, se ejecuta el archivo ejecutable **autostart.sh** cada vez que la computadora Jetson es activada y encendida mediante el botón de inicio, ejecutando todos los procesos. A continuación se describe el contenido de estos procesos del archivo ***autostart.sh*** (Véase Anexos B.1 para el código completo) que la Figura 4.5 representa:

- Inicio: Se enciende la computadora.
- Permisos en puertos USB: Se otorgan los permisos de lectura y escritura en arduino para motor brushless (tracción del modelo) y en controlador para servomotor (ángulo de dirección).
- roscore: Se inicializa ROS y se detiene todo durante aproximadamente 5 segundos, lo necesario para terminar de ejecutar el proceso principal.
- roslaunch: Se inician todos los procesos del robot con este launch (cámara y motores).
- export ROS: Todos los procesos son enviados por la red siempre y cuando la computadora esté conectada a una zona WiFi.

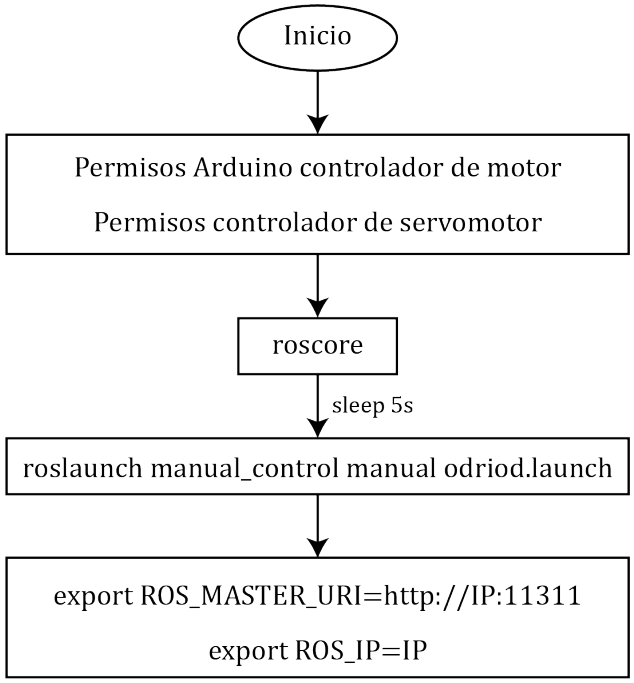


Figura 4.5: Esquema del ejecutable bash para iniciar el robot

4.2. Simulación

En esta sección se describe el desarrollo correspondiente al entorno de simulación, comenzando por las características del prototipo usado, la adquisición de los datos en el entorno de simulación Gazebo: Adquisición de la imagen, ángulo de dirección y láser LiDAR; y la creación de tres bases de datos para el entrenamiento de tres diferentes modelos CNN que cumplan con cada uno de los tres retos: Navegación autónoma sin obstáculos, con obstáculos estáticos y con obstáculos móviles.

4.2.1. Prototipo

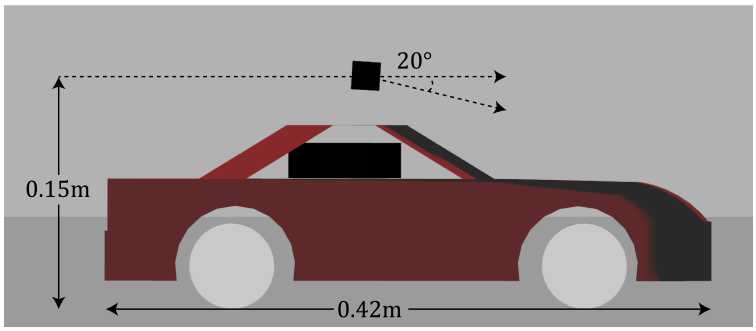


Figura 4.6: Medidas del prototipo elegido para realizar el movimiento en simulación

El prototipo usado para realizar las tareas de los tres retos se muestra en la Figura 4.6, el cual cuenta con una cámara RGB, esta cámara cuenta con las mismas características que el utilizado en el prototipo real (Astra Pro) en una posición adecuada para obtener la mejor resolución y eliminar el mayor ruido posible de la imagen. En relación con eso, la cámara se encuentra a una altura de 15 cm. sobre el suelo y a un ángulo de 20° negativos sobre la horizontal, el prototipo tiene 42 cm. de largo y 21 cm. de ancho. Por otro lado, el vehículo se desplaza a velocidad constante. A continuación se detalla la velocidad de desplazamiento empleada en cada tarea:

- Navegación autónoma sin obstáculos: 0,377 m/s
- Navegación autónoma con obstáculos estáticos: 0,377 m/s
- Navegación autónoma con obstáculos móviles: 0,48 m/s

Estas velocidades son seleccionadas en relación a la velocidad mínima permisible a la cual el prototipo real puede realizar una vuelta a una curva de 90° y evadir obstáculos siendo manejado por un mando, ambas tareas sin que el prototipo se detenga en su totalidad. No obstante, estas velocidades son utilizadas tanto para recabar la información, como para dar movimiento autónomo.

4.2.2. Entorno de simulación

El entorno seleccionado para la adquisición de datos es mostrado en la Figura 4.7 y se encuentra en el simulador Gazebo. Este entorno consiste en un circuito que mide 12 metros de largo por 9 metros de ancho. Cuenta con 4 curvas y 2 carriles para las 2 direcciones que se encuentran separadas por una línea discontinua, mientras que los límites de los carriles son líneas continuas de color blanco. La pista está sobre una base gris de dos tonos que funciona como diferenciador de la pista y el entorno.

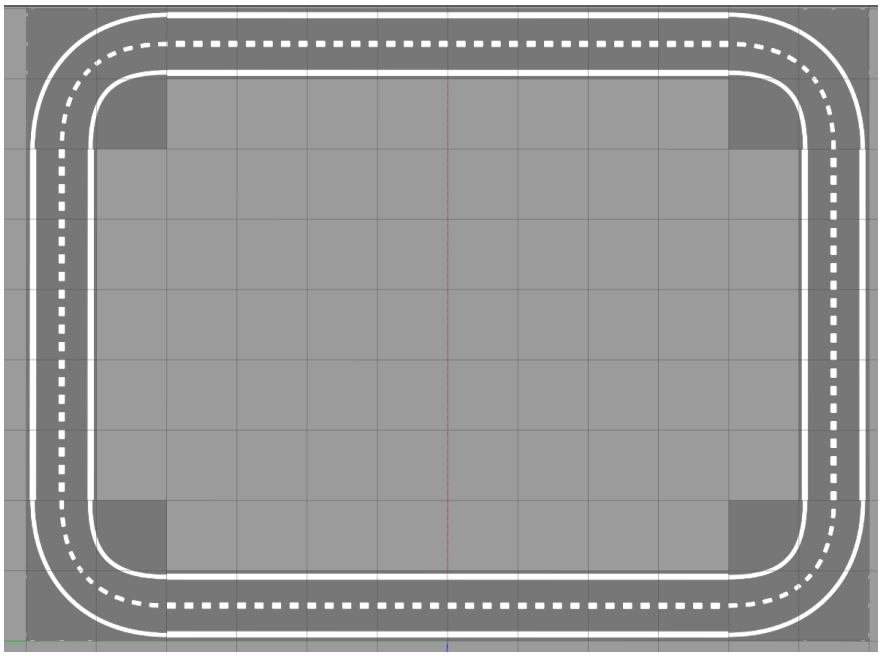
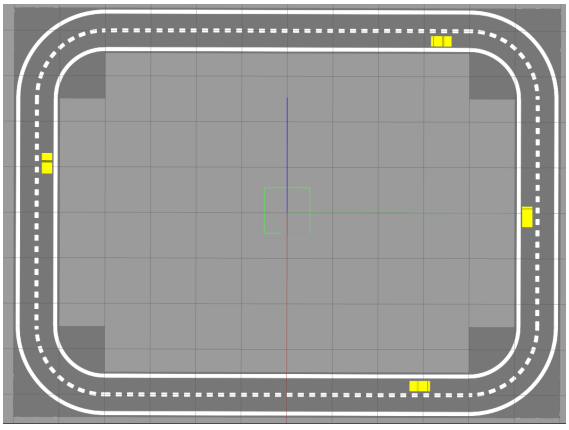


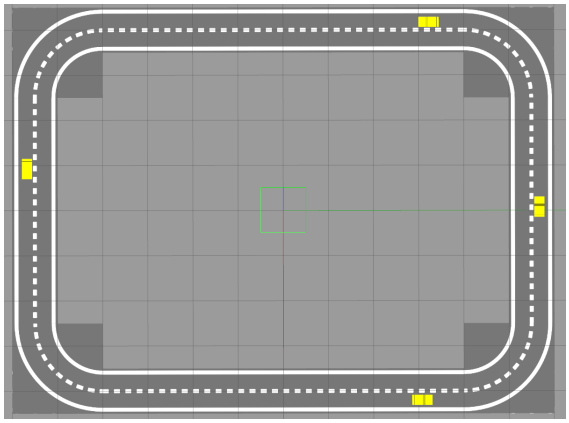
Figura 4.7: Entorno circular simple de 4 esquinas

El circuito mostrado en la Figura 4.7 es el utilizado para el primer reto: Navegación autónoma sin obstáculos.

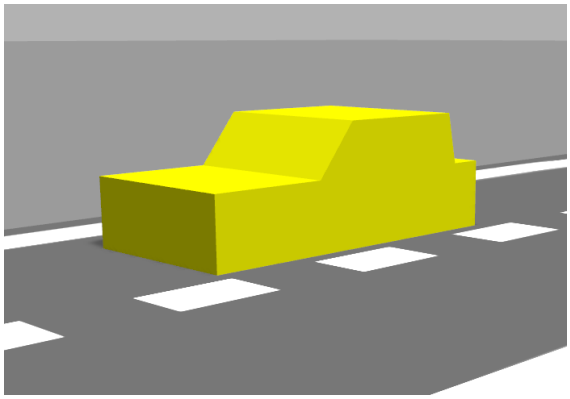
Por otro lado, la Figura 4.8 muestra el entorno utilizado para la navegación autónoma con obstáculos estáticos. Si bien es la misma que la descrita en la Figura 4.7, en este caso particular se han agregado obstáculos en el carril interno y externo del circuito, estos obstáculos pueden visualizarse en las Figuras 4.8(a) y 4.8(b) respectivamente, el obstáculo elegido para dicha tarea se observa en la Figura 4.8(c), que es un diseño asistido por computadora (CAD, por sus siglas en inglés) previamente diseñado en un programa (SolidWorks) y exportado al simulador (Gazebo). Este modelo cuenta con propiedades físicas y renderizado para colisiones, que además da significado a la forma geométrica que tiene.



(a) Obstáculos carril interno



(b) Obstáculos carril externo



(c) Obstáculo seleccionado para evadir

Figura 4.8: Descripción del entorno de simulación con obstáculos estáticos

En ese mismo orden de ideas, la Figura 4.9 muestra el obstáculo móvil seleccionado para el reto 3 (Navegación autónoma con obstáculos móviles), que es un prototipo duplicado al usado para hacer las tareas de navegación autónoma, con la única diferencia de que el color

de este obstáculo es azul. Este modelo tiene una velocidad constante que opera en el mismo carril que el prototipo principal, resultado de utilizar un controlador de posición que controla al prototipo obstáculo para desplazarse a diferentes puntos en el circuito, de manera que el modelo recorre toda la pista infinitamente, siempre y cuando el simulador esté en ejecución.

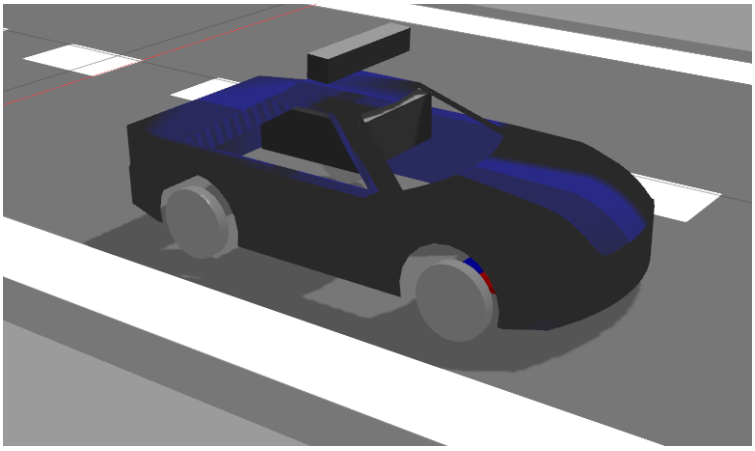


Figura 4.9: Obstáculo móvil propuesto

4.2.3. Adquisición de datos

La información necesaria para crear las bases de datos y por lo tanto, realizar los entrenamientos a las redes neuronales, provienen de las imágenes recopiladas por la cámara RGB, situada en la parte superior del prototipo. Por otro lado, la información del ángulo de dirección también es necesaria para la base de datos, y para el caso en el que se tienen obstáculos, se requiere de la información proveniente del sensor LiDAR. Por lo tanto, la adquisición de datos se divide en tres principales partes:

- Adquisición de la imagen
- Adquisición del ángulo de dirección
- Adquisición de la distancia de objetos mediante láser (LiDAR)

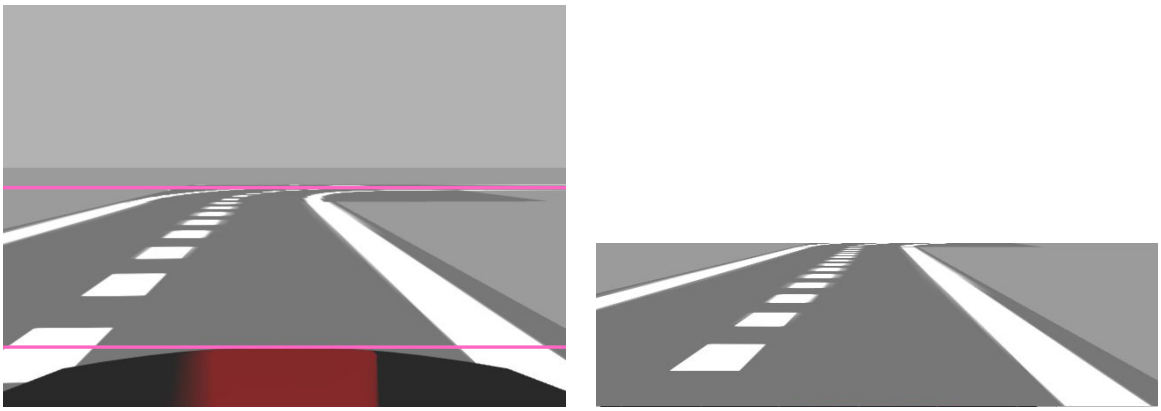
Esta adquisición de la información comprende desde la manera en que se encuentra situada la cámara, el área que observa la cámara, cómo se obtiene la información de la

cámara al mismo tiempo que se obtiene el ángulo de dirección y el sensor LiDAR, toda esta información debe suceder al mismo tiempo para asegurar que lo obtenido coincida entre sí al mismo tiempo.

4.2.3.1. Adquisición de la imagen

La imagen obtenida en el simulador se observa en la Figura 4.10(a), cuyas medidas son de 640x480 pixeles en un formato RGB, resultado de simular una cámara de profundidad con frecuencia de actualización de 30 Hz y con un campo de visión de 1,04 radianes.

Región de interés. El uso de una región de interés cumple el objetivo de eliminar el fondo y objetos que no son de importancia para la visión, descartando información innecesaria en la imagen. La Figura 4.10(a) muestra la imagen obtenida por la cámara y a la que se le aplica un recorte utilizando herramientas propias de python, obteniendo una vista especializada de la zona, el resultado puede observarse en la Figura 4.10(b), donde se obtiene un resultado más limpio y se aprecian mejor las líneas que comprenden el carril por el cual el modelo debe navegar autónomamente. Esta área seleccionada corresponde a los pixeles 225 a 410 en la altura comenzando de abajo, mientras que para el largo conserva los 640 pixeles de la cámara, lo que es suficiente para detectar las 2 líneas limitadoras del carril en uso y eliminar el fondo innecesario de la imagen.



(a) Imagen obtenida de la cámara (b) Imagen del área de interés

Figura 4.10: Área de interés en líneas rojas y resultado del área de interés

La imagen resultante de la Figura 4.10(b) es la utilizada para generar la base de datos, esta imagen es obtenida y guardada a una frecuencia de 20 Hz, y a una velocidad de desplazamiento del prototipo de 0,377 m/s para el caso en el que no hay obstáculos y en el que hay obstáculos estáticos, por otro lado, para el caso particular en el que los obstáculos son móviles, la velocidad es de 0,48 m/s, mencionado en la sección 4.2.1.

4.2.3.2. Adquisición del ángulo de dirección

El prototipo ya cuenta con un controlador para el ángulo de dirección, por lo tanto, asignar el ángulo de dirección deseado resulta posible sin realizar algún cambio, en ese sentido, la Figura 4.11 muestra que el ángulo de dirección que el prototipo puede alcanzar se encuentra entre -22° y 22° aproximadamente. Sin embargo, dado el formato necesario para utilizar los ángulos, estos datos se deben convertir a cantidades de Modulación por Ancho de Pulso (PWM, por sus siglas en inglés), los cuales se encuentran entre 0 y 180, donde 0 corresponde al ángulo máximo que puede girar a la derecha y 180 el ángulo máximo hacia la izquierda.

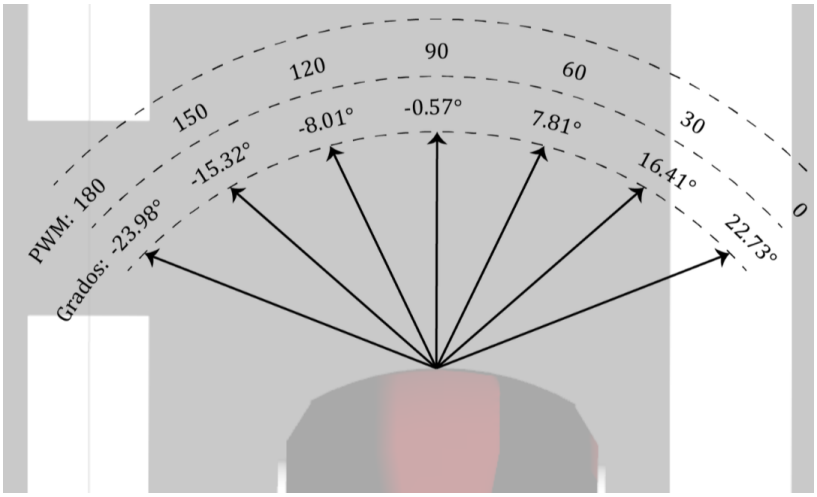
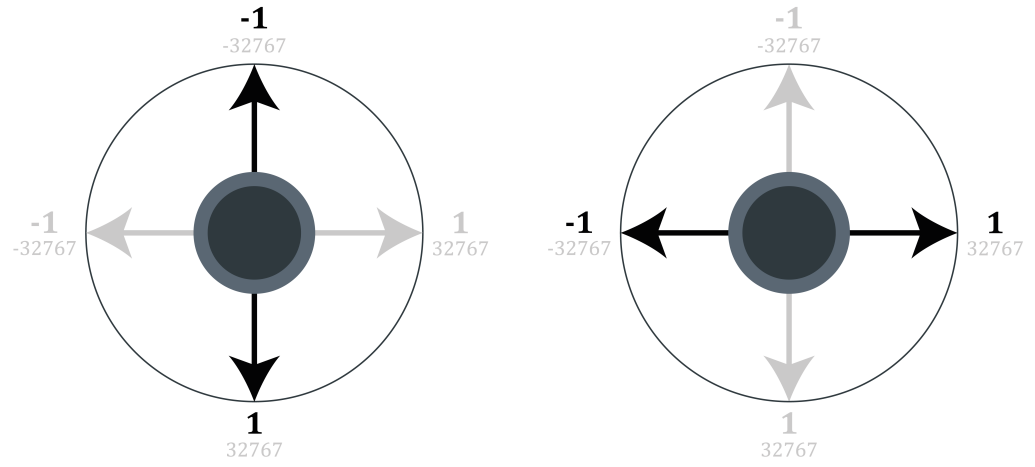


Figura 4.11: Parametrización del ángulo de dirección

La obtención de los ángulos de dirección generados para el movimiento del prototipo se obtienen a partir de un mando alámbrico de la marca Xbox®, mostrado en la Figura 4.12. Los potenciómetros utilizados son los señalados en dicha ilustración, por lo que el potenciómetro L es usado para asignar la velocidad, mientras que el potenciómetro R es utilizado para cambiar la dirección.

Figura 4.12: Mando para el movimiento del modelo en la simulación

Para la implementación en ROS es necesario el uso de un paquete controlador genérico de mandos. El paquete **joy** contiene el nodo **joy_node**, que interactúa entre el mando y ROS, este nodo publica constantemente un mensaje **Joy** que contiene el estado actual de cada uno de los botones y ejes de los joysticks en dos vectores, respectivamente (Véase Anexos C.1 para código de programación).



(a) Potenciómetro L de velocidad (b) Potenciómetro R de dirección

Figura 4.13: Valores de los potenciómetros L y R del mando

Los valores de los potenciómetros son los mostrados en la Figura 4.13, siendo el máximo un valor de 32767 y como mínimo un valor de -32767 en las 4 direcciones, sin embargo, el paquete utilizado normaliza estos valores y como resultado el valor máximo es 1, mientras que el valor mínimo es -1, el caso en el que los joystick se encuentren en cualquier zona de la cuadrícula, el resultado será un compuesto de los dos valores pertenecientes a dicha cuadrícula, siendo el máximo de 1,1 en el valor más alto, y para el caso en el que el potenciómetro está suelto, es decir, en el centro, el valor será 0.

En consecuencia, el ángulo de dirección es utilizado para crear la base de datos, estos datos son generados y guardados en una frecuencia de 20 Hz (del mismo modo que la imagen), mientras que el prototipo se desplaza a una velocidad de 0,377 m/s en el caso en el que no hay obstáculos y en el que hay obstáculos estáticos, para el caso en el que los obstáculos son móviles, la velocidad es de 0,48 m/s (mencionado en la sección 4.2.1).

4.2.3.3. Adquisición de la distancia de objetos mediante láser (LiDAR)

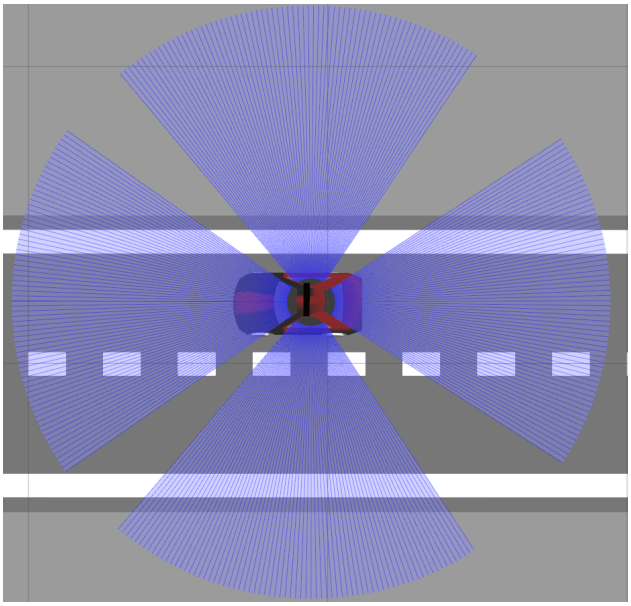


Figura 4.14: Rayos generados por sensor LiDAR y sus limitaciones

El sensor utilizado para medir distancias en los 360° del prototipo AutoNOMOS Mini V2 es un LiDAR, este sensor tiene la capacidad de detectar objetos a partir de 0.079 m y no mayores a una distancia de 8 m, sin embargo, para alcances de la detección de objetos de esta investigación en simulación, se ha reducido la distancia máxima hasta 1m., el rango

entre rayos láser es de 1°, por lo que el sensor da como resultado un vector de 360 datos en una frecuencia de 30 Hz. La Figura 4.14 muestra al prototipo AutoNOMOS Mini V2 con los rayos del sensor LiDAR de color azul, en esta misma figura se muestran cuatro zonas sin detección, esto debido a que el sensor se encuentra dentro de modelo y la carcasa no permite la salida del láser en esas zonas.

Por lo tanto, el sensor LiDAR empleado y del que además se desea guardar la información, realiza las lecturas y se guardan en una frecuencia de 30 Hz, una frecuencia diferente a la de la imagen y la del ángulo de dirección, esto debido a que no es posible manipular el tiempo de muestreo del sensor, esta información es recabada mientras el prototipo se desplaza a una velocidad de 0,377 m/s en el caso en el que los obstáculos son estáticos, mientras que para el caso en el que los obstáculos son dinámicos, la velocidad es de 0,48 m/s (información mencionada en la sección 4.2.1), el sensor LiDAR no es empleado en la tarea en la que no hay obstáculos.

4.2.4. Base de datos para la navegación autónoma sin obstáculos

Para la creación de la base de datos en la actividad en la que no hay obstáculos, es necesario almacenar la información obtenida por la región de interés a partir de la cámara y al mismo tiempo la información obtenida por el ángulo de dirección del prototipo a partir del mando y sus joysticks, todo esto mientras el prototipo es manipulado avanzando en el circuito un tiempo de 10 minutos sin salirse del carril y en ambas direcciones, respetando la circulación del carril derecho. La velocidad a la cual se realiza la navegación con mando es de 0,377 m/s (mencionado en la sección 4.2.1).

La Figura 4.15 muestra en un gráfico rqt (pluging de ROS explicado en la sección 2.6.2) las conexiones entre nodos y tópicos durante la obtención de las imágenes y ángulos para la creación de un rosbag (archivo de guardado generado por ROS y explicado en la sección 2.6.3) que contiene toda la información de la cámara y del mando durante el tiempo de grabación, y que además se encuentra etiquetado con el tiempo en el momento en el que se guarda un dato nuevo (útil para enlazar las imágenes con los ángulos en cada instante). El gráfico rqt es generado por la librería rqt_graph. Los nodos están señalados por óvalos, mientras que los tópicos son marcados por rectángulos y su uso se detalla a continuación:

- ***Nodos:***
  - **/joy_node**
    - Es el nodo encargado del enlace entre el mando y ROS.
  - **/ackermann_drive_joyop_node**
    - Nodo para mover al modelo del AutoNOMOS Mini en el simulador con el mando y sus joysticks.
  - **/gazebo**
    - Entorno de simulador donde se encuentra el circuito y modelo.
  - **/VIDEO**
    - Nodo que contiene la imagen del área de interés.
  - **/record**
    - Nodo generado a partir del rosbag para la grabación de los mensajes de imagen y ángulo.
- ***Tópicos:***
  - **/joy**
    - Tópico por el cual se mandan los mensajes en dos vectores del mando al software, en este caso, a ROS.
  - **/AutoNOMOS_mini/manual_control/speed**
    - Tópico del AutoNOMOS que recibe la velocidad proveniente del mando y lo envía al simulador.
  - **/AutoNOMOS_mini/manual_control/steering**
    - Tópico del AutoNOMOS que recibe la dirección de giro de las ruedas frontales del modelo proveniente del mando y lo envía al simulador.
  - **/app/camera/rgb/image_raw/compressed**
    - Tópico generado en el simulador que envía las imágenes provenientes de la cámara y las envía al nodo de /VIDEO.
  - **/angulo**
    - Tópico generado por el mando que se está publicando constantemente y que se guarda en el rosbag para generar la base de datos.
  - **/imageROI**
    - Tópico generado por el área de interés que se está publicando constantemente y que se guarda en el rosbag para generar la base de datos.

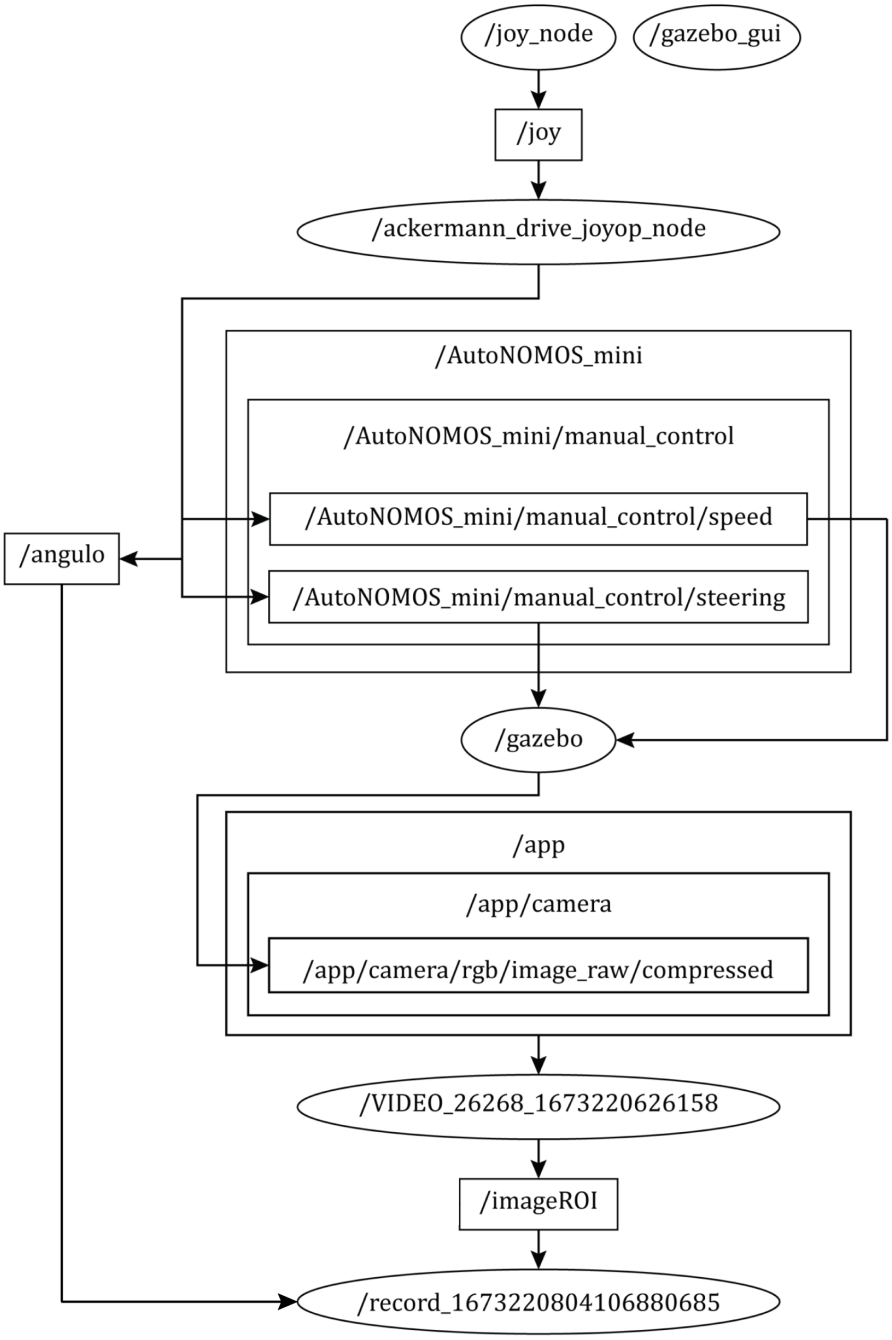


Figura 4.15: Gráfico rqt de tópicos y nodos para adquisición de datos para simulación

Algoritmo 1 Extracción de información del rosbag

```
1: function EXTRACCIÓN(imagesBag, anglesBag)  ▷ Extrae la información de la imagen y ángulo
2:     t1 ← timeseries(imagesBag).Time  ▷ Vector de tiempo de las imágenes
3:     t2 ← timeseries(anglesBag).Time  ▷ Vector de tiempo de los ángulos
4:     k ← 1
5:     si size(t2) mayor que size(t1) entonces  ▷ Sí el tiempo de ángulos es mayor que el de imágenes
6:         para i = 1 hasta size(t1) hacer  ▷ Recorrer el vector de tiempo de imágenes
7:             [val, indx] ← min(abs(t1(i) − t2))  ▷ Obtener el índice de los tiempos iguales
8:             si val es menor que 0,1 entonces  ▷ Si es menor a un umbral
9:                 idx(k, :) ← [i  indx]  ▷ Armar la matriz de índices
10:                k ← k + 1
11:            fin si
12:        fin para
13:    si no ▷ Sí el tiempo de imágenes es > que el de ángulos, hacer lo anterior viceversa
14:        para i = 1 hasta size(t2) hacer
15:            [val, indx] ← min(abs(t2(i) − t1))
16:            si val es menor que 0,1 entonces
17:                idx(k, :) ← [i  indx]
18:                k ← k + 1
19:            fin si
20:        fin para
21:    fin si
22:    regresa matriz : [contador indice]  ▷ Regresa una matriz de 2 columnas
23: fin function
```

El archivo generado a partir de la grabación de los tópicos es un rosbag con extensión **.bag**, y para extraer cada imagen con su respectivo ángulo es necesario hacer uso de un programa computacional manipulador de archivos, el pseudocódigo del programa se divide en dos partes, la extracción que representa el Algoritmo 1 y el guardado de la información, Algoritmo 2; en consecuencia, la Figura 4.16 muestra de manera gráfica la manera en que se extraen las imágenes y ángulos: Se crea un vector de imágenes y ángulos respectivamente, el vector de menor tamaño se toma como referencia, y se resta a todo el vector de mayor tamaño, se obtiene el valor mínimo absoluto y se crea una matriz del valor de ángulo con el índice en el que se encuentra en el vector, por último, si cada valor es menor que un umbral asignado entonces se guarda el índice en un vector con un contador, este contador está asociado a cada imagen, de esta manera se tiene conocimiento sobre qué ángulo pertenece a cada una de las imágenes (Véase Anexos C.3 para el código computacional de este proceso).

Algoritmo 2 Guardado de información extraído del rosbag

```
1: function GUARDADO(Matrix)                    ▷ Guarda las imágenes con su ángulo
2:     para i = 1 hasta size(idx) hacer
3:         imagen = lectura de imagenes en idx(i)
4:         angulo = lectura de angulos en idx(i)
5:         indice = i
6:         vector[i, 1] = i
7:         vector[i, 2] = angulo
8:         imwrite(imagen, indice.png)
9:     fin para
10:    writematrix(vector, steering.xls)
11: fin function
```

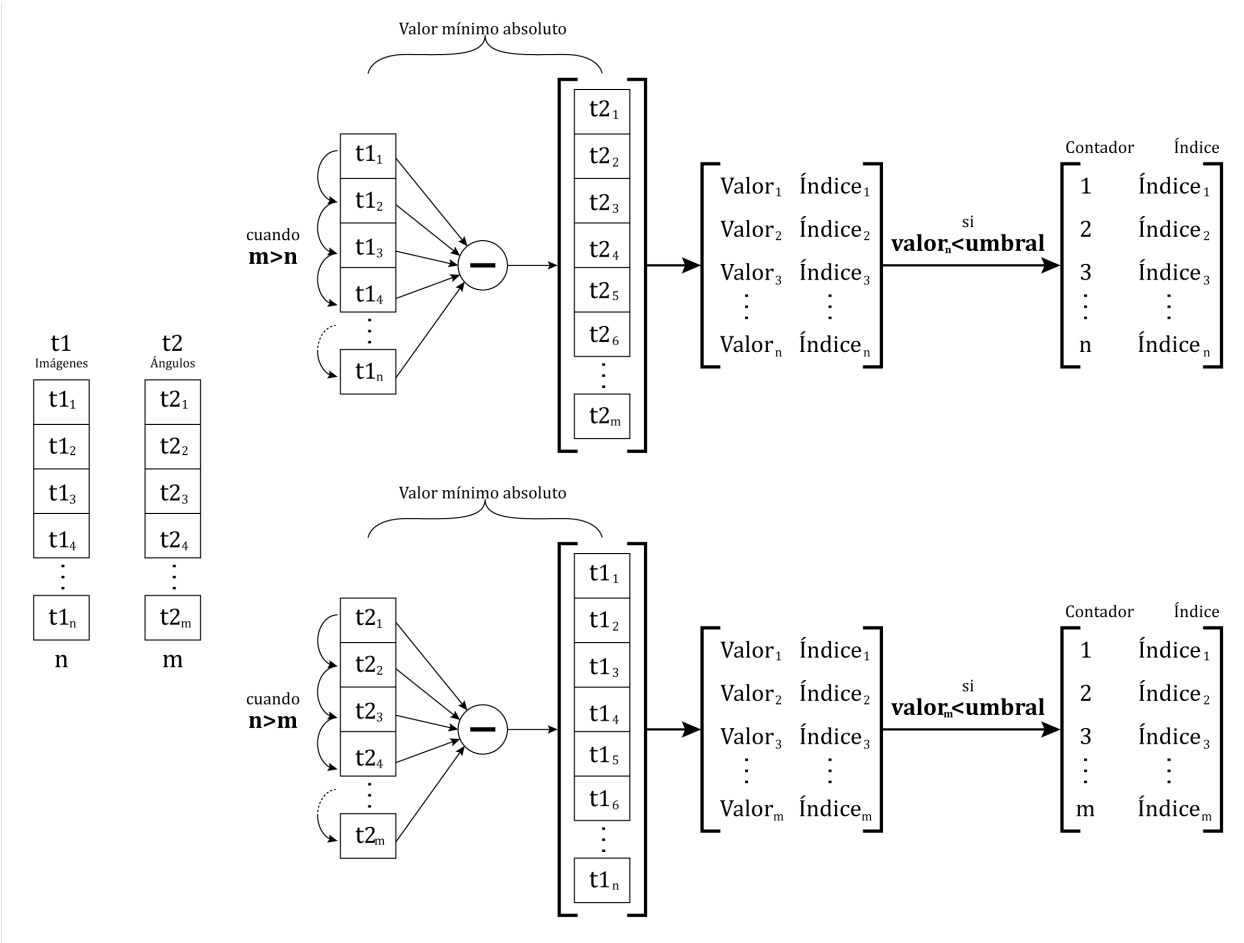


Figura 4.16: Descripción de los pseudocódigos 1 y 2

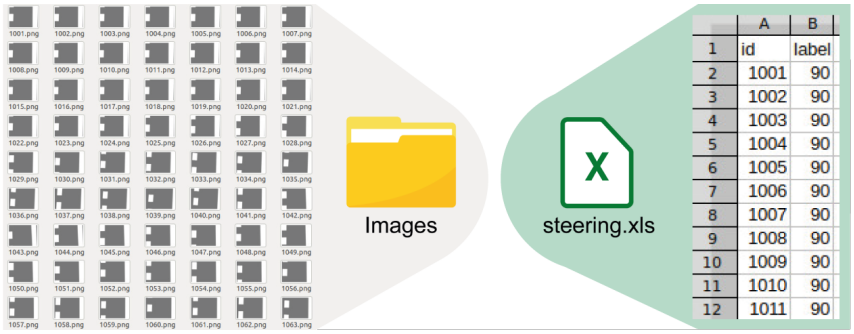


Figura 4.17: Base de datos generada a partir de las imágenes y ángulos de dirección

La base de datos es almacenada localmente en una carpeta (Figura 4.17), y a su vez la carpeta está constituida por dos archivos, un archivo llamado **steering.xls** y una carpeta llamada **Images**, la carpeta contiene 8351 imágenes, lo equivalente a 10 minutos del movimiento del prototipo con el mando en el simulador, mientras que el documento de extensión **.xls** es una hoja de trabajo que contiene 2 columnas, la primer columna equivale al nombre de cada imagen (**id**), mientras que la segunda columna representa la etiqueta (**label**), es decir, el ángulo.

4.2.5. Base de datos para la navegación autónoma con obstáculos estáticos

La creación de una base de datos para la navegación autónoma en la que hayan obstáculos estáticos consta de reunir la información necesaria para realizar dicha tarea, por ello es necesario obtener imágenes de la región de interés obtenidas a partir de la cámara, el ángulo de dirección que el prototipo obtiene por cada imagen, y a diferencia de la navegación sin obstáculos (caso anterior, sección 4.2.4), es necesario utilizar la información que el sensor LiDAR registra en el mismo tiempo que la cámara y el ángulo de dirección. Esta información es obtenida mientras el prototipo es guiado utilizando un mando y esquivando cuatro obstáculos amarillos situados en las cuatro rectas del circuito y en el sentido derecho de la pista en el que se esté moviendo el prototipo (circuito de la Figura 4.8).

La Figura 4.18 muestra el gráfico rqt de las conexiones entre nodos y tópicos durante la extracción de información para un archivo rosbag que contiene la información de la cámara, el ángulo de dirección y el vector de distancias del sensor LiDAR. Los nodos son mostrados por óvalos, mientras que los tópicos son rectángulos. A diferencia del rqt de la Figura 4.15, la Figura 4.18 muestra que también se graba el sensor LiDAR del tópico **\scan**.

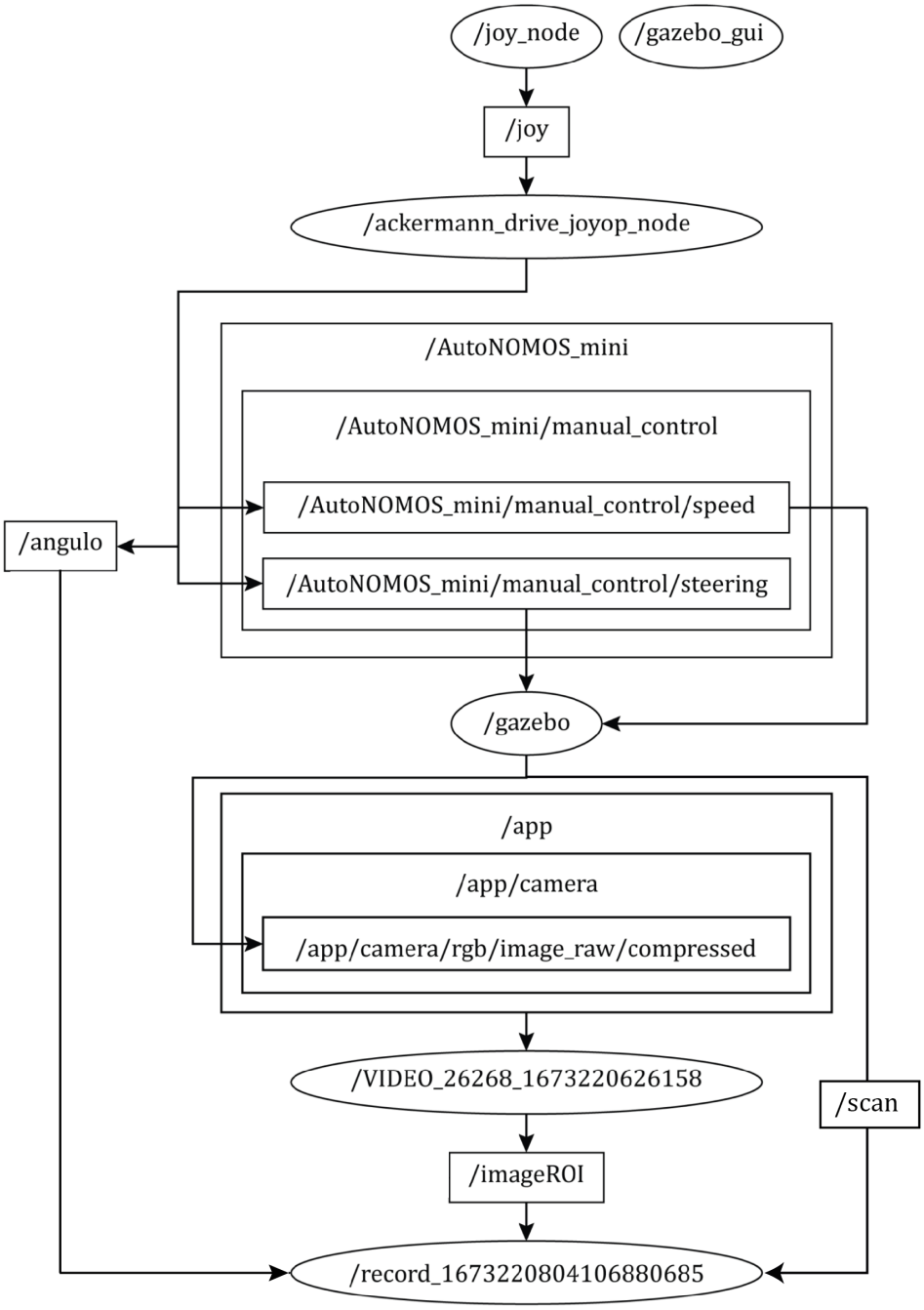


Figura 4.18: Gráfico rqt de tópicos y nodos para adquisición de datos en simulación de imágenes, ángulo de dirección y LiDAR

- Tópico
    - \scan: En este tópico se manda la información del LiDAR en vector con una frecuencia de 30 Hz.

El archivo **.bag** generado durante el manejo del prototipo para la creación de la base de datos contiene la imagen, ángulo de dirección y vector del LiDAR, sin embargo es necesario un script para manipular los archivos (Véase C.4 para programa completo), el pseudocódigo se divide en dos partes, la extracción de la información (Algoritmo 3) y el guardado de esta información (Algoritmo 4).

A diferencia de la creación de la base de datos sin obstáculos, es necesario convertir la información contenida en los vectores del LiDAR en imágenes, esta información está guardada en una hoja de cálculo al igual que el ángulo, para realizar esta transformación, se utiliza otro script que convierte cada vector de 360 datos en una imagen de 20 pixeles por 18 pixeles (Véase C.5 para programa completo), en una escala de 0 a 255 y profundidad de 1 (escala de grises), el pseudocódigo que realiza esta tarea es el Algoritmo 5 y la Figura 4.19 muestra el resultado de un vector convertido a imagen que proviene del vector de distancias. En este sentido, los pixeles más claros representan un objeto detectado, y a mayor cercanía este se acerca más a un color blanco, mientras que las zonas de ausencia de objetos tienden a color negro.

Figura 4.19: Resultado de convertir el vector de distancias LiDAR a imagen

Algoritmo 3 Extracción de información del rosbag

function EXTRACCIÓN($imagesBag, anglesBag, lidarBag$) ▷ Extrae la información
  $t1 \leftarrow timeseries(imagesBag).Time$ ▷ Vector de tiempo de las imágenes
  $t2 \leftarrow timeseries(anglesBag).Time$ ▷ Vector de tiempo de los ángulos
  $t3 \leftarrow timeseries(lidarBag).Time$ ▷ Vector de tiempo de mediciones LiDAR
  $k \leftarrow 1$ ▷ Sí el tiempo de ángulo y el tiempo de LiDAR es mayor que el de imágenes:
  si $size(t2)$ *mayor que* $size(t1)$ **y** $size(t3)$ *mayor que* $size(t1)$ **entonces**
    para $i = 1$ *hasta* $size(t1)$ **hacer** ▷ Recorrer el vector de tiempo de imágenes
      $[val1, indx1] \leftarrow min(abs(t1(i) - t2))$ ▷ Índice de tiempos imágenes-ángulos
      $[val2, indx2] \leftarrow min(abs(t1(i) - t3))$ ▷ Índice de tiempos imágenes-LiDAR
      si $val1$ *menor que* 0,1 **y** $val2$ *menor que* 0,1 **entonces** ▷ Menor a umbral
        $idx1(k, :) \leftarrow [i \;\; indx1]$ ▷ Armar la matriz de índices imágenes-ángulos
        $idx2(k, :) \leftarrow [i \;\; indx2]$ ▷ Armar la matriz de índices imágenes-LiDAR
        $k \leftarrow k + 1$
      fin si
    fin para
  fin si ▷ Sí el tiempo de ángulo y el tiempo de LiDAR es mayor que el de imágenes:
  si $size(t1)$ *mayor que* $size(t2)$ **y** $size(t3)$ *mayor que* $size(t2)$ **entonces**
    para $i = 1$ *hasta* $size(t2)$ **hacer** ▷ Recorrer el vector de tiempo de ángulos
      $[val1, indx1] \leftarrow min(abs(t2(i) - t1))$ ▷ Índice de tiempos ángulos-imágenes
      $[val2, indx2] \leftarrow min(abs(t2(i) - t3))$ ▷ Índice de tiempos ángulos-LiDAR
      si $val1$ *menor que* 0,1 **y** $val2$ *menor que* 0,1 **entonces** ▷ Menor a umbral
        $idx1(k, :) \leftarrow [i \;\; indx1]$ ▷ Armar la matriz de índices ángulos-imágenes
        $idx2(k, :) \leftarrow [i \;\; indx2]$ ▷ Armar la matriz de índices ángulos-LiDAR
        $k \leftarrow k + 1$
      fin si
    fin para
  fin si ▷ Sí el tiempo de ángulo y el tiempo de LiDAR es mayor que el de imágenes:
  si $size(t1)$ *mayor que* $size(t3)$ **y** $size(t2)$ *mayor que* $size(t3)$ **entonces**
    para $i = 1$ *hasta* $size(t3)$ **hacer** ▷ Recorrer el vector de tiempo de LiDAR
      $[val1, indx1] \leftarrow min(abs(t3(i) - t1))$ ▷ Índice de tiempos LiDAR-imágenes
      $[val2, indx2] \leftarrow min(abs(t3(i) - t2))$ ▷ Índice de tiempos LiDAR-ángulos
      si $val1$ *menor que* 0,1 **y** $val2$ *menor que* 0,1 **entonces** ▷ Menor a umbral
        $idx1(k, :) \leftarrow [i \;\; indx1]$ ▷ Armar la matriz de índices LiDAR-imágenes
        $idx2(k, :) \leftarrow [i \;\; indx2]$ ▷ Armar la matriz de índices LiDAR-ángulos
        $k \leftarrow k + 1$
      fin si
    fin para
  fin si
  regresa $matriz : [contador \; indice]$ ▷ Regresa una matriz de 2 columnas
fin function

Algoritmo 4 Guardado de información extraído del rosbag

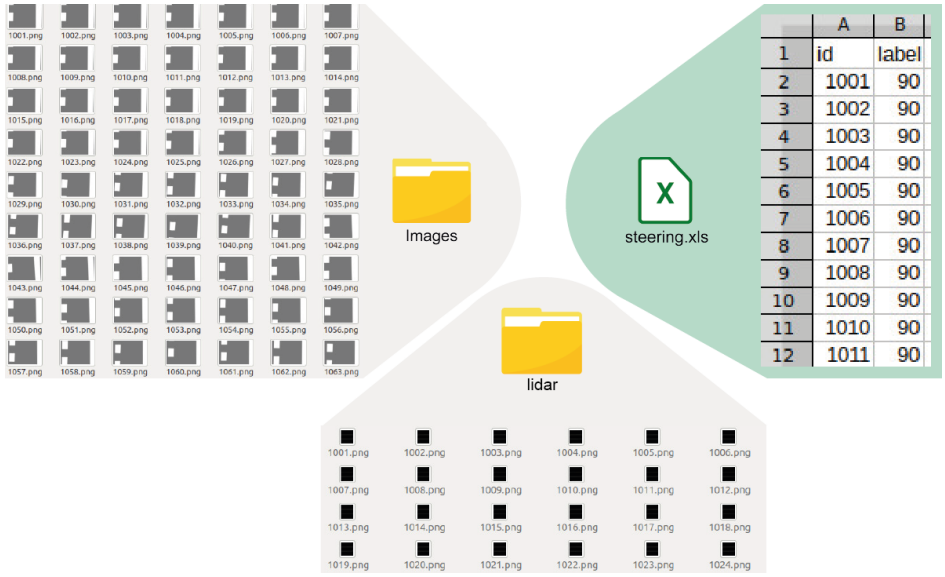
```
1: function GUARDADO(Matrix)                    ▷ Guarda imágenes, ángulo y vector LiDAR
2:     si size(t2) mayor que size(t1) y size(t3) mayor que size(t1) entonces
3:         para i = 1 hasta size(idx1) hacer   ▷ Recorrer el vector de tiempo de imágenes
4:             imagen = lectura de imagenes en idx(i)
5:             angulo = lectura de angulos en idx(i)
6:             lidar = lectura de vector lidar en idx(i)
7:             indice = i
8:             vector[i, 1] = i, vectorLidar[i, 1] = i
9:             vector[i, 2] = angulo
10:            vectorLidar[i, 2 : 361] = lidar'
11:            imwrite(imagen, indice.png)
12:        fin para
13:    fin si
14:    si size(t1) mayor que size(t2) y size(t3) mayor que size(t2) entonces
15:        para i = 1 hasta size(idx1) hacer   ▷ Recorrer el vector de tiempo de imágenes
16:            imagen = lectura de imagenes en idx(i)
17:            angulo = lectura de angulos en idx(i)
18:            lidar = lectura de vector lidar en idx(i)
19:            indice = i
20:            vector[i, 1] = i, vectorLidar[i, 1] = i
21:            vector[i, 2] = angulo
22:            vectorLidar[i, 2 : 361] = lidar'
23:            imwrite(imagen, indice.png)
24:        fin para
25:    fin si
26:    si size(t1) mayor que size(t3) y size(t2) mayor que size(t3) entonces
27:        para i = 1 hasta size(idx1) hacer   ▷ Recorrer el vector de tiempo de imágenes
28:            imagen = lectura de imagenes en idx(i)
29:            angulo = lectura de angulos en idx(i)
30:            lidar = lectura de vector lidar en idx(i)
31:            indice = i
32:            vector[i, 1] = i, vectorLidar[i, 1] = i
33:            vector[i, 2] = angulo
34:            vectorLidar[i, 2 : 361] = lidar'
35:            imwrite(imagen, indice.png)
36:        fin para
37:    fin si
38:    writematrix(vector, steering.xls), writematrix(lidar, lidar.xls)
39: fin function
```

Algoritmo 5 Vector LiDAR a imagen

```
1: function LidarToImg(Lidar.xls)   ▷ Extrae y convierte en imágenes los vectores de distancia
2:     LidarData = extraer(Lidar.xls)
3:     para k = 1 hasta size(LidarData) hacer
4:         etiqueta = LidarData(k)
5:         matrizImg = zeros[20, 18]
6:         Bandera = 2
7:         para i = 1 hasta size(matrizImg, renglones) hacer
8:             para j = 1 hasta size(matrizImg, columnas) hacer
9:                 si LidarData(k, Bandera) mayor que 1 entonces
10:                    matrizImg[i, j] = 0
11:                si no
12:                    value = (lidarData(k, flag)/1) * 255
13:                    matrizImg(i, j) = value
14:                fin si
15:                flag = flag + 1
16:            fin para
17:        fin para
18:    fin para
19:    lidarFileName = strcat(lidar + label + png)
20:    imwrite(matrizImg, lidarFileName)
21: fin function
```

Por último, la base de datos es almacenada localmente en una carpeta primaria (Figura 4.20), esta carpeta a su vez contiene dos carpetas secundarias y un archivo llamado **steering.xls**, las carpetas corresponden a las imágenes de la pista y a las imágenes del sensor LiDAR, por lo que cada carpeta contiene 42'219 imágenes. Por otro lado, el documento de extensión **.xls** contiene 2 columnas: la primer columna equivale al identificador de cada imagen, mientras que la segunda equivale a la etiqueta (ángulo de dirección), y al igual que en los otros casos este cuenta con 42'219 datos.

El prototipo fue manejado por el mando un tiempo aproximado de 40 minutos (20 minutos para cada carril del circuito) y a una velocidad constante de 0,377 m/s (mencionado en la sección 4.2.1), con una frecuencia de actualización de 20 Hz en la imagen de la cámara así como en el ángulo de dirección, mientras que para las imágenes del LiDAR la frecuencia de actualización es de 30 Hz.



	A	B
1	id	label
2	1001	90
3	1002	90
4	1003	90
5	1004	90
6	1005	90
7	1006	90
8	1007	90
9	1008	90
10	1009	90
11	1010	90
12	1011	90

Figura 4.20: Base de datos generada a partir de las imágenes de cámara, imágenes de LiDAR y ángulos de dirección

4.2.6. Base de datos para la navegación autónoma con obstáculos móviles

La base de datos para la navegación en el caso en el que se cuenta con obstáculos móviles tiene un total de 62′283 imágenes provenientes de la cámara, 62′283 imágenes transformadas del sensor LiDAR, y por lo tanto, la misma cantidad de datos etiquetados (ángulo de dirección). De los cuales, 42′419 datos son provenientes del segundo reto (detallado en la sección 4.2.5), de modo que 19′864 son datos nuevos, tal como se muestra en la Figura 4.21, la razón de este aumento es debido a que el obstáculo debe estar en movimiento, y estos nuevos datos corresponden a un obstáculo similar al AutoNOMOS (prototipo empleado para cumplir los retos) que cuenta con las mismas propiedades para dar movimiento, con la única diferencia del color (el prototipo es rojo, mientras que el obstáculo azul). La grabación de estos datos nuevos fue de aproximadamente 5 minutos entre los dos carriles.

La velocidad del prototipo para esta tarea es de 0,48 m/s (detallado en la sección 4.2.1), por otro lado, las frecuencias de actualización para la imagen de la cámara y para el ángulo de dirección son de 20 Hz, mientras que para la imagen del sensor LiDAR es de 30 Hz.

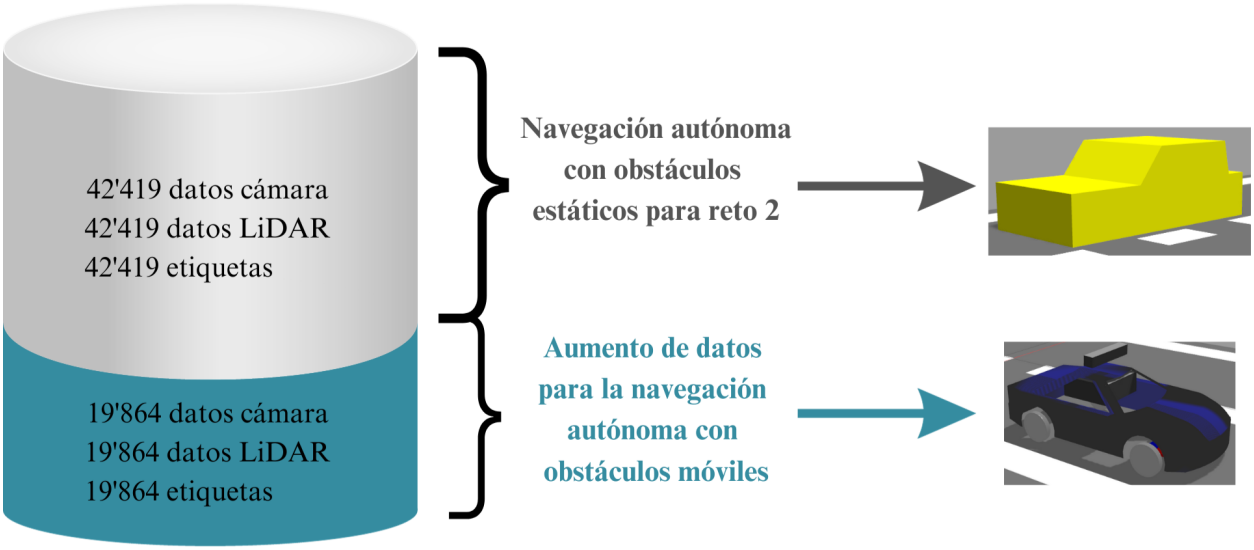


Figura 4.21: Base de datos para la navegación autónoma con obstáculos móviles

4.2.7. Red neuronal convolucional para reto 1: Navegación autónoma sin obstáculos

La base de datos creada con imágenes y ángulos de dirección son la entrada y salida respectivamente de la red neuronal convolucional (CNN) a implementar para su entrenamiento, y además, es una modificación de lo planteado por NVIDIA en la Figura 1.3, con la finalidad de realizar navegación autónoma prediciendo el ángulo de dirección a partir de lo observado por la cámara.

Por lo tanto, la Figura 4.22 es la CNN propuesta para el movimiento autónomo del prototipo en la simulación, específicamente en la navegación sin obstáculos, esta red cuenta con cinco capas convolucionales que posteriormente, lo mapas de características son aplanados, para finalmente pasar a cuatro capas del regresor completamente conectadas de manera secuencial. Esta propuesta fue elegida en la búsqueda del menor error posible a la base de datos empleada para la tarea.

Las partes de esta CNN se detallan en las tablas 4.2 y 4.3, representando la parte convolucional y la parte de regresión lineal respectivamente (Véase Anexos D.1 para mayor detalle en la construcción del programa computacional). Todas las capas convolucionales cuentan con kernel de valores aleatorios, y posteriormente por una función de activación, en este caso por una función **elu** de la cual se extraen mapas de características en sus diferentes capas.

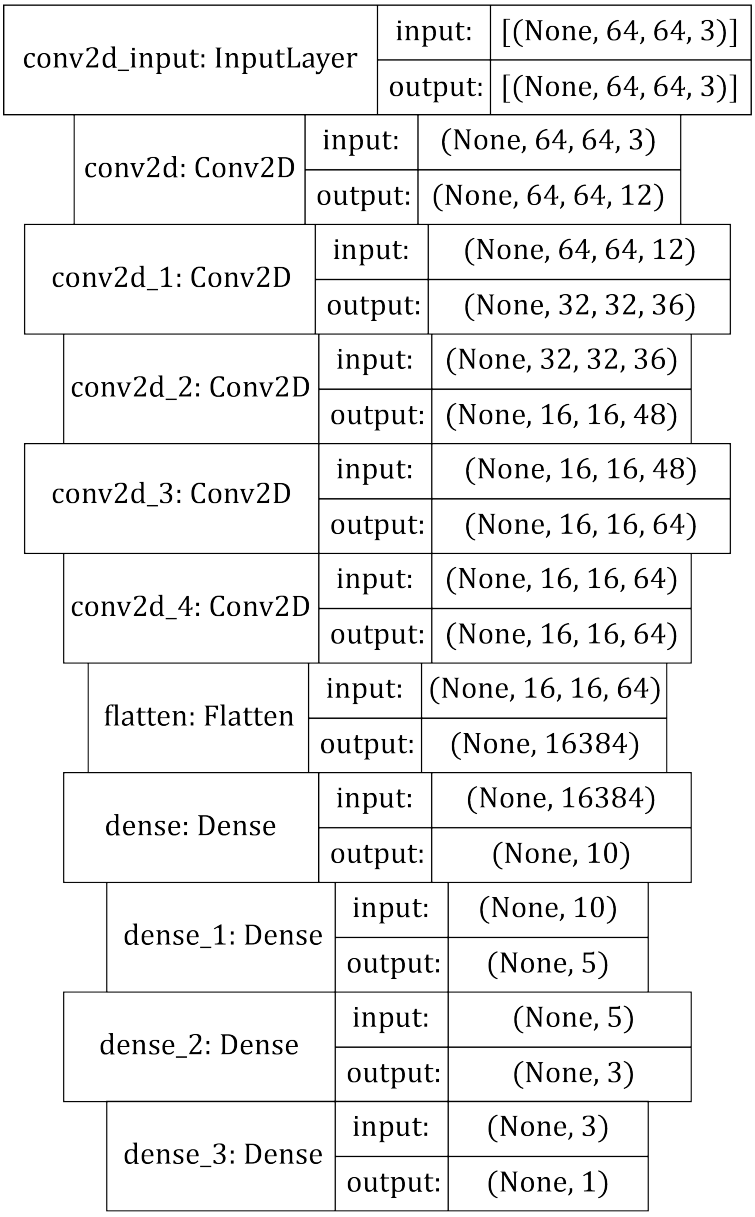


Figura 4.22: Red neuronal convolucional para la dirección del reto uno (Véase E.1)

La extracción de estos mapas de características pasan por un aplanado formando un vector denominado **flatten**, dicho vector está conformado por 262′144 valores que representan los valores de las características ordenadamente. Este aplanado es presentado al regresor lineal, formado por tres capas ocultas cuya función de activación es de tipo **elu** para finalmente en la capa **fc9** hacer una ponderación lineal de la capa oculta inmediata anterior y obtener el

resultado que finalmente se convertirá en el ángulo de dirección tras ser entrenada la CNN.

Capa	Tamaño		Mapas de características	Función de activación
	x	y		
conv1+elu	64	64	12	elu
conv2+elu	64	64	36	elu
conv3+elu	64	64	48	elu
conv4+elu	64	64	64	elu
conv5+elu	64	64	64	elu

Tabla 4.2: Parte convolucional de la CNN a entrenar para el movimiento del prototipo

Capa	Tamaño	Función de activación
flatten	262144	-
fc6+elu	10	elu
fc7+elu	5	elu
fc8+elu	3	elu
fc9+elu	1	-

Tabla 4.3: Parte neuronal de la CNN a entrenar para el movimiento del prototipo

4.2.8. Red neuronal convolucional para reto 2: Navegación autónoma con obstáculos estáticos

La base de datos creada con las imágenes del entorno, imágenes del sensor LiDAR y sus respectivos ángulos de dirección son las entradas y salidas, respectivamente de la CNN propuesta para implementar el entrenamiento.

La Figura 4.23 muestra la CNN propuesta para dar movimiento y evasión de obstáculos de manera autónoma al prototipo, está divida en dos partes, y cada una recibe una imagen: la que proviene de la cámara y la que proviene del sensor LiDAR, cuenta con 4 capas convolucionales completamente conectadas, se realiza un aplanado en ambos casos y se concatenan, para finalmente estar conectado a 4 capas de un regresor. Esta propuesta es una modificación de lo mencionado en la sección 4.2.7, que a su vez es la modificación planteada por NVIDIA en la Figura 1.3, sin embargo, como ya se mencionó, en este caso cuenta con una entrada extra a la CNN, por lo tanto, partir de la modificación empleada en la sección

anterior otorga un panorama de lo que debería tenerse como resultado para buscar el mínimo error posible, realizando algunas modificaciones para este caso.

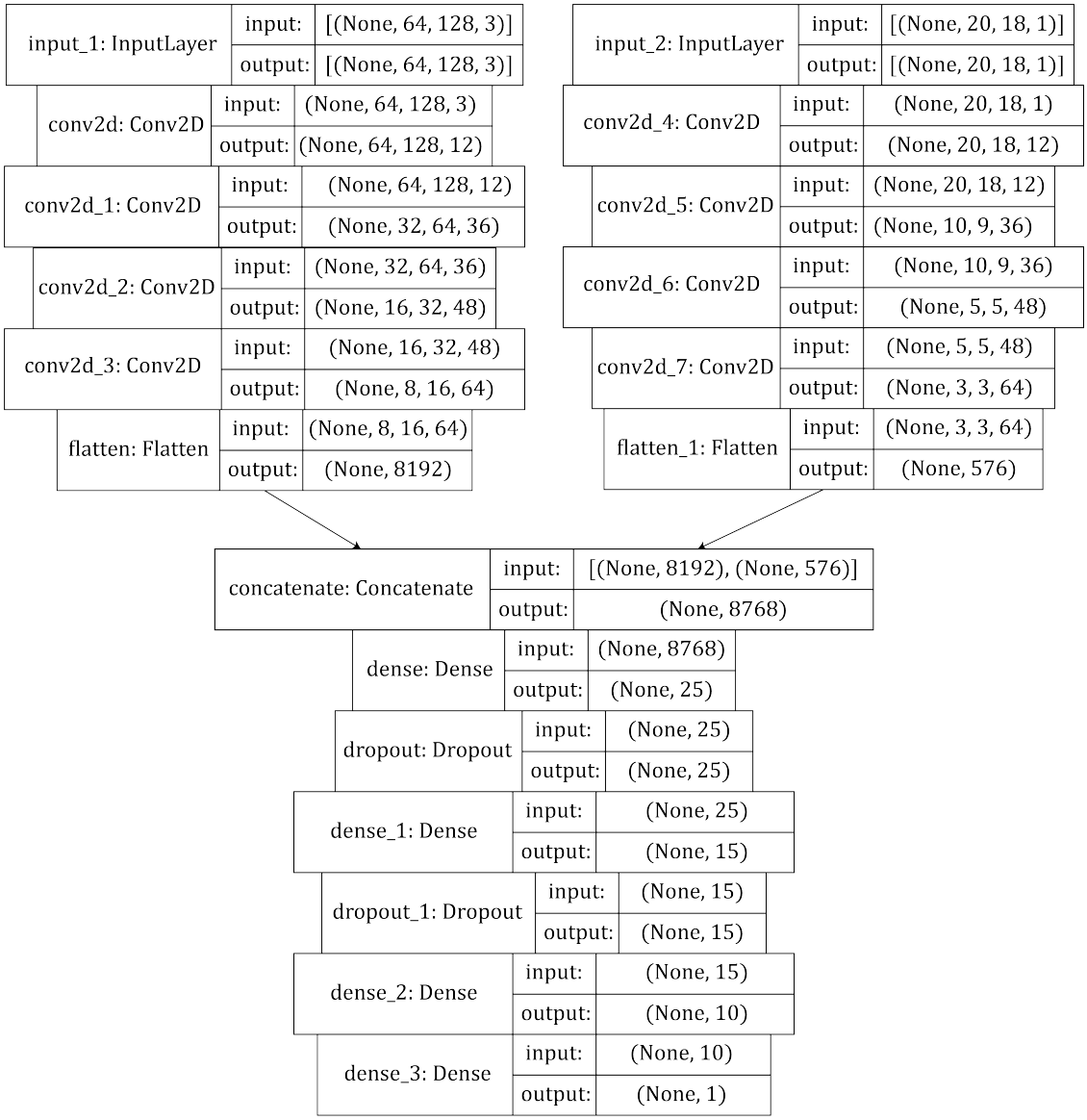


Figura 4.23: Red neuronal convolucional con dos entradas para el reto dos(Véase E.2)

Cada sección de la CNN se detalla en las Tablas 4.4 y 4.5: La primera corresponde a la imagen de la cámara, que tiene una dimensión de 128x64x3 pixeles, pasa por 4 capas convolucionales hasta tener un tamaño de 32x16 y 64 capas correspondientes a los mapas

de características, se realiza un aplanado obteniendo una longitud de 8'192, por otro lado, la segunda tabla corresponde a la parte de la imagen del sensor lidar, cuya entrada es de 18x20 pixeles, recorre 4 capas convolucionales hasta tener un tamaño de 3x3 y 64 capas de características, todas con una función de activación **elu**.

Capa	Tamaño		Mapas de características	Función de activación
	x	y		
Img_Cam	128	64	3	elu
conv1+elu	128	64	12	elu
conv2+elu	64	32	36	elu
conv3+elu	64	32	48	elu
conv4+elu	32	16	64	elu

Tabla 4.4: Sección convolucional de la CNN, parte superior (imagen de la cámara).

Capa	Tamaño		Mapas de características	Función de activación
	x	y		
Img_Lidar	18	20	3	elu
conv1+elu	18	20	12	elu
conv2+elu	9	10	36	elu
conv3+elu	5	5	48	elu
conv4+elu	3	3	64	elu

Tabla 4.5: Sección convolucional de la CNN, parte inferior (imagen del sensor lidar).

Por último, la Tabla 4.6 muestra que después de concatenar los datos de ambas capas convolucionales pasa por 4 capas del regresor completamente conectadas, para que la capa **fc9** tenga una sola neurona para la salida que corresponde al ángulo de dirección (Véase D.2 para mayor información sobre el programa computacional completo).

Capa	Tamaño	Función de activación
flatten	8768	-
fc5+elu	25	elu
fc6+elu	15	elu
fc7+elu	10	elu
fc8+elu	1	-

Tabla 4.6: Parte neuronal de la CNN después de la concatenación

4.2.9. Red neuronal convolucional para reto 3: Navegación autónoma con obstáculos móviles

La CNN propuesta para esta tarea es similar a la mostrada en el segundo reto (Figura 4.23, sección 4.2.8), sin embargo, al ser este un reto más complejo, cuenta con algunos cambios adicionales, buscando que el entrenamiento encuentre el menor error posible empleando la base de datos para dicha tarea.

La red recibe las mismas dos imágenes de entrada, provenientes de la cámara y sensor LiDAR, por lo que busca predecir el ángulo de dirección, en consecuencia, se propone la topología mostrada en la Figura 4.24 (Véase D.3 para mayor detalle en el programa computacional), en ella se observan los siguientes cambios: Cuenta con una capa convolucional extra en la generación de mapas de características para ambas entradas justo antes de ser concatenadas, por otro lado, las capas ocultas de regresión tienen diferente cantidad de neuronas.

Capa	Tamaño	Función de activación
flatten	4608	-
fc5+elu	100	elu
fc6+elu	50	elu
fc7+elu	15	elu
fc8+elu	10	elu
fc9+elu	1	-

Tabla 4.7: Parte neuronal de la CNN después de la concatenación

La tabla 4.7 describe las capas internas que se encuentran después de la extracción de características, cuenta con cuatro capas completamente conectadas y una salida, esta salida no cuenta con función de activación, mientras que todas las demás capas utilizan la función de activación elu.

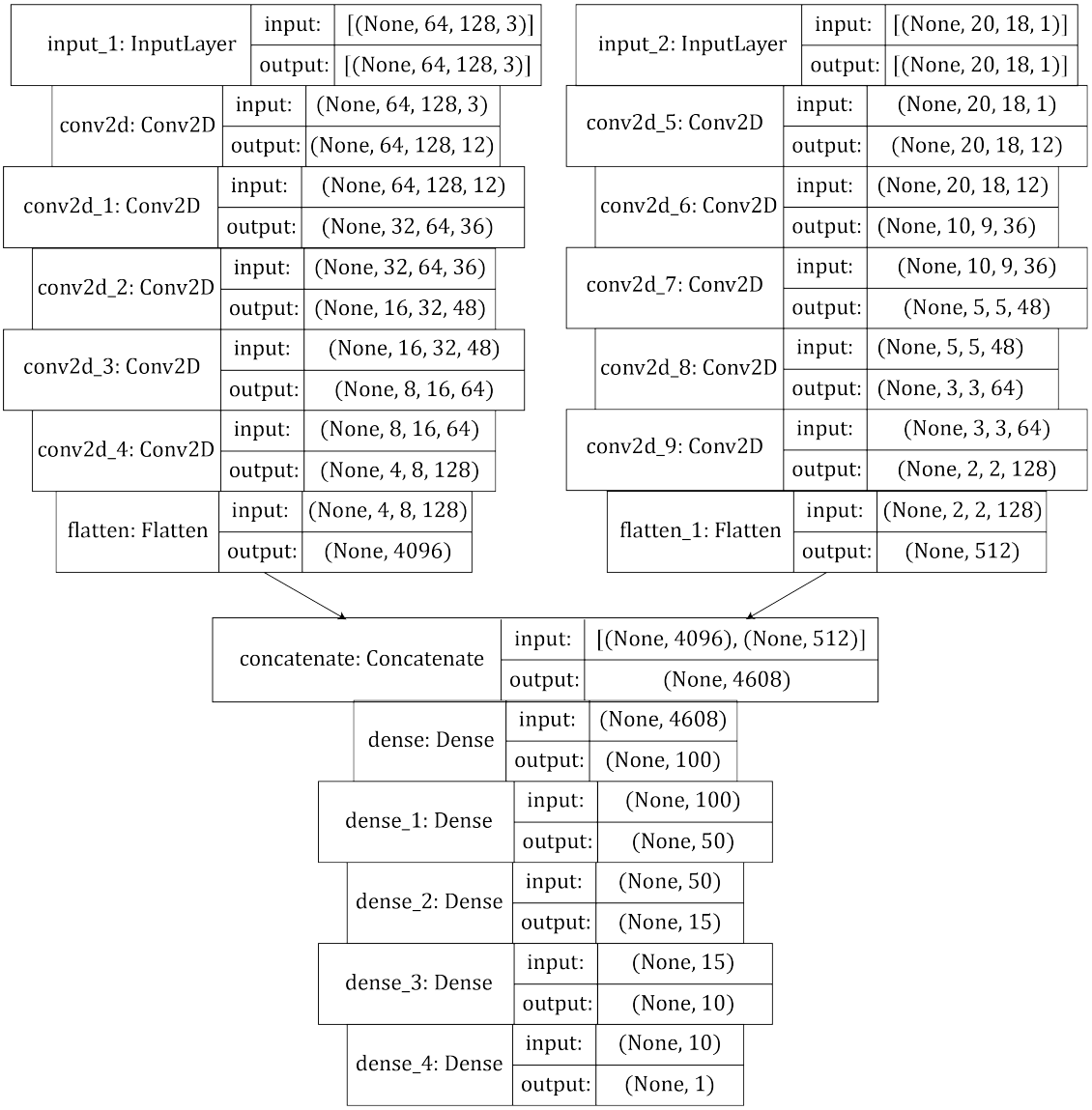


Figura 4.24: Red neuronal convolucional con dos entradas para el reto tres (Véase E.3)

4.3. Implementación

En esta sección se detallan las actividades implementadas en el prototipo físico, que analógicamente han sido realizadas en simulación. Sin embargo, algunas actividades difieren de la simulación ya que no se tiene un ambiente completamente aislado, por otro lado, hace

falta realizar algunas correcciones o cambios para lograr los objetivos. En consecuencia, se realizaron las siguientes tareas: Adquisición de la imagen, adquisición del ángulo de dirección y del sensor LiDAR, aunado a lo anterior, también se realizó la creación de dos bases de datos para el entrenamiento de dos modelos diferentes (CNN) que cumplan con cada uno de los dos retos: Navegación autónoma sin obstáculos y con obstáculos estáticos.

4.3.1. Prototipo

El prototipo empleado para las tareas de navegación es mostrado en la Figura 4.25, en ella se observa la posición de la cámara a través del cual se obtiene la imagen, esta cámara se encuentra en una posición adecuada para eliminar la mayor información innecesaria posible, y por consiguiente observar únicamente lo necesario, en ese sentido la cámara se encuentra a 17.5 cm sobre el suelo y a una inclinación negativa sobre la horizontal de 20°; aunado a lo anterior se observa el sensor LiDAR en el centro del chasis.

Figura 4.25: Posición de la cámara Orbeec Astra Pro en el prototipo

Las medidas del prototipo son 39 cm de largo, por 17 cm de ancho y 25 cm de altura con cámara, por otro lado la velocidad constante de desplazamiento a la cual se ajusta el prototipo para cada caso se detalla a continuación:

- Navegación autónoma sin obstáculos: 0,37 m/s
- Navegación autónoma con obstáculos estáticos: 0,37 m/s

4.3.2. Entorno de implementación

El entorno elegido para la navegación autónoma sin obstáculos es una adaptación del mostrado en la Figura 1.5, el cual a su vez es una representación del entorno usado en simulación de la Figura 4.7, la razón de esta acción es la de observar los resultados bajo un entorno similar entre ambos ambientes, con ello también observar las diferencias de manipular ambos entornos. Este entorno mide 7.11 m por 3.24 m, mientras que cada carril mide 0.42 m de ancho.

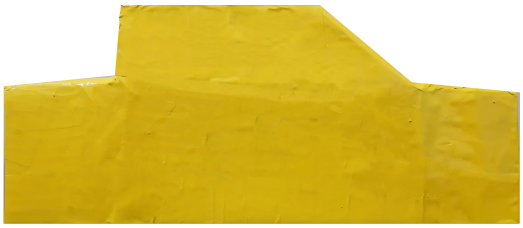
Por otro lado, el entorno controlado para la navegación autónoma con obstáculos estáticos es la misma que la utilizada para la navegación autónoma sin obstáculos, cuya diferencia reside únicamente en los obstáculos, este entorno es mostrado en la Figura 4.26, por un lado la Figura 4.26(a) para los obstáculos en el carril externo, y por el otro la Figura 4.26(b) para los obstáculos en el carril externo, la finalidad utilizar este circuito para ambas tareas es proveer el tiempo suficiente al prototipo para evadir ciertos obstáculos en el entorno y reincorporarse de nuevo a su carril.

(a) Obstáculos en carril externo

(b) Obstáculos en carril interno

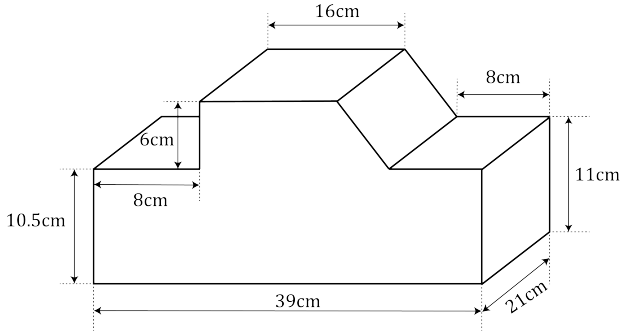
Figura 4.26: Circuito para la navegación autónoma con obstáculos

Aunado a lo anterior, se han agregado dos obstáculos estáticos de color amarillos similares a los empleados en simulación (Figura 4.8(c)), estos obstáculos se muestran sobre el circuito de la Figura 4.26, mientras que en la Figura 4.27 se observa a detalle estos obstáculos (Figura 4.27(a) y Figura 4.27(b)), su colocación en el circuito se encuentra exactamente a la mitad, y permanecen en esa posición en todo momento, las medidas de ambos obstáculos son 39 cm de largo por 21 cm de ancho, mientras que la altura total es de 17 cm, la Figura 4.27(c) describe a detalle las dimensiones.



(a) Vista lateral de obstáculo

(b) Vista isométrica de obstáculo



(c) Dimensiones del obstáculo

Figura 4.27: Obstáculo implementado en el entorno

4.3.3. Adquisición de datos

Para cumplir con los diferentes retos propuestos, es necesaria recopilar la información necesaria para el entrenamiento de las dos diferentes redes neuronales convolucionales para los dos diferentes retos, por lo tanto, la adquisición de datos al igual que en la simulación, se realiza a partir de la imagen obtenida a través de la cámara RGB, del mismo modo se obtiene la información del ángulo de dirección del prototipo. En el caso particular en el que se evaden obstáculos, es necesaria la información del sensor LiDAR.

De modo que, la adquisición comprende desde las técnicas de colocación de la cámara en el robot a una altura y ángulo adecuados y con una región de interés seleccionada, así como la adquisición del ángulo de dirección utilizando un mando de videojuegos, y por último, la manipulación de los datos del sensor LiDAR. No obstante, la creación de una base de datos para cada uno de los dos retos se realiza con la información obtenida. Por lo tanto, la adquisición de datos se divide en tres principales partes:

- Adquisición de la imagen
- Adquisición del ángulo de dirección
- Adquisición de la distancia de objetos mediante láser (LiDAR)

4.3.3.1. Adquisición de la imagen

La obtención de la imagen es el resultado a partir de la implementación de la cámara Orbeec Astra Pro que se encuentra montada en la parte superior del prototipo con la vista hacia el frente, en el punto exacto en el que ya no se observa el chasis del prototipo, mostrado en la Figura 4.25, cuyas medidas son de 640x480 pixeles en un formato RGB de 8 bits, con frecuencia de actualización de 30 Hz y con un campo de visión de 1,04 radianes (del mismo modo que en la cámara en simulación). El uso de esta cámara tiene dos funciones, la adquisición de imágenes para la creación de la base de datos y posteriormente la predicción en tiempo real para el movimiento del prototipo.

Región de interés. El uso de seleccionar la región de interés cumple con el objetivo de eliminar información innecesaria sobre la imagen, en este caso el fondo que se puede apreciar

al horizonte. En consecuencia, la región de interés es decidida a partir de la posición de la cámara de la Figura 4.25, de esta manera al obtener la imagen y decidir un área de interés de la Figura 4.28(a), se obtiene lo mostrado en la Figura 4.28(b), el área de interés seleccionada comprende de 50 pixeles menos, de la parte superior verticalmente a partir de la imagen original.

(a) Imagen obtenida por la cámara (b) Área de interés

Figura 4.28: Imagen original y área de interés

4.3.3.2. Adquisición del ángulo de dirección

El prototipo al igual que en la simulación cuenta con un controlador para asignar el ángulo de dirección, en consecuencia, la Figura 4.29 muestra el rango en el que el ángulo de dirección puede asignarse, por lo que se encuentra entre -22° y 22° aproximadamente. Sin embargo, dado el formato necesario para utilizar los ángulos, estos datos se deben convertir a cantidades de Modulación por Ancho de Pulso (PWM, por sus siglas en inglés), los cuales se encuentran entre 1000 y 2000, donde 2000 corresponde al ángulo máximo que puede girar a la derecha y 2000 el ángulo máximo hacia la izquierda.

Para obtener los ángulos de dirección en el tiempo y que a su vez el prototipo utiliza para desplazarse, se emplea el mando alámbrico utilizado para la simulación mostrado en la Figura 4.30 de la marca Xbox®, y a diferencia de la simulación, en este caso se utilizan dos botones, el botón **Y** y el botón **A**, en este caso el joystick **L** no es empleado.

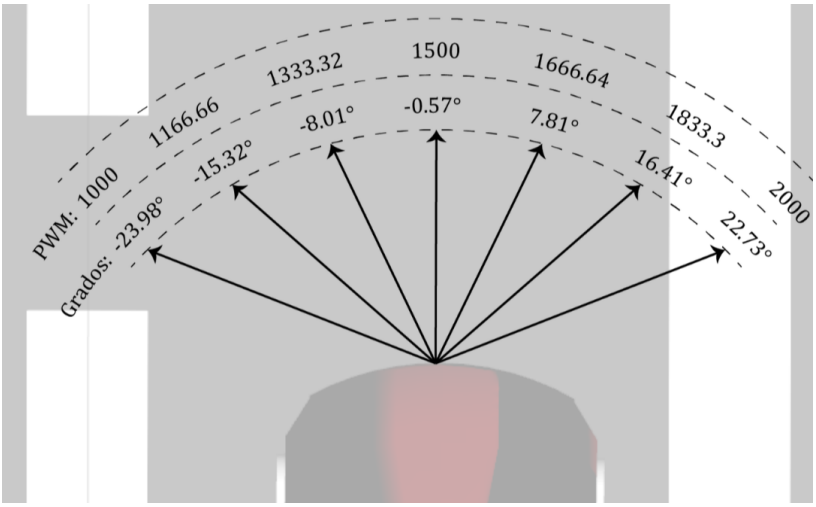


Figura 4.29: Parametrización del ángulo de dirección en el prototipo real

La Figura 4.30 muestra los botones y joystick utilizados para el movimiento del prototipo real, el joystick **R** realiza el movimiento del ángulo de dirección de izquierda a derecha, mientras que el botón **1** realiza la tarea de asignar una velocidad constante al modelo físico, y por lo tanto, el botón **2** es empleado para que el prototipo se detenga. Por lo tanto, estos dos botones únicamente cumplen la función de inicio y finalización del movimiento respectivamente.

Figura 4.30: Mando para el movimiento del modelo en la implementación

En consecuencia, se ha utilizado un programa capaz de realizar las actividades de adquisición de datos por parte del mando, este programa es desarrollado con ROS, por lo que se utiliza una librería genérica para su movimiento, el programa completo puede ser consultado en C.2. Por lo tanto, el ángulo de dirección es utilizado para crear la base de datos, estos datos son generados y guardados en una frecuencia de 20 Hz (al igual que la imagen), mientras que el prototipo es desplazado a una velocidad de 0,37 m/s en ambos casos de las tareas a cumplir, en el que no hay obstáculos y en el que sí hay obstáculos estáticos (mencionado en la sección 4.3.1).

4.3.3.3. Adquisición de la distancia de objetos mediante láser (LiDAR)

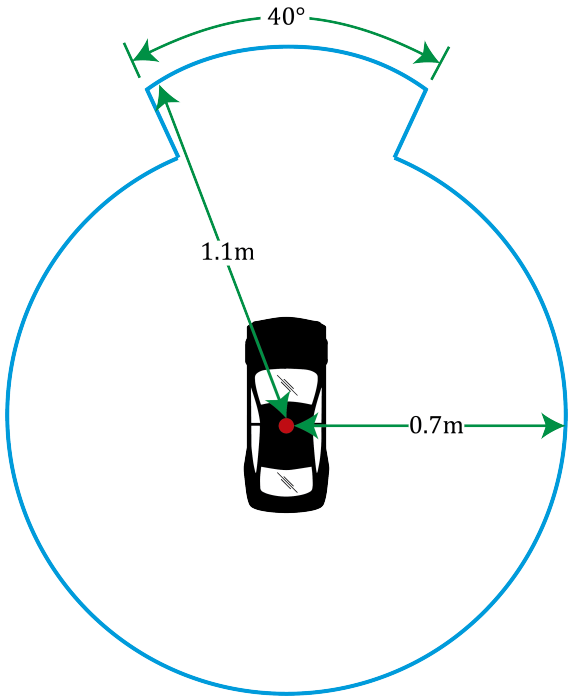


Figura 4.31: Medición de sensor LiDAR en el modelo real

El sensor para medir distancias en un radio de los 360° es un LiDAR, y al igual que en la simulación, tiene la capacidad de detectar objetos a partir de 0.15 m y no mayores a una distancia de 8 m, sin embargo, en este caso se ha reducido la distancia máxima frontal hasta 1.1 m en un ángulo de 40 grados, mientras que las mediciones restantes tienen una distancia máxima de 0.7 m, como se muestra en la Figura 4.31. El rango entre rayos láser es de 1°, por lo que el vector de resultado contiene 360 datos (resolución de una lectura individual por grado), en una frecuencia de 30 Hz.

Estas lecturas son necesarias para crear las bases de datos para las dos tareas a cumplir, por lo tanto, esta información es reunida mientras el prototipo se desplaza a una velocidad constante de 0,37 m/s, para ambas tareas (nav. autónoma sin obstáculos y con obstáculos), la información sobre la velocidad es mencionada en 4.3.1.

4.3.4. Base de datos para la navegación autónoma sin obstáculos

La creación de la base de datos se obtiene a partir de las imágenes generadas por el área de interés señalado en la Figura 4.28, y a su vez por el ángulo de dirección que el prototipo tenga en todo momento con ayuda del mando descrito en la Figura 4.30, esta información es necesaria para el caso en el que no hay obstáculos, y a su vez recabada en un tiempo aproximado de 20 minutos sin salirse del carril y en ambas direcciones, respetando el movimiento sobre el carril derecho. La velocidad de desplazamiento a la cual se realiza esta acción es de 0,37 m/s (mencionado en la sección 4.3.1).

La Figura 4.32 muestra el diagrama rqt generado por ROS con el pluging **rqt_graph** para la obtención de las imágenes y ángulos que serán grabados en un archivo tipo **rosbag**, los nodos se señalan con óvalos, mientras que los tópicos son rectángulos y el uso de cada uno se detalla a continuación:

- ***Nodos:***
  - **/joy_node**
    - Es el nodo encargado del enlace entre el mando y ROS.
  - **/ackermann_drive_joyop_node**
    - Nodo para mover al modelo del AutoNOMOS Mini desde el mando con sus joysticks y botones.
  - **/app/camera/camera_nodelet_manager**
    - Nodo encargado del enlace entre la cámara Intel y ROS.
  - **/app/camera/driver**
    - Nodo controlador de todas las configuraciones de grabado que la cámara Intel tiene disponible.
  - **/ImageCut**
    - Nodo que configura la imagen y escoge un área de interés.

- **/brushless_control_node**
  - Nodo controlador del motor brushless que funciona para la velocidad y arranque del modelo.
- **/motor_control_node**
  - Nodo controlador del motor de dirección.
- **/bldc_board_communication_node**
  - Nodo controlador de la comunicación entre el motor BLDC y ROS.
- **/VIDEO**
  - Nodo que contiene la imagen del área de interés.
- **/record**
  - Nodo generado a partir del rosbag para la grabación de los mensajes de imagen y ángulo.

- ***Tópicos:***
  - **/joy**
    - Tópico por el cual se mandan los mensajes en dos vectores del mando al software, en este caso, a ROS.
  - **/brushless_control/manual_control/speed**
    - Tópico del AutoNOMOS que recibe la velocidad proveniente del botón del mando y lo envía al modelo físico.
  - **/brushless_control/manual_control/stop_start**
    - Tópico del AutoNOMOS que recibe la orden de comenzar a moverse proveniente del botón del mando y lo envía al modelo físico.
  - **/manual_control/steering**
    - Tópico del AutoNOMOS que recibe la dirección de giro de las ruedas frontales del modelo proveniente del mando y lo envía al modelo físico.
  - **/app/camera/rgb/image_raw/compressed**
    - Tópico generado por la cámara Intel que envía las imágenes provenientes del mismo y las envía al nodo de /VIDEO e /ImageCut.
  - **/bldc_board_communicacion/led_back**
    - Tópico por el cual se pueden configurar los leds colocados en la parte posterior del chasis.
  - **/bldc_board_communicacion/led_front**

- Tópico por el cual se pueden configurar los leds colocados en la parte frontal del chasis.

- **/angulo**
  - Tópico generado por el mando que se está publicando constantemente y que se guarda en el rosbag para generar la base de datos.
- **/imageROI**
  - Tópico generado por el área de interés que se está publicando constantemente y que se guarda en el rosbag para generar la base de datos.

El archivo generado tras la grabación de los tópicos de **/VIDEO** e **/imageROI** es un archivo rosbag de extensión **.bag**, la extracción de la información proveniente de este archivo se realiza con la ayuda del script detallado en los pseudocódigos 1 y 2 que a su vez son empleados para el archivo rosbag de la simulación, y del mismo modo, en el primero se extraen las imágenes y ángulos, mientras que en el segundo se realiza el guardado de esta información.

La Figura 4.17 representa la forma de almacenamiento que se genera por el pseudocódigo 2 al igual que en la simulación, por un lado una carpeta que contiene 23'100 imágenes y por el otro un archivo **.xls** que representa las etiquetas de cada imagen, estos datos, como ya se comentó, equivalen a 20 minutos aproximadamente de movimiento continuo del prototipo físico en el entorno elegido a velocidad constante.

4.3.5. Base de datos para la navegación autónoma con obstáculos estáticos

A diferencia de la base de datos para la navegación autónoma sin obstáculos, y aunado a la información que ya se reúne en ese caso (imágenes del área de interés y ángulo de dirección), también es necesario almacenar la información del sensor LiDAR en el mismo tiempo que se reunen los datos de la cámara y el ángulo de dirección. Dicha información es obtenida mientras el prototipo es guiado empleando un mando y esquivando dos obstáculos amarillos en el carril en el que se encuentre navegando (carril interno y carril externo), esta información y al igual que sucede en la simulación, es almacenada en un archivo de ROS llamado rosbag.

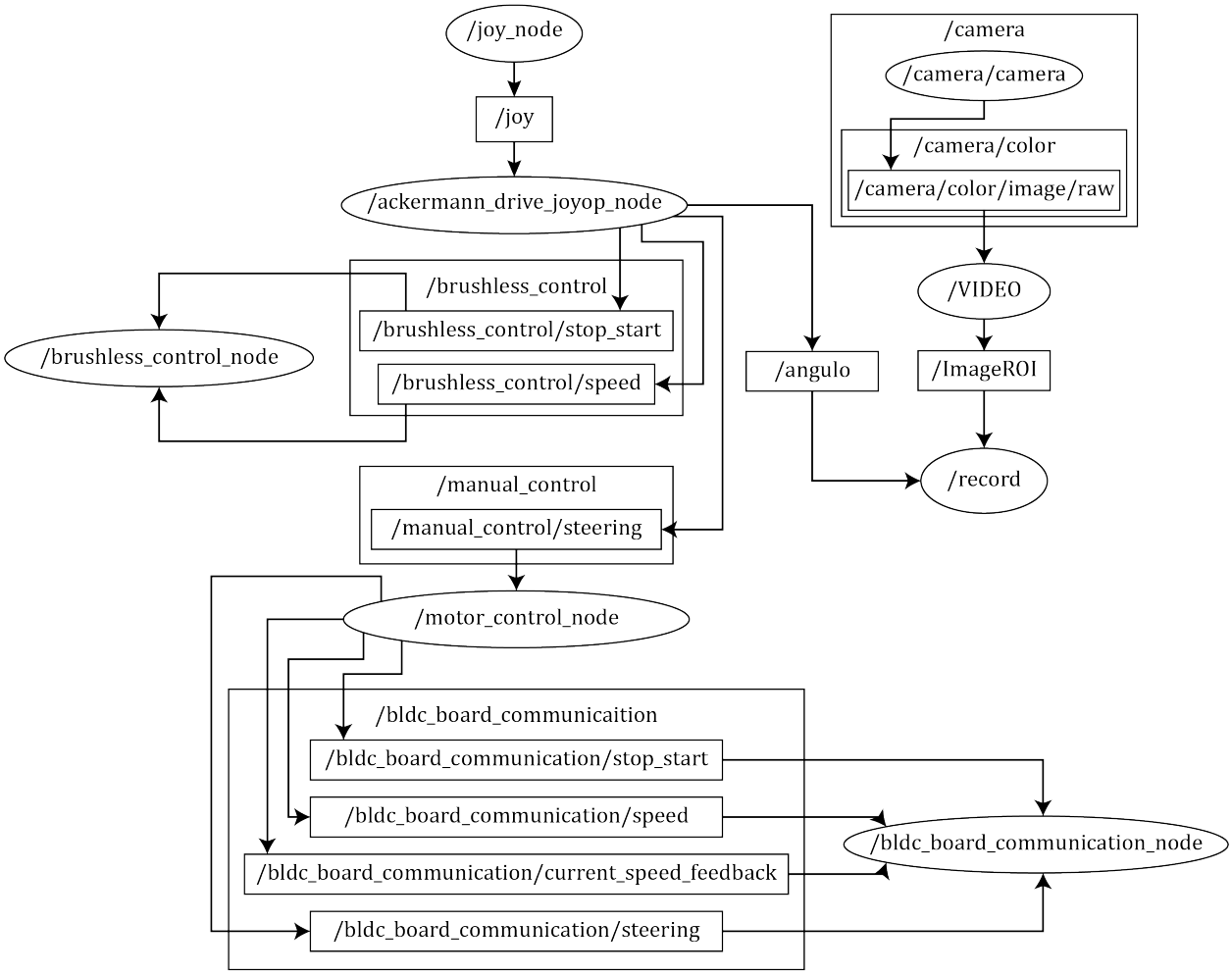


Figura 4.32: Gráfico rqt de tópicos y nodos del prototipo implementado para la adquisición de datos (imagen y ángulo de dirección)

Por lo tanto, la información fue reunida en un tiempo aproximado de 40 minutos y para ambos carriles, a una velocidad de desplazamiento constante de 0,37 m/s (mencionado en 4.3.1). La frecuencia para la imagen de la cámara y del ángulo de dirección es de 20 Hz, mientras que el sensor LiDAR emplea una frecuencia de 30 Hz.

El gráfico rqt de la comunicación entre nodos y tópicos que se mantiene mientras se graba la información de la cámara, ángulo de dirección y vectores del LiDAR se puede observar en la Figura 4.33. Del mismo modo que ocurre en la simulación, la base de datos se encuentra en un archivo con extensión **.bag**, y para extraer la información de este archivo se utiliza un script dividido en dos partes, extracción de información (Algoritmo 3) y guardado de

la base de datos (Algoritmo 4), para una mayor comprensión del programa computacional véase C.4.

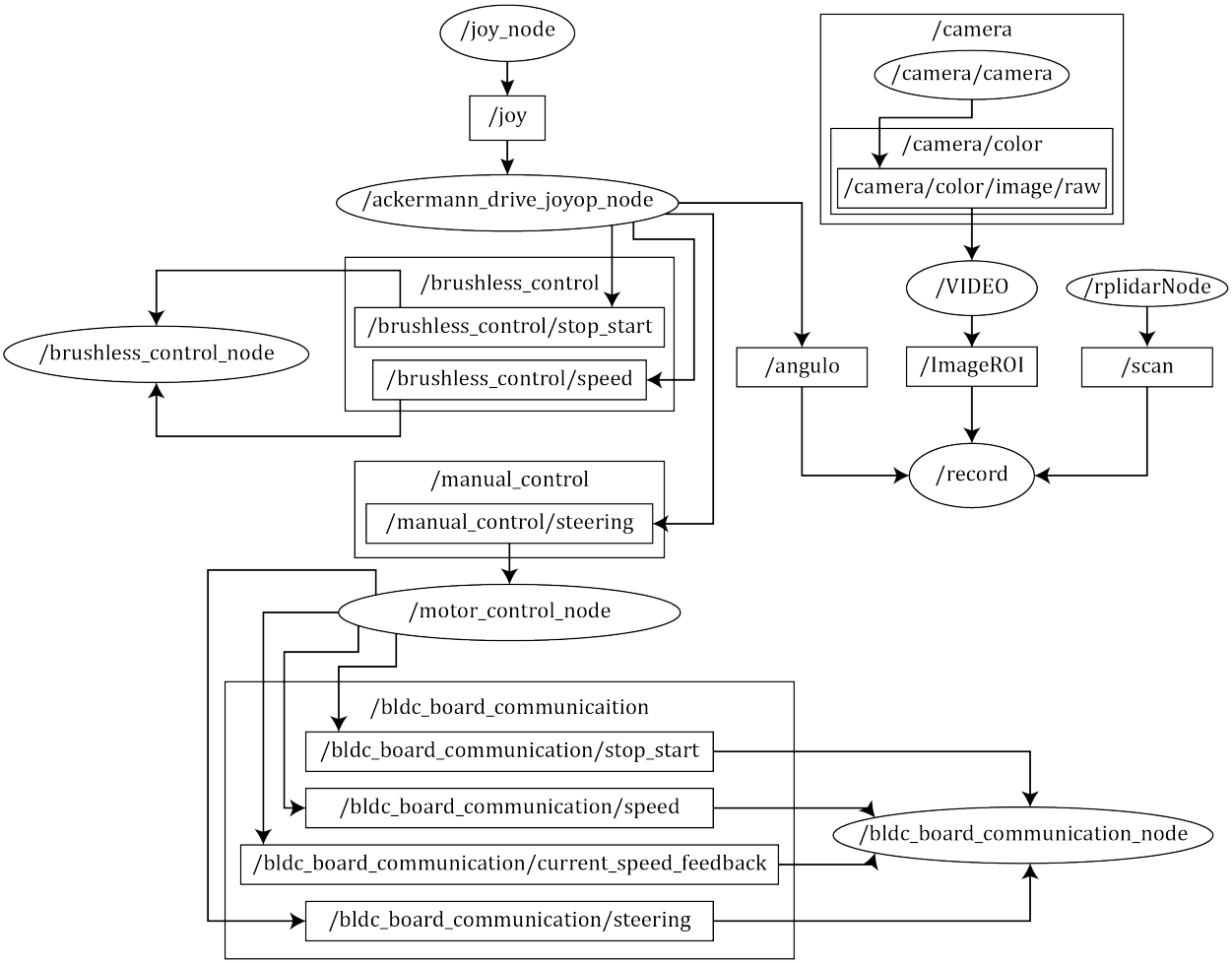


Figura 4.33: Gráfico rqt de tópicos y nodos del prototipo implementado para la adquisición de datos (imagen, LiDAR y ángulo de dirección)

La información del LiDAR está contenida en un archivo de hoja de cálculo en forma de vectores con 360 columnas por cada dato, se utiliza el mismo script usado en la simulación para convertir cada uno de los datos en imágenes de 20 pixeles de largo por 18 pixeles de ancho, donde el dato de mayor valor (1.1 m y 0.7 m) tiene un valor de 255, por otro lado, el dato con menor distancia (0 m) recibe un valor de 0, de un solo canal (escala de grises), el Algoritmo 5 describe esta tarea, mientras que la Figura 4.34 muestra el resultado sobre una imagen del LiDAR (Véase C.5 para el código computacional que realiza esta tarea).

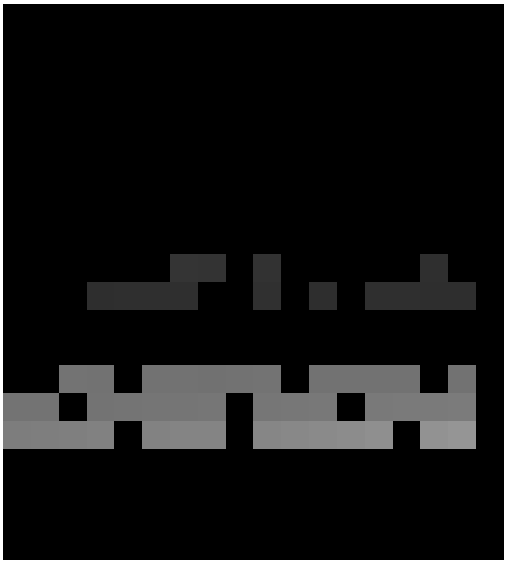


Figura 4.34: Vector de distancias LiDAR a imagen

4.3.6. Red neuronal convolucional para reto 1: Navegación autónoma sin obstáculo

La base de datos creada con las imágenes y ángulos resultado de controlar el modelo físico son la entrada y salida de la red neuronal convolucional (CNN) respectivamente, la red planteada es la mostrada en la Figura 4.35, esta CNN es una modificación de lo planteado e implementado en la simulación, específicamente en la CNN utilizada para la navegación sin obstáculos, detallado en la sección 4.2.7, que a su vez, es una variación de lo planteado por NVIDIA en la Figura 1.3.

Por lo tanto, la red CNN de la Figura 4.35 es la elegida para el movimiento del prototipo en la implementación física para la tarea de la navegación sin obstáculos, cuenta con 5 capas convolucionales que posteriormente se conecta a 4 capas que cumplen la tarea de regresión (Véase D.4 para una mayor representación del programa computacional). La decisión del uso de esta CNN es con la finalidad de obtener el menor error posible con los datos empleados en la base de datos.

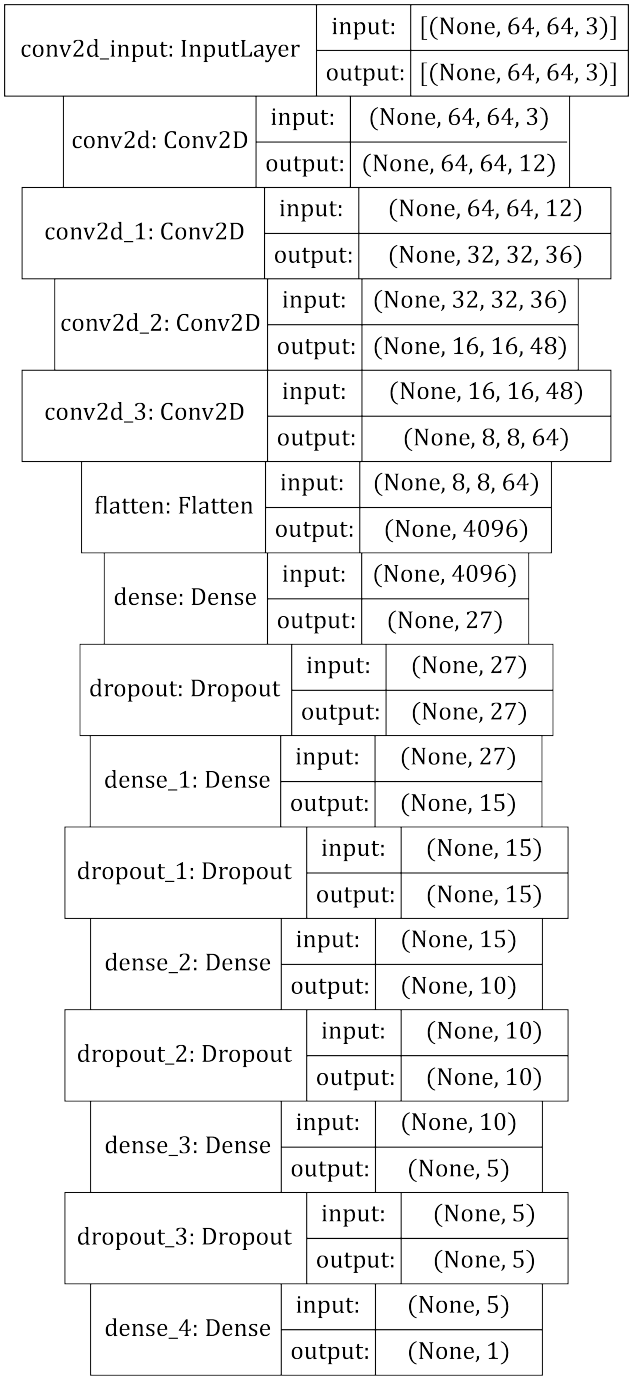


Figura 4.35: CNN para la dirección del modelo implementado (Véase E.4)

Capa	Tamaño		Mapas de características	Función de activación
	x	y		
conv1+elu	64	64	12	elu
conv2+elu	64	64	36	elu
conv3+elu	64	64	48	elu
conv4+elu	64	64	64	elu
conv5+elu	64	64	64	elu

Tabla 4.8: Parte convolucional de la CNN a entrenar para el movimiento del modelo.

La Tabla 4.8 representa la etapa convolucional de la red de la Figura 4.35, y es la misma que etapa que la utilizada en la **simulación**, sin embargo la diferencia radica en el regresor, Tabla 4.9, ya que la cantidad de neuronas por cada capa aumenta, esto debido a la complejidad de las imágenes a procesar.

Capa	Tamaño	Función de activación
flatten	262144	-
fc6+elu	20	elu
fc7+elu	15	elu
fc8+elu	10	elu
fc9+elu	1	-

Tabla 4.9: Parte neuronal de la CNN a entrenar para el movimiento del modelo.

4.3.7. Red neuronal convolucional para reto 2: Navegación autónoma con obstáculos estáticos

Del mismo modo que en el caso anterior (sección 4.3.6), la base de datos creada a partir de las imágenes de la cámara y sensor LiDAR son las entradas de la red neuronal convolucional (CNN), mientras que el ángulo de dirección es la salida, la red planteada se puede observar en la Figura 4.36. Esta CNN, como ya se mencionó, es una modificación del caso en el que no hay obstáculos, y a su vez, de lo planteado en el caso en el que hay obstáculos estáticos en simulación (sección 4.2.8). Esto otorga una visión para hallar el menor error posible en su entrenamiento.

Por lo tanto, la imagen proveniente de la cámara pasa por 6 capas convolucionales y la imagen del LiDAR pasa por 5 capas convolucionales antes del aplanado, después de ello se concatena en un vector más grande (1'600 datos), para finalmente pasar por 7 capas completamente conectadas.

Capa	Tamaño		Mapas de características	Función de activación
	x	y		
Img_Cam	64	64	3	elu
conv1+elu	64	64	12	elu
conv2+elu	32	32	36	elu
conv3+elu	16	16	64	elu
conv4+elu	8	8	128	elu
conv5+elu	4	4	150	elu
conv6+elu	2	2	200	elu

Tabla 4.10: Sección convolucional de la CNN, parte superior (imagen de la cámara).

La Tabla 4.10 muestra la etapa convolucional de la red propuesta para la entrada de la imagen de la cámara, con dimensiones 64x64x3 pixeles y pasando por 5 capas convolucionales hasta tener una dimensión de 2x2 y 200 mapas de características, mientras que la Tabla 4.11 muestra la etapa convolucional de la red para la entrada de la transformación del LiDAR a imagen cuyas dimensiones son 18x20x1 y pasando por 5 capas convolucionales para finalmente tener una dimensión de 2x2 y 200 mapas de características, misma dimensión que el resultado de la primer parte.

Capa	Tamaño		Mapas de características	Función de activación
	x	y		
Img_Lidar	18	20	1	elu
conv1+elu	18	20	12	elu
conv2+elu	9	10	36	elu
conv3+elu	5	5	64	elu
conv4+elu	3	3	128	elu
conv5+elu	2	2	200	elu

Tabla 4.11: Sección convolucional de la CNN, parte inferior (imagen del sensor lidar).

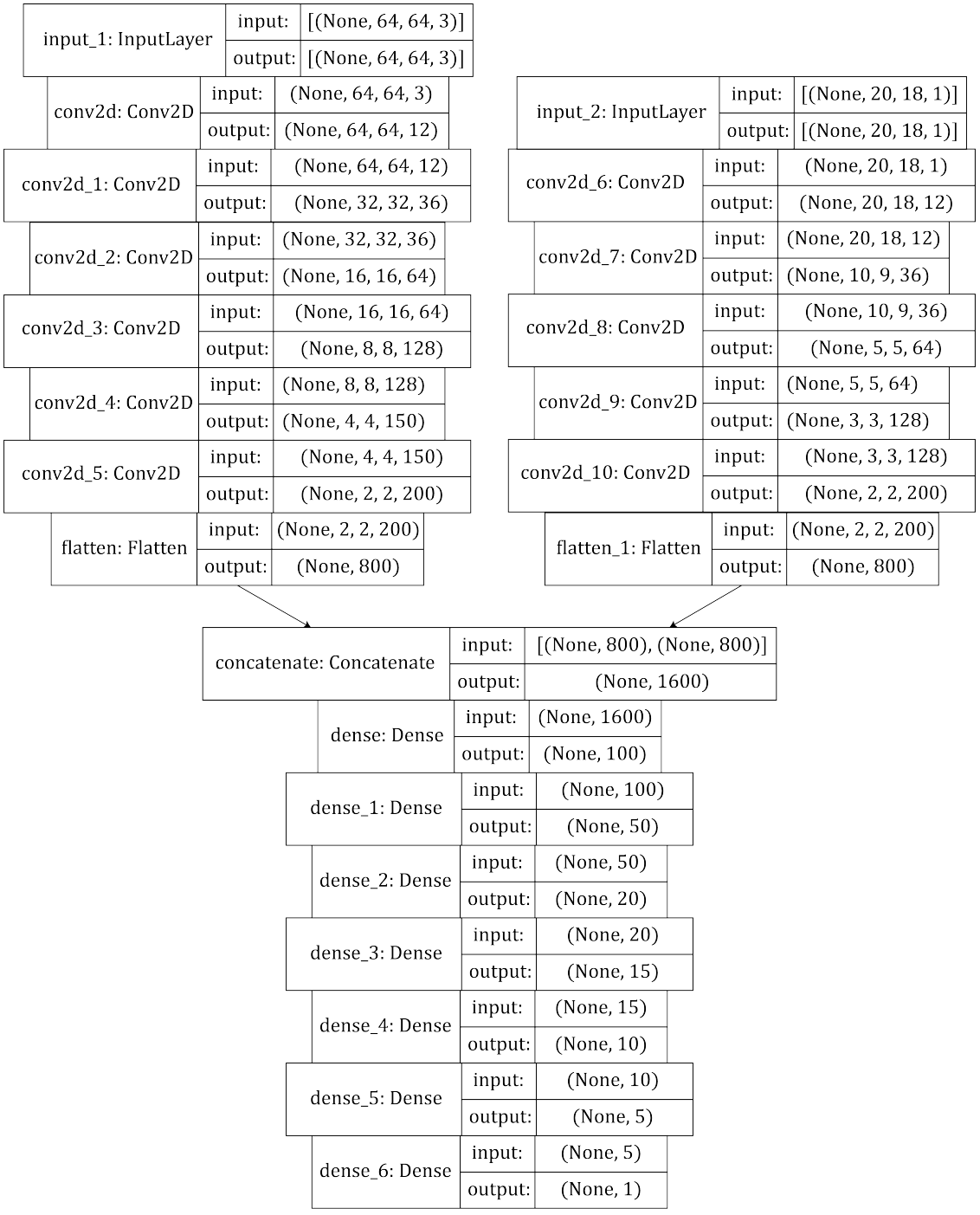


Figura 4.36: CNN para la predicción de dirección con obstáculos (Véase E.5)

Capa	Tamaño	Función de activación
flatten	1600	-
fc1+elu	100	elu
fc2+elu	50	elu
fc3+elu	20	elu
fc4+elu	15	elu
fc5+elu	10	elu
fc6+elu	5	elu
fc7	1	elu

Tabla 4.12: Parte neuronal (completamente conectado) de la CNN después de la concatenación

Capítulo 5

Resultados de entrenamiento

En este capítulo se emplea la información reunida de las bases de datos creadas en el capítulo de Desarollo, por lo tanto, se cuenta con cinco bases de datos, tres bases de datos para simulación y dos bases de datos para la implementación física, el objetivo dependerá de cada base de datos para la que fue creada. La Tabla 5.1 muestra el resumen de la información contenida en cada base de datos que se empleará en este capítulo. Por otro lado, para estas actividades se utilizo una CPU Intel Core i5 de novena generación, aunado a esto, también se empleó una GPU Nvidia GTX 1650 de 4 GB, además se contó con 24 GB en memoria RAM.

Base de datos	Datos(imágenes, ángulos y/o LiDAR)
Reto 1 (Simulación)	10'515
Reto 2 (Simulación)	42'419
Reto 3 (Simulación)	62'283
Reto 1 (Real)	23'100
Reto 2 (Real)	78'394

Tabla 5.1: Tamaño de cada base de datos

El entrenamiento y sus resultados se dividen en dos partes: El entrenamiento correspondiente a las tres CNN para los tres casos en simulación y el entrenamiento correspondiente a dos CNN para los dos casos de implementación física, en cada caso se muestran gráficas del error empleado, el Error Cuadrático Medio (MSE) entre los datos de entrenamiento y

los datos de prueba, por lo tanto, estos datos se encuentran divididos con un 90 % del total para entrenamiento y un 10 % de los datos restantes para la prueba.

5.1. Entrenamiento de las CNN para simulación

Para esta sección, los entrenamientos de simulación se dividen en tres partes, y cada una corresponde a cada uno de los retos, por lo que cada reto cuenta con su propia base de datos, una CNN y en cada caso, se tienen gráficas que corresponden al entrenamiento de estas CNN.

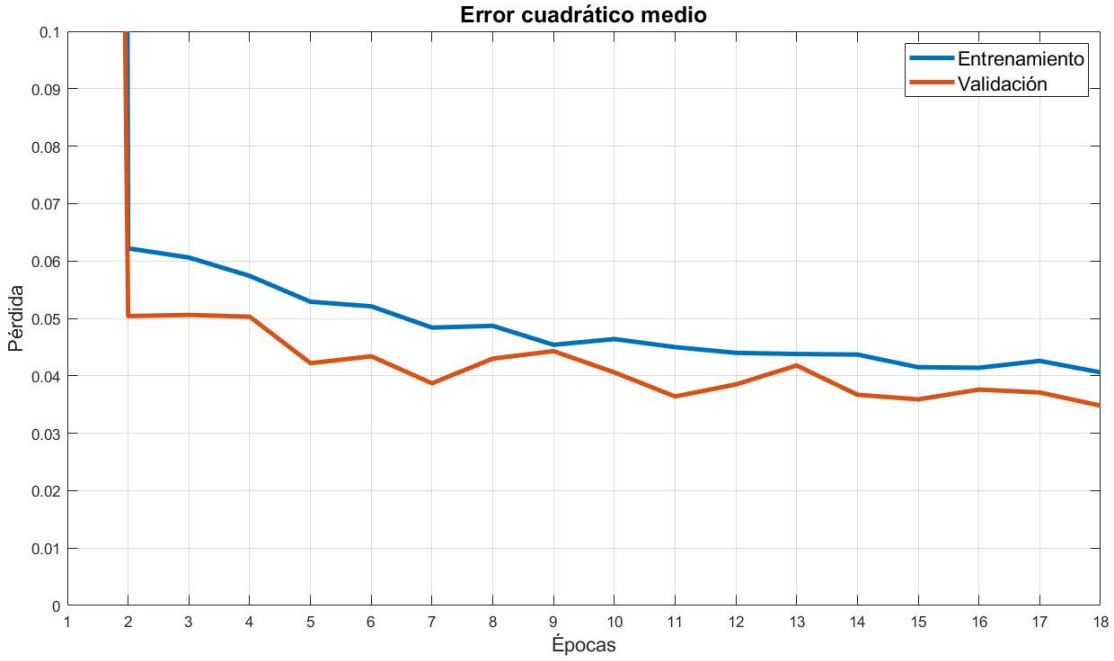


Figura 5.1: Gráfica de pérdida de la simulación para reto uno

Los resultados de simulación para la navegación autónoma sin obstáculos se dan a partir de entrenar la red neuronal convolucional de la Figura 4.22, utilizando como métrica de validación el error cuadrático medio por cada época de entrenamiento, la Figura 5.1 representa los resultados de utilizar esta métrica, obteniendo el menor error de 0,0187 en la validación y 0,021 en los datos de entrenamiento, tras 23 épocas de entrenamiento y utilizar 8351 imágenes con su respectiva etiqueta del ángulo, los datos de entrenamiento se divide en 66 lotes que se presentan en cada época a la red. Los parámetros de la red neuronal de la Figura 5.1

al ser entrenada se tiene en total 283′563 parámetros. Por otro lado, la Tabla F.1 muestra diferentes pruebas con diferentes neuronas en la sección del regresor de la CNN con las que se obtienen los mejores resultados, marcado con negritas la que se elige para la navegación autónoma en la simulación que tiene el menor error cuadrático medio.

Los resultados de la CNN en simulación para la navegación autónoma con obstáculos estáticos son a partir de entrenar la CNN con dos entradas de la Figura 4.23 y utilizando la métrica de validación el error cuadrático medio en cada época del entrenamiento, dan como resultado lo mostrado en la Figura 5.2. El menor error se obtiene después de 19 épocas, siendo este de 0.0101 en la validación, mientras que para los datos de entrenamiento es de 0.0145, este entrenamiento fue realizado utilizando 38178 imágenes de la cámara y 38178 imágenes del lidar, ambos respectivamente con el ángulo de dirección, divididos en lotes de 381 datos para cada época del entrenamiento. De acuerdo con el entrenamiento, los parámetros, entradas y salidas de la CNN, se tiene en total 384'602 parámetros. La Tabla F.2 muestra algunas pruebas con diferente cantidad de neuronas en la parte del regresor que ayudaron a encontrar el menor error.

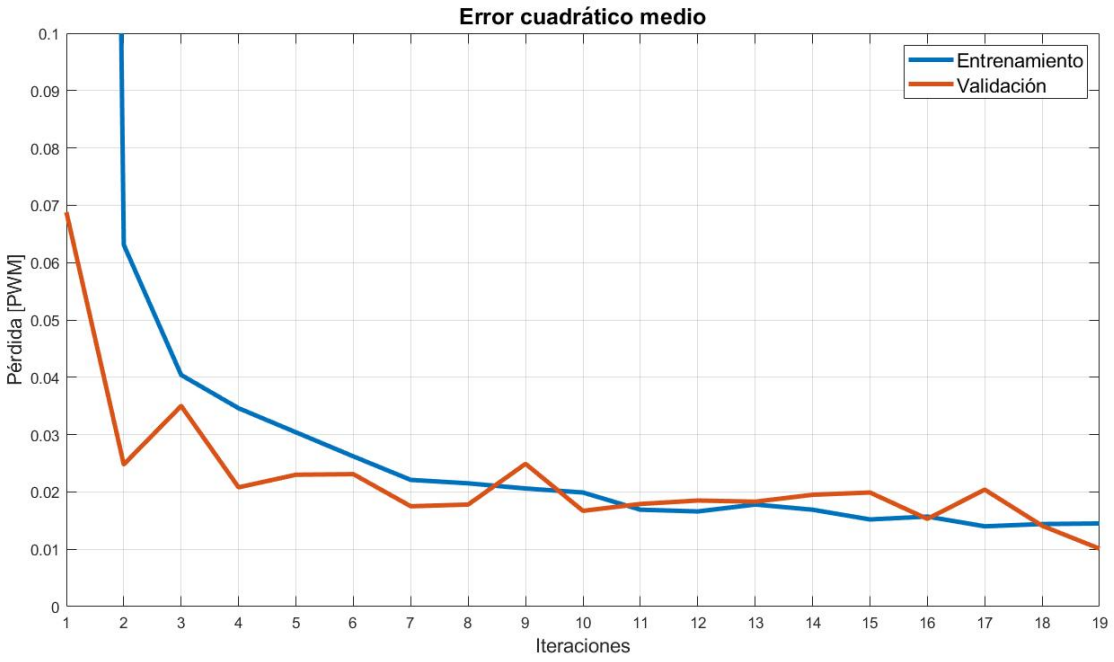


Figura 5.2: Gráfica de pérdida de la simulación para reto dos

Los resultados del entrenamiento en el que se tienen obstáculos móviles son a partir de entrenar la base de datos para este reto, por lo tanto se hace uso de la CNN mostrada en

la Figura 4.24 para entrenarla, por consiguiente, la gráfica del error puede ser observada en la Figura 5.3, en ella se observa que el menor error de entrenamiento alcanzado es de 0,0097, mientras que el error de validación es de 0,0103. Por otro lado, la Tabla F.3 muestra algunas pruebas extra antes de encontrar el mejor resultado, se muestran cuatro que resultan significativas también para observar el comportamiento en la búsqueda del menor error, eso se logra con 186 neuronas en la parte del regresor de la CNN.

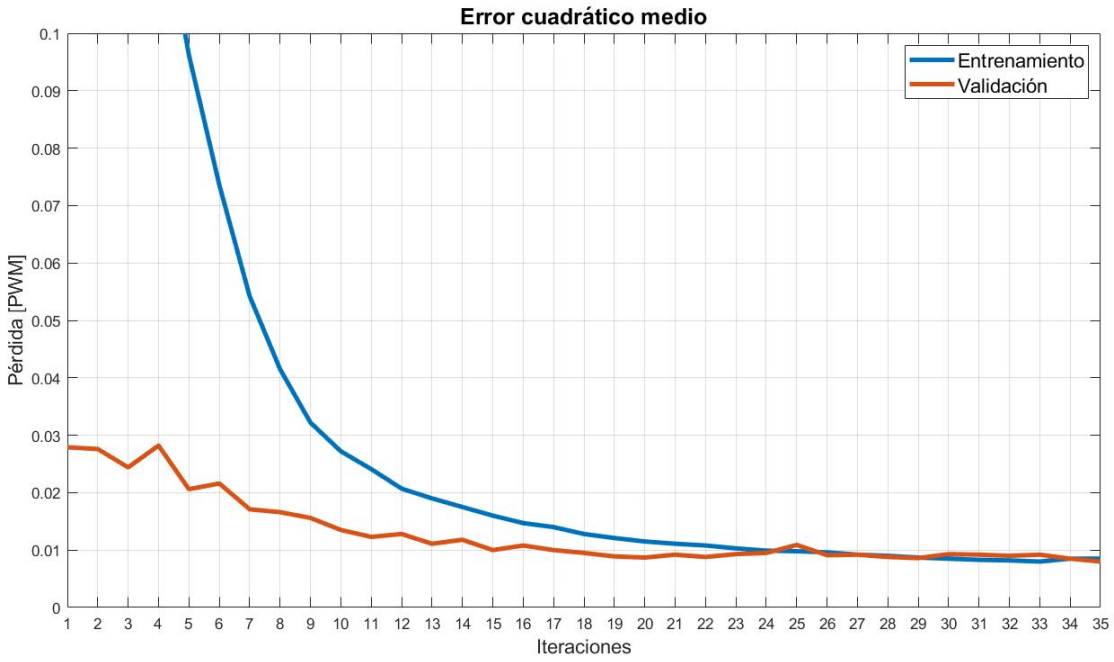


Figura 5.3: Gráfica de pérdida de la simulación para reto tres

5.2. Entrenamiento de las CNN para implementación física

En esta sección, a diferencia de la anterior, se describen dos entrenamientos de dos CNN, que corresponden a cada uno de los retos, por lo tanto, en cada caso se cuenta con su propia base de datos, la CNN a entrenar, y por lo tanto, las gráficas de los resultados de pérdida (error).

Los resultados de entrenar una red neuronal convolucional con una base de datos con 8391 imágenes y sus ángulos se pueden observar en la Figura 5.4, donde se observa que tras 18 épocas se obtienen los mejores resultados, con un error de 0,0348 para los datos de validación y un error de 0,0406 para los datos de entrenamiento, los datos de entrenamiento se divide en 70 bloques presentados a la CNN. El total de parámetros que se tiene de la red de la Figura 5.4 al ser entrenada, son 447822 parámetros en total. La Tabla F.4 muestra las pruebas más significativas de utilizar diferente cantidad de neuronas en el regresor de la CNN, comienza con 161 neuronas y se reduce esta cantidad como en la simulación hasta hallar el menor error posible, marcado en negritas.

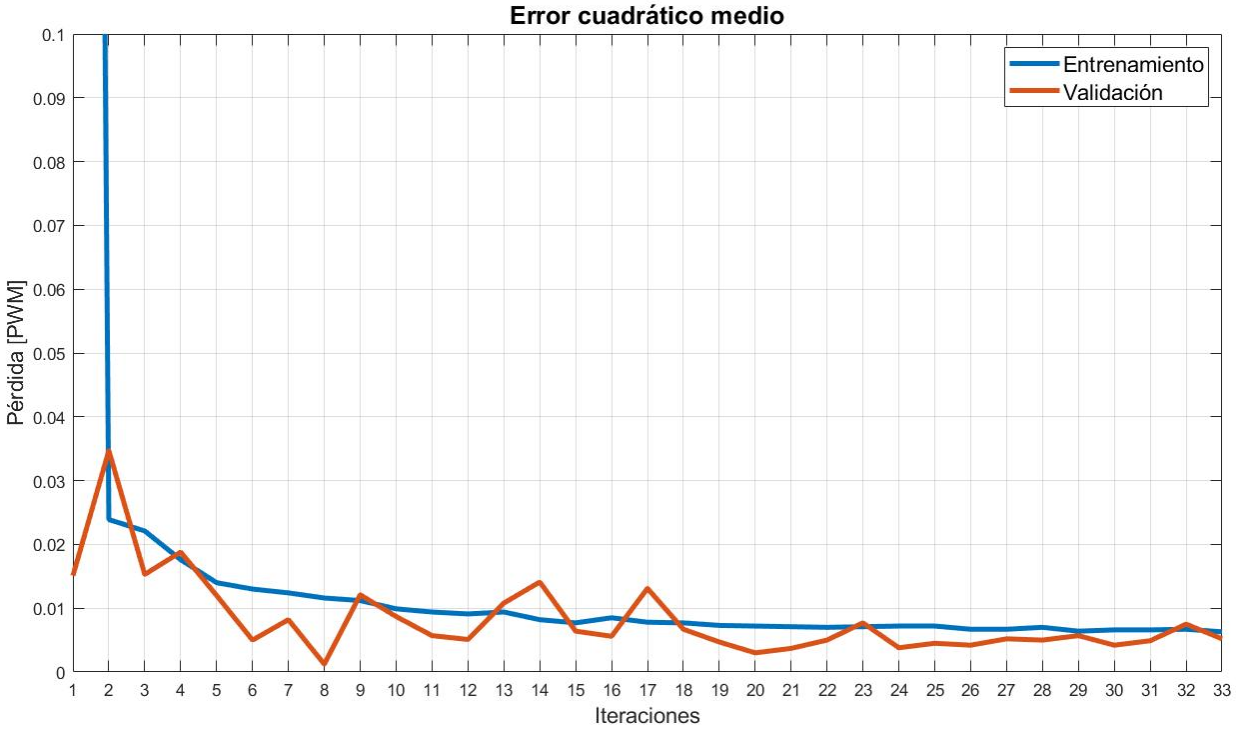


Figura 5.4: Gráfica de pérdida de la implementación para el reto uno

Los resultados de implementación para la navegación autónoma con obstáculos estáticos después de entrenar la CNN de dos entradas y una salida de la Figura , se muestran en la Figura, cuya gráfica representa el error cuadrático medio para los datos de entrenamiento (azul) y los datos de prueba (rojo), separados de la base de datos: 90 % entrenamiento y 10 %prueba de un total de 78'394 datos, el mejor error conseguido se resuelve tras 33 épocas de entrenamiento, obteniendo un error de 0.0063 para el entrenamiento y 0.0052 para los datos de prueba.

El total de parámetros usados es de 2'585'336 para esta prueba. Por otro lado, la Tabla F.5 muestra algunas pruebas con diferentes parámetros, esto con la intención de hallar el menor error posible.

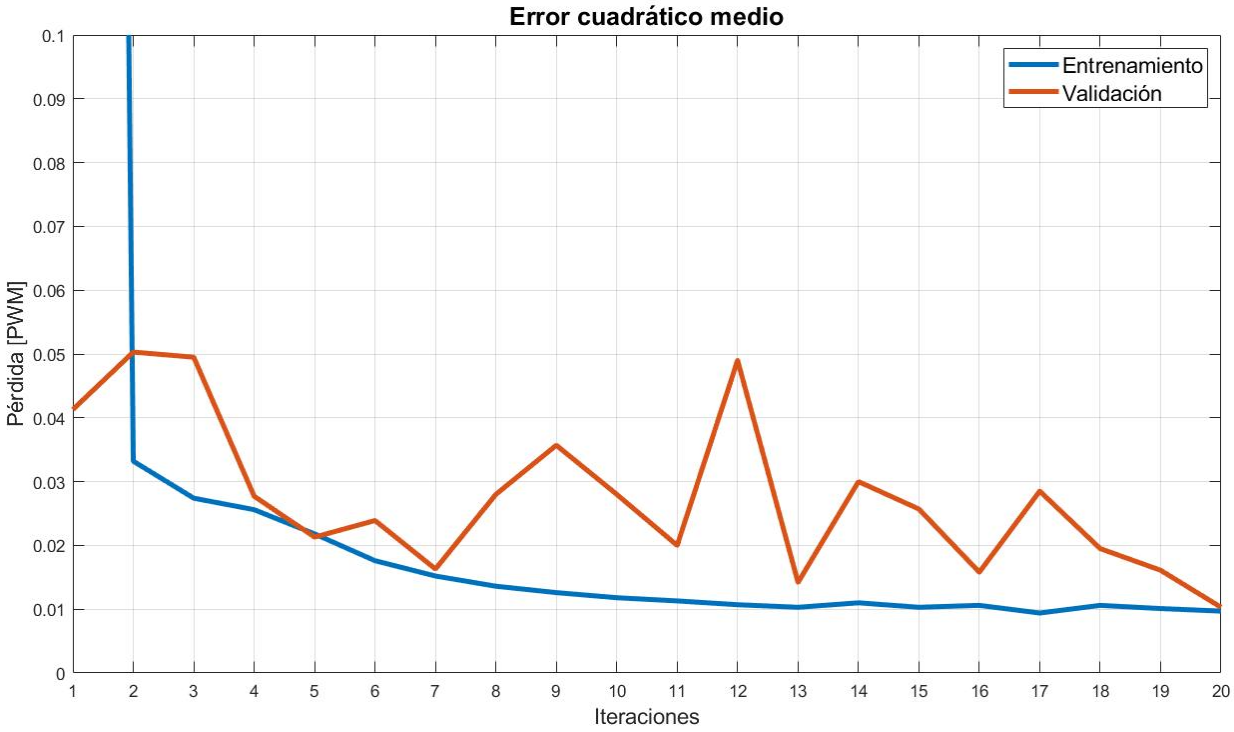


Figura 5.5: Gráfica de pérdida de la implementación para el reto dos

Capítulo 6

Resultados de navegación autónoma

En este capítulo se presentan los resultados obtenidos después de entrenar las redes neuronales convolucionales (CNN) para cada uno de los retos, con la finalidad de realizar navegación autónoma en el prototipo AutoNOMOS. Estas actividades han sido implementadas tanto en simulación como físicamente, en consecuencia, esta sección muestra gráficamente la predicción de cada CNN al dar una vuelta, ya sea con o sin obstáculos, comparándola con el ángulo deseado en cada momento, y observar gráficamente la fidelidad de las predicciones de las CNN con lo que se desea que realice el prototipo en cada caso (navegar automáticamente, y/o evadir obstáculos), sin embargo, lo deseado solo es una manera indirecta de realizar el movimiento, es decir, no necesariamente debería tener el mismo comportamiento, y con ello se demuestra que los modelos predictivos logran no salirse del circuito, manteniendo su propio comportamiento. Análogamente, los resultados se dividen de la siguiente manera:

- Simulación:
  - Navegación autónoma sin obstáculos (Topología de la Figura 4.22)
  - Navegación autónoma con obstáculos estáticos (Topología de la Figura 4.23)
  - Navegación autónoma con obstáculos móviles (Topología de la Figura 4.24)
- Implementación física:
  - Navegación autónoma sin obstáculos (Topología de la Figura 4.35)
  - Navegación autónoma con obstáculos estáticos (Topología de la Figura 4.36)

La Figura 6.1 muestra el circuito empleado tanto para el entrenamiento como para la comprobación de la navegación autónoma sin obstáculos, con obstáculos estáticos y con obstáculos móviles, en la ilustración también se pueden observar algunas etiquetas que han sido colocadas como referencia para detallar de mejor manera los resultados explicados a lo largo del capítulo, por lo que las curvas han sido etiquetadas con letras (O1, O2, O3 y O4), mientras que el inicio y final de cada prueba se encuentra en la parte superior central del circuito, siendo el mismo para el carril interno como para el carril externo, los obstáculos estáticos (O1, O2, O3 y O4) se encuentran en el mismo lugar para ambos carriles, y los obstáculos móviles son señalados con carros amarillos (OM1 y OM2), justo en la posición en la que sucede el rebase.

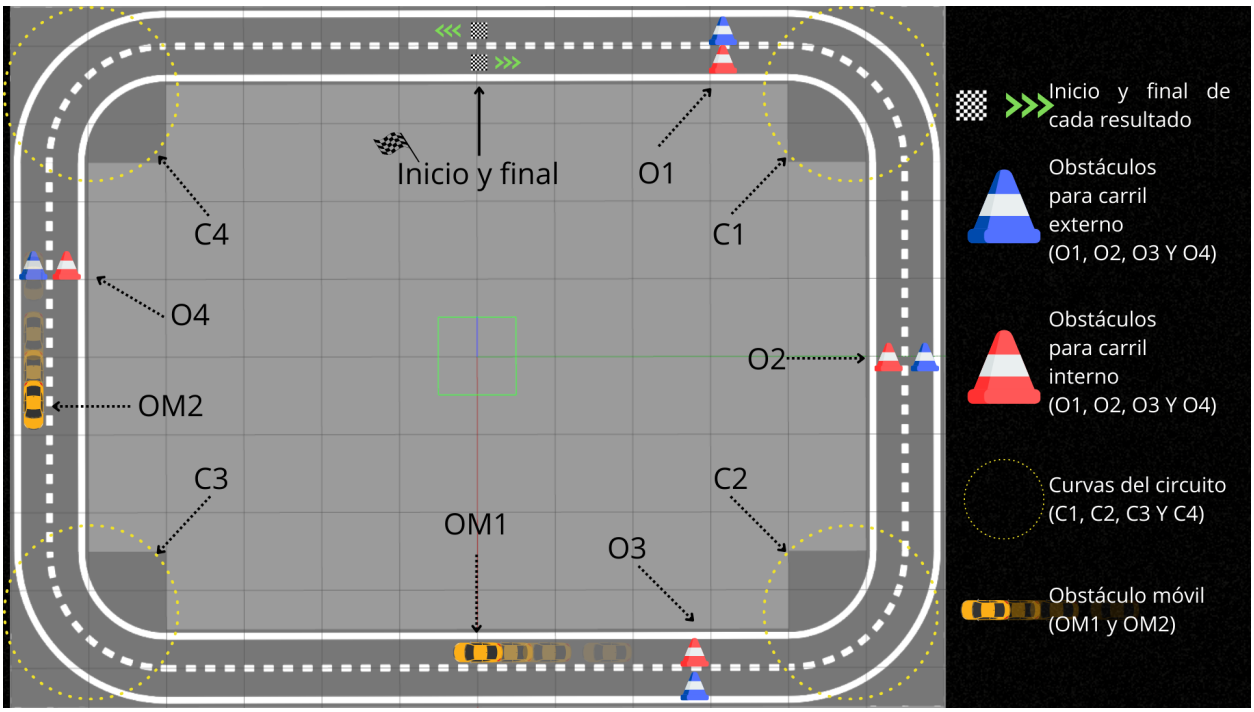


Figura 6.1: Circuito empleado para los resultados en simulación

Por otro lado, la Figura 6.2 muestra el circuito real empleado en la implementación física (independiente al mostrado en la Figura 6.1), en ella se observan cuatro curvas (C1, C2, C3 y C4), y a diferencia del circuito de simulación, en este caso se emplean únicamente dos obstáculos (O1 y O2) debido a la corta distancia entre una curva y otra, además del espacio antes y después del obstáculo para poder evadirlo, los obstáculos son colocados en el carril en el que se encuentra navegando el prototipo. En lo que se refiera al inicio y final de cada carril, estos se encuentran en diferentes sitios, para el carril interno el obstáculo se encuentra justo después de C4, y para el carril externo se encuentra después de C1.

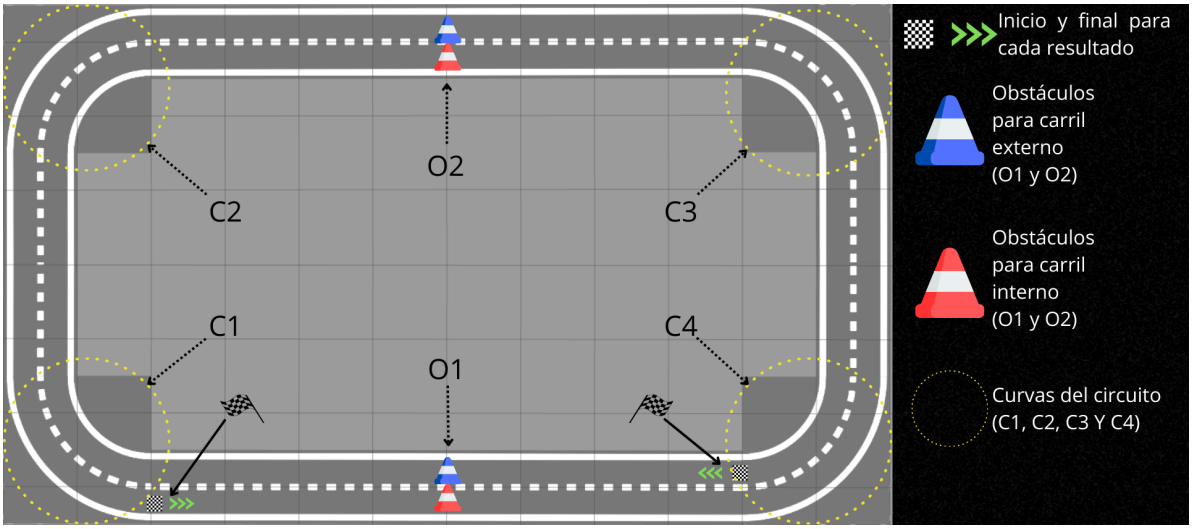


Figura 6.2: Circuito empleado para los resultados en implementación real

Los resultados están representados en gráficas, y para todos los casos el eje de la abscisa (eje x) representa el tiempo que le toma realizar una vuelta al prototipo en segundos, por otro lado el eje de la ordenada (eje y) representa el PWM para realizar un giro hacia cualquier dirección, en este caso el valor para los resultados de simulación se encuentran entre 180 (giro a la izquierda) y 0 (giro a la derecha). Por otro lado, en el caso de la implementación real, los valores de PWM se encuentran entre 1000 (giro a la izquierda) y 2000 (giro a la derecha).

6.1. Resultados sin obstáculos en simulación

La gráfica mostrada en la Figura 6.3 señala el resultado de la navegación autónoma sin obstáculos para una sola vuelta (simulación), y en el carril interno del circuito, en relación a ello, los valores en color rojo representan la predicción que realiza la CNN en todo momento, mientras que la línea punteada negra representa los valores que convencionalmente se desea que realice el prototipo (al mismo tiempo que la CNN realiza la predicción del ángulo), estos datos se encuentran en la base de datos, es decir, forma parte del manejo que se realizó con el mando para crear cada base de datos. En esta misma ilustración se observan cuatro grandes zonas etiquetadas (C1, C2, C3 y C4), las cuales representan las cuatro curvas, mismas con las que cuenta el circuito, además, se observa que los datos tienen una mayor concentración de PWM entre valores de $0 - 90$, lo que significa que el prototipo giró únicamente a la derecha, indicando que las curvas, en este caso específico se encuentran hacia la derecha del prototipo.

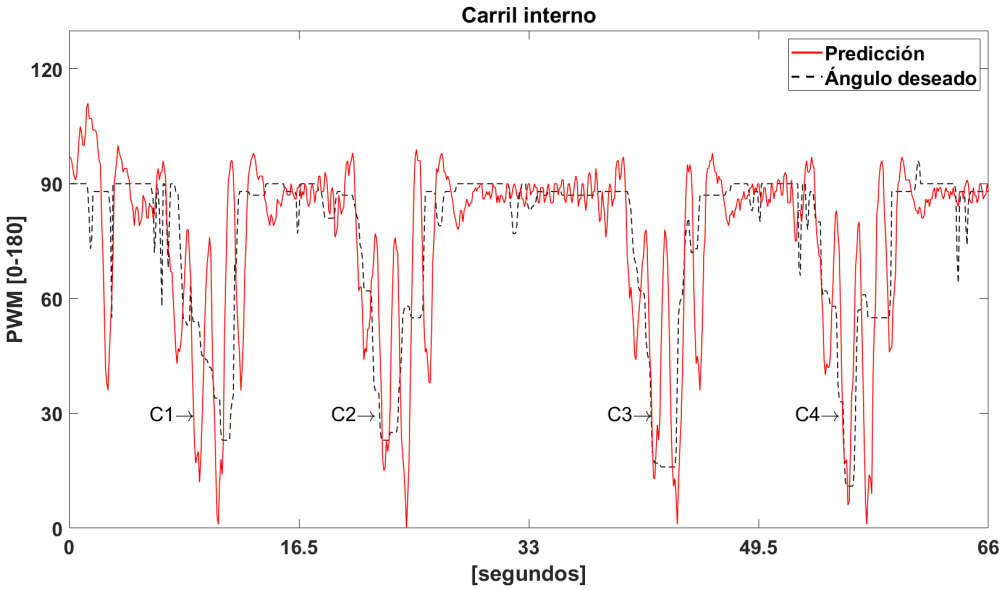


Figura 6.3: Gráfica de la predicción para el ángulo de dirección para la navegación autónoma sin obstáculos (carril interno)

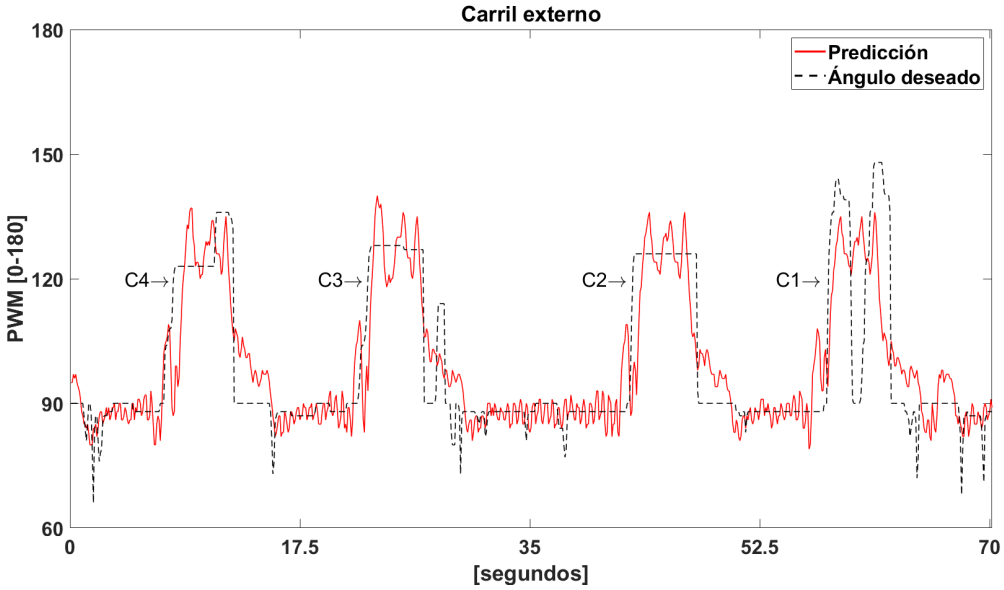


Figura 6.4: Gráfica de la predicción para el ángulo de dirección para la navegación autónoma sin obstáculos (carril externo)

Por otro lado, la Figura 6.4 muestra el resultado de realizar navegación autónoma sin obstáculos en el carril externo para la simulación, y al igual que la gráfica mostrada en la Figura 6.3, esta ilustración muestra valores de color rojo, que representan la predicción que realiza la CNN en todo el momento que dura la vuelta, mientras que la línea punteada negra, al igual que en el caso de la gráfica del carril interno, representa los datos de manejar el prototipo con el mando, es decir, estos datos forman parte de la base de datos para esta tarea, por lo que esta información es parte de lo que convencionalmente se desea que siga la predicción de la CNN, también se observan cuatro zonas que representan las cuatro curvas del circuito (C4, C3, C2, y C1, respectivamente a lo largo del tiempo), cuyos valores de PWM se concentran principalmente entre $90 - 180$, lo que indica que las cuatro vueltas se realizan hacia la izquierda del prototipo, y cuyas etiquetas se encuentran en orden inverso que el mostrado en la Figura 6.3 (C4, C3, C2, y C1), esto debido a que ahora se mueve el prototipo en el carril externo (sentido contrario al carril interno). Por último, se observa en el eje de la abscisa que la duración aproximada de realizar una vuelta completa para esta tarea es de 70 s.

6.2. Resultados con obstáculos estáticos en simulación

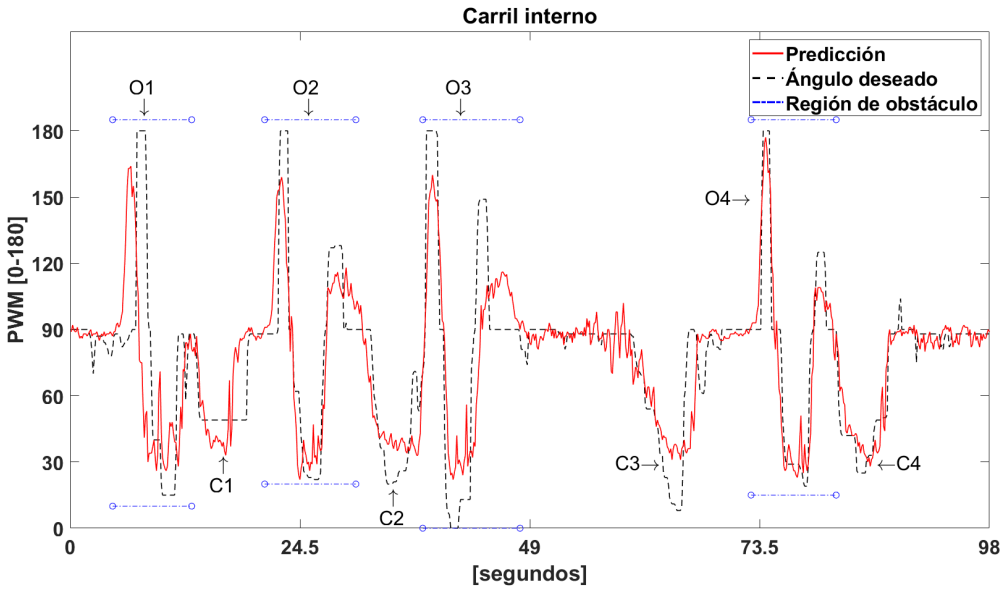


Figura 6.5: Gráfica de la predicción para el ángulo de dirección para la navegación autónoma con obstáculos estáticos(carril interno)

La Figura 6.5 muestra la gráfica del PWM en el tiempo realizado en una vuelta para la navegación autónoma con obstáculos estáticos (simulación) y para el carril interno, con respecto a ello, los valores en color rojo representan la predicción realizada por el modelo CNN en todo momento, por otro lado, la línea punteada negra se refiere al manejo del prototipo realizado con el mando, representando lo que se desea que el prototipo realice. En ese mismo orden de ideas, la ilustración muestra cuatro concentraciones entre $0-90$ del PWM, etiquetados con Ci ($i = 1, 2, 3, 4$), los cuales representan las cuatro curvas, en este caso es más complicado observarlos en comparación a los resultados mostrados en la navegación sin obstáculos, debido a la presencia de otras alteraciones del PWM en el tiempo, los cuales representan a los cuatro obstáculos, y para una mejor visualización, estas alteraciones se encuentran marcadas en cuatro regiones azules (O1, O2, O3 y O4 respectivamente), en consecuencia, cada alteración está dada por tres variaciones, es decir, tres máximos de PWM en el tiempo, los cuales son generados para evadir los obstáculos (el primero con un movimiento del prototipo hacia la izquierda, luego a la derecha para colocarse en el carril contrario y por último, a la derecha, para reincorporarse a su carril). En este caso específico el tiempo que le toma al prototipo realizar el movimiento mientras evade los obstáculos en una vuelta completa al circuito es de 98 s.

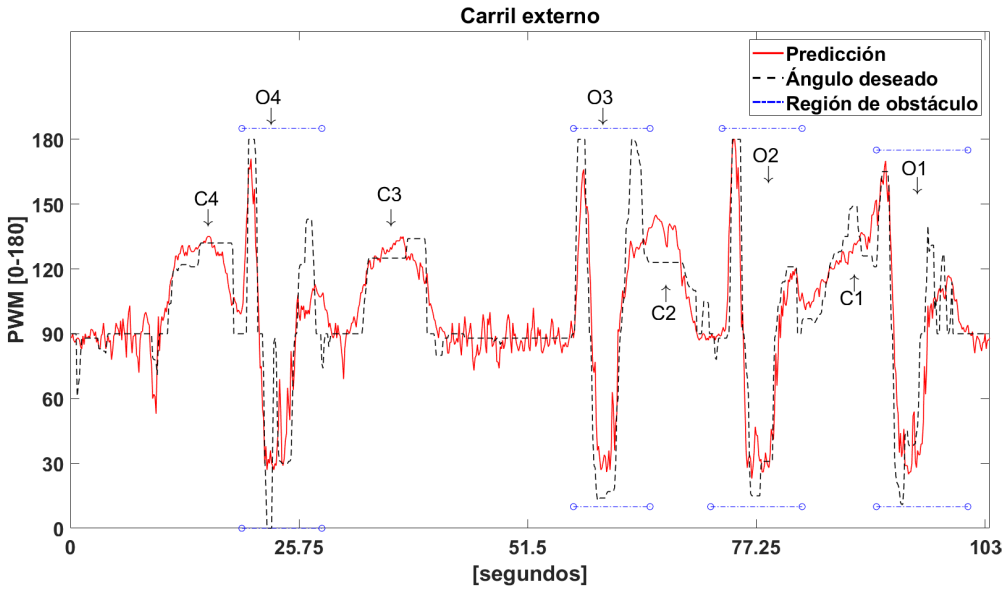


Figura 6.6: Gráfica de la predicción para el ángulo de dirección para la navegación autónoma con obstáculos estáticos (carril externo)

Con respecto a la navegación autónoma con obstáculos estáticos para el carril externo (simulación), la Figura 6.6 muestra la información de los resultados obtenidos. A diferencia de lo mostrado en la gráfica de la Figura 6.5 (caso en el carrl interno), las curvas (en el

orden: C4, C3, C2 y C1) se encuentran en la región de $90 - 180$ con respecto al PWM, esto debido a que el prototipo se mueve en sentido contrario, y por lo tanto, las curvas se realizan hacia la izquierda, por otro lado, también se cuenta con cuatro obstáculos como en el carril interno, mismos que se pueden observar por las regiones azules en la gráfica (O1, O2, O3 y O4 respectivamente), y además, es visible que el efecto que estos causan en cuanto a la predicción del PWM en el tiempo son tres máximos por cada obstáculo, destacados en cada región (color azul), cabe evidenciar que este efecto es igual o similar a lo obtenido en la Figura 6.5, debido a que no importa el carril en el que se encuentre el prototipo, siempre se busca adelantar al obstáculo por la izquierda. El tiempo aproximado que le toma al prototipo realizar una vuelta es de 103 s.

6.3. Resultados con obstáculos móviles en simulación

El efecto de la navegación autónoma con obstáculos móviles (simulación), se muestra en la gráfica de la Figura 6.7, la línea roja corresponde a la predicción generada por la CNN correspondiente a la tarea, mientras que la línea discontinua negra pertenece al manejo del prototipo con el mando; del mismo modo se observan cuatro grandes variaciones relacionadas con las curvas (C1, C2, C3 y C4), cuyos valores de PWM se encuentran concentrados entre 0 y 90, lo que indica que las maniobras para pasar las curvas se realizan girando a la derecha, además, se observa una variación de PWM diferente a las generadas por las curvas en el centro de la gráfica y señalado en una región azul, el cual es resultado de un obstáculo que se encuentra en constante movimiento (OM1), y al igual que sucede en la navegación con obstáculos estáticos (Figuras 6.5 y 6.6), el obstáculo genera las mismas tres variaciones por el PWM en el tiempo, resultado del movimiento del prototipo para evadir un obstáculo: un movimiento del prototipo hacia la izquierda, para comenzar a evadir el obstáculo, después hacia la derecha para colocarse en el carril contrario y paralelo al obstáculo, y finalmente, hacia la derecha, para reincorporarse a su carril. Por último, el tiempo que le toma al prototipo recorrer todo el circuito y que es observado en el eje de la abscisa es de aproximadamente 131 s.

En lo que se refiere a la navegación autónoma con obstáculos móviles para el carril externo (simulación), la Figura 6.8 muestra los resultados de predicción tras navegar autónomamente y con mando en el circuito, este circuito cuenta con un obstáculo móvil a velocidad constante, en esta gráfica también se muestran las cuatro aglomeraciones pertenecientes a las curvas (C4, C3, C2 y C1 respectivamente), al igual que el obstáculo móvil (OM2) que sobrepasa después de la primer curva. El tiempo de recorrido que tarda el prototipo en realizar la vuelta completa es de aproximadamente 144 s.

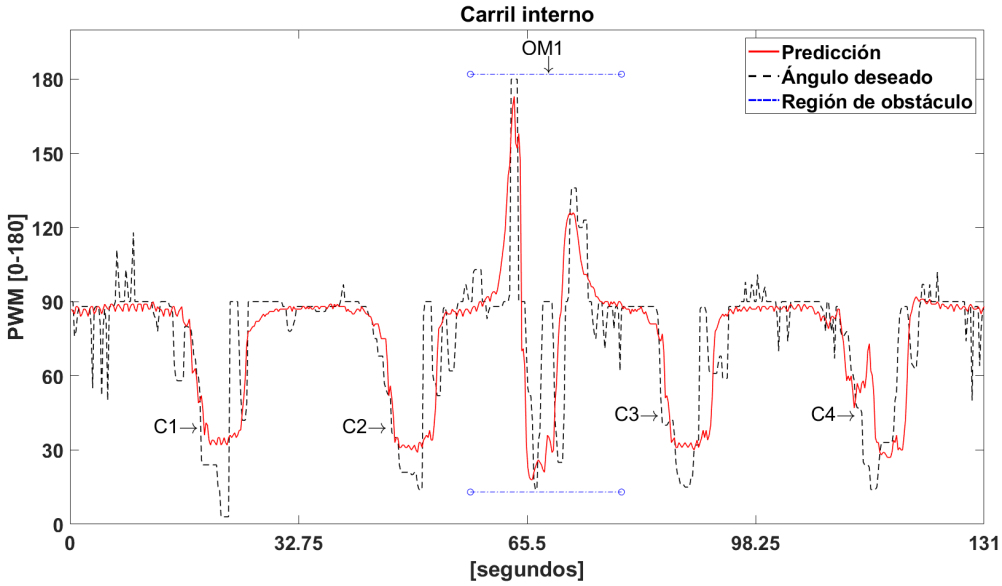


Figura 6.7: Gráfica de la predicción para el ángulo de dirección para la navegación autónoma con obstáculos móviles (carril interno)

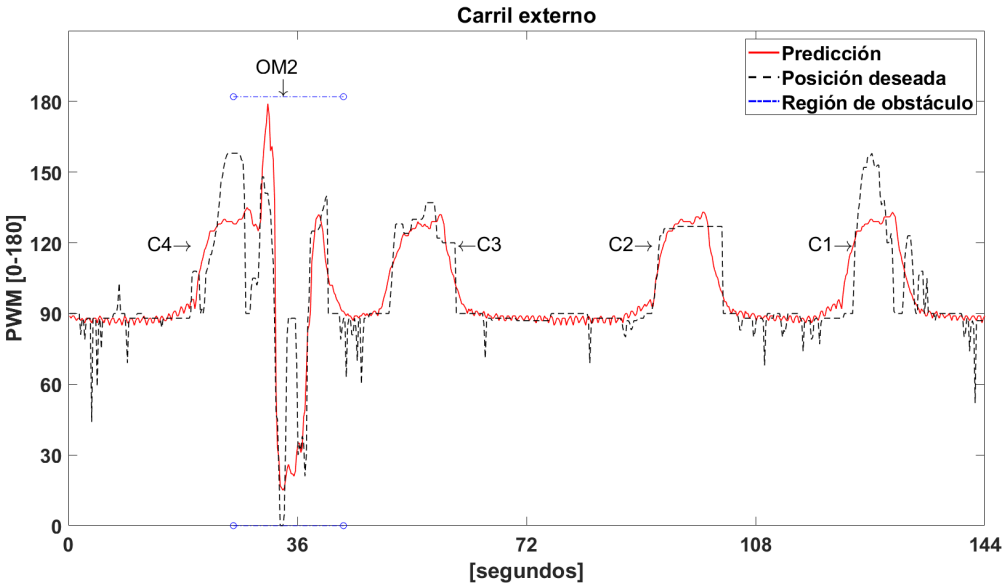


Figura 6.8: Gráfica de la predicción para el ángulo de dirección para la navegación autónoma con obstáculos móviles (carril externo)

6.4. Resultados sin obstáculos en implementación real

El resultado de realizar navegación autónoma sin obstáculos en la implementación real y en el carril interno se puede observar en la gráfica de la Figura 6.9, al igual que en los casos anteriores de simulación, el análisis es de una sola vuelta, por tal razón, la línea de color rojo representa la predicción de la CNN en todo el momento del recorrido, en cambio, la línea negra punteada corresponde al manejo con el mando para la misma vuelta. Las cuatro grandes variaciones de PWM en el tiempo señaladas como C1, C2, C3 y C4 respectivamente, corresponden a las curvas que tiene que realizar el prototipo y que además forman parte del circuito, del mismo modo, estos datos relevantes de PWM están inclinados a valores en el rango de $1500 - 2000$ (en términos de PWM), indicativo de que todas las vueltas se realizan hacia la derecha. En consecuencia, el tiempo estimado que le toma al prototipo realizar una vuelta completa, y sin salirse de su carril es de 40 s.

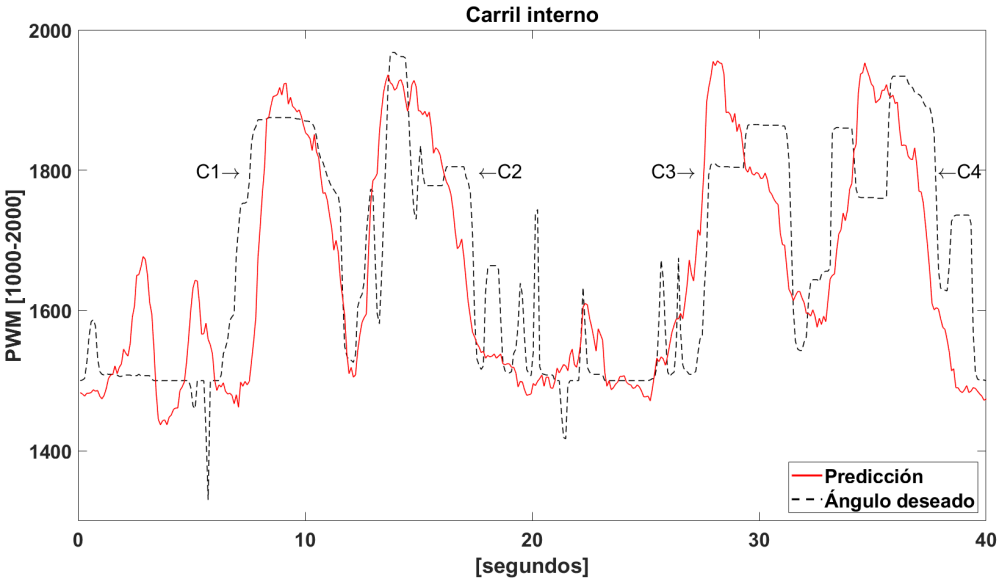


Figura 6.9: Gráfica de la predicción para el ángulo de dirección para la navegación autónoma sin obstáculos (carril interno)

Por otra parte, la Figura 6.10 muestra la gráfica del comportamiento que tiene el prototipo en una vuelta, sin embargo, y a diferencia de la gráfica de la Figura 6.9, comienza en otra zona y además, en el carril externo; la línea roja de igual manera, corresponde a la predicción de la CNN la cual es encargada de predecir el ángulo de dirección en todo momento, mientras que la línea negra punteada corresponde al manejo del mando de videojuegos realizando el movimiento de manera controlada sin salirse del carril, en este caso, los valores tienen mayor

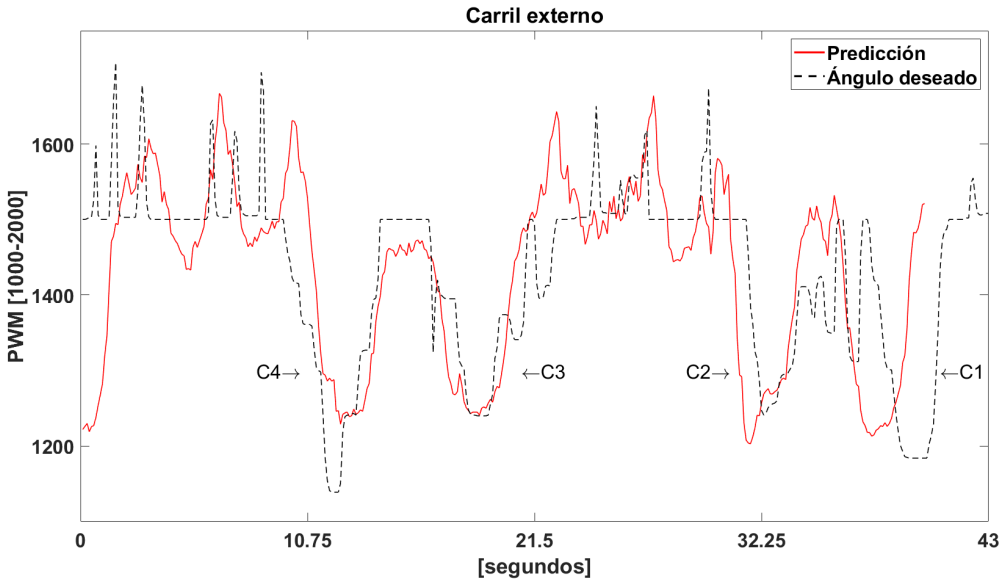


Figura 6.10: Gráfica de la predicción para el ángulo de dirección para la navegación autónoma sin obstáculos (carril externo)

inclinación en un rango entre $1000-1500$ (en términos de PWM), lo que significa, a diferencia del carril interno, que el prototipo gira a la izquierda en cada curva, señalados como C4, C3, C2 y C1 (respectivamente). El tiempo estimado de la realización de una vuelta por parte del prototipo sin salirse de su carril es de 43 s.

6.5. Resultados con obstáculos estáticos en implementación física

La gráfica de la Figura 6.11 muestra el comportamiento para una vuelta realizando la navegación autónoma con obstáculos estáticos y en el carril interno, la línea roja corresponde a la predicción realizada por la CNN propuesta para evadir obstáculos, mientras que la línea punteada negra representa el manejo con el mando en una vuelta realizada al circuito, esta ilustración, además de mostrar el efecto del PWM ante las curvas (C1, C2, C3 y C4), cuya orientación del PWM se encuentra entre $1500-2000$ debido a que las curvas las realiza hacia la derecha, también muestra las zonas en dos regiones azules donde ocurre la evasión de obstáculos (O1 y O2), en este caso la evasión implica tres grandes variaciones de PWM en el tiempo que realiza la predicción, estas variaciones corresponden al movimiento del prototipo hacia la izquierda, para comenzar a evadir el obstáculo, luego hacia la derecha

para colocarse en el carril contrario y paralelo al obstáculo, para finalmente, reincorporarse a su carril moviéndose a la derecha, este efecto es notorio en ambas regiones de ambos obstáculos, por último, el tiempo aproximado que le toma al prototipo realizar una vuelta sin salirse del circuito y evadiendo los dos obstáculos es de 47 s.

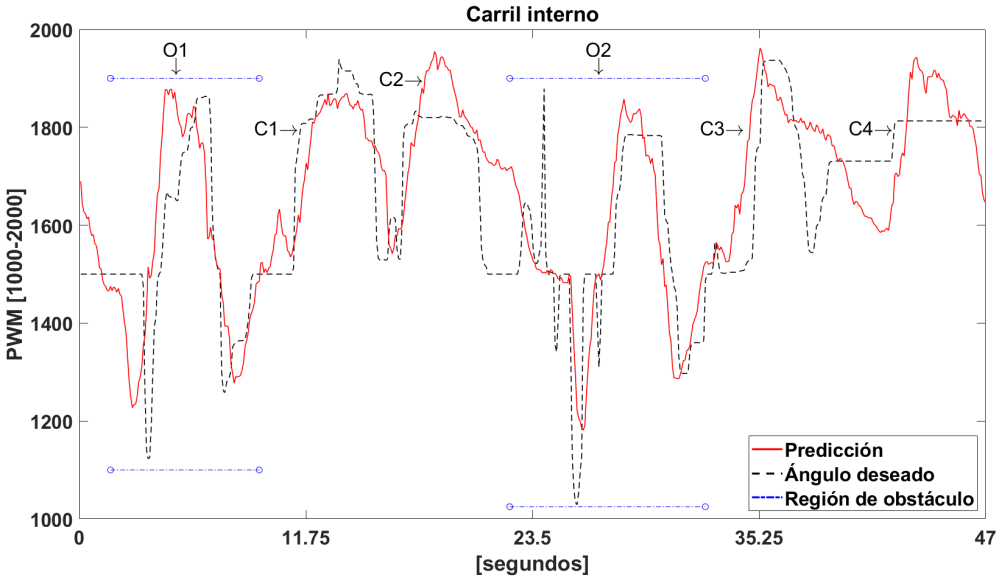


Figura 6.11: Gráfica de la predicción para el ángulo de dirección para la navegación autónoma con obstáculos estáticos(carril interno)

No obstante, la navegación autónoma con obstáculos estáticos para el carril externo se puede apreciar en la gráfica de la Figura 6.12, en ella se observa el efecto del PWM debido a las cuatro curvas (C4, C3, C2 Y C1), el efecto del PWM se encuentra entre $1000 - 1500$ debido a que el prototipo ahora gira en sentido anti-horario para cada curva, mientras que los obstáculos se encuentran marcados por una región azul (O1 y O2), estos obstáculos tienen la misma convención que el mostrado en la gráfica de la Figura 6.11, debido a que el revase de los obstáculos se realiza hacia el carril izquierdo, y del mismo modo, estas variaciones corresponden al movimiento del prototipo hacia la izquierda, para comenzar a evadir el obstáculo, luego hacia la derecha para colocarse en el carril contrario y paralelo al obstáculo, para finalmente, reincorporarse a su carril moviéndose a la derecha. Por último, el tiempo aproximado de realizar la tarea de navegación evadiendo obstáculos por parte del prototipo para una vuelta es de 50 s.

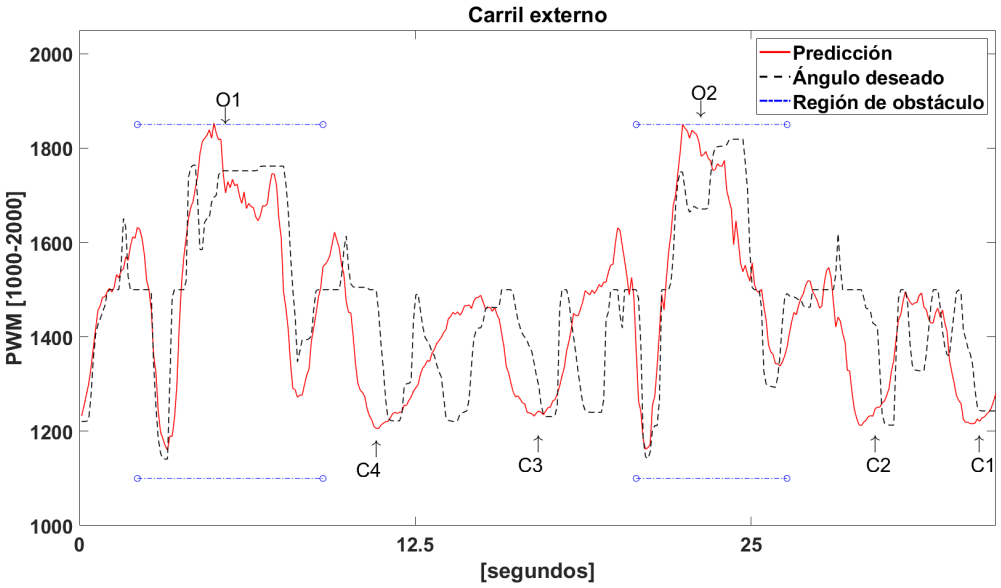


Figura 6.12: Gráfica de la predicción para el ángulo de dirección para la navegación autónoma con obstáculos estáticos (carril interno)

6.6. Error cuadrático medio

Concentrar la información de los resultados de las gráficas de navegación autónoma en una tabla y para cada una de las tareas es determinante para interpretar de otra manera los datos. En este contexto, los datos previamente mostrados (Figuras 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10, 6.11 y 6.12) son información cualitativa de los resultados, sin embargo, cuantificar estos datos proporcionan una comprensión profunda de los mismos datos, en este caso, se hace uso del error cuadrático medio para concentrar la información de lo que predice cada una de las CNN en comparación de lo obtenido al manejar el prototipo con un mando, esta información se encuentra dada en valores del PWM, por un lado, en la simulación los valores van de 0 (giro a la derecha) a 180 (giro a la izquierda), mientras que por el otro, en la implementación real estos valores van desde 1000 (giro a la izquierda) hasta 2000 (giro a la derecha).

Por lo tanto, la Tabla 6.1 muestra el error para cada una de las gráficas de los casos de navegación en simulación, en ella se pueden observar los errores de los tres retos para una vuelta, esta información es reunida a partir gráficas de las Figuras 6.3 y 6.4, correspondientes al caso en el que no hay obstáculos (reto uno), por otro lado, las gráficas de las Figuras 6.5 y 6.6 corresponden al caso en el que se tienen obstáculos estáticos (reto dos), y por último,

para el tercer caso (en el que el obstáculo es móvil, reto tres), las gráficas de las Figuras 6.7 y 6.8 son las empleadas en este caso. Es de suma importancia señalar que, en los tres desafíos, el error del PWM en el carril externo es inferior al obtenido en el carril interno.

Simulación	Error (PWM)
Reto 1: carril interno	334.9127
Reto 1: carril externo	108.5191
Reto 2: carril interno	516.6958
Reto 2: carril externo	277.0400
Reto 3: carril interno	367.6446
Reto 3: carril externo	182.4835

Tabla 6.1: Concentración de resultados para simulación

Por otro lado, en la Tabla 6.2 se muestran los resultados en cuanto al error cuadrático medio para el caso de la implementación real. En ese mismo orden, se pueden observar el error para los dos retos, dicha información se muestra a partir de lo obtenido en las gráficas de las Figuras 6.9 y 6.10 para el caso en el que no hay obstáculos (reto uno) y de las gráficas de las Figuras 6.11 y 6.12 para el caso en el que los obstáculos son estáticos (reto dos). Y del mismo modo que sucede en la simulación, por lo menos en el caso sin obstáculos, el error en el carril externo es mayor que en el caso del carril interno, sin embargo esto es completamente diferente en el caso en el que se tienen obstáculos estáticos, ya que el error en el carril externo es mayor que en el caso del carril interno.

Simulación	Error (PWM)
Reto 1: carril interno	9263
Reto 1: carril externo	8450.2010
Reto 2: carril interno	11’176.1675
Reto 2: carril externo	11’922.974

Tabla 6.2: Concentración de resultados para implementación real

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se detallan las conclusiones de los puntos más importantes observados a lo largo de la realización de este trabajo con respecto a los resultados obtenidos y empleando los objetivos específicos planteados al inicio de este proyecto, por otro lado, se realizan algunas propuestas de actividades que puedan seguir enriqueciendo este trabajo, y en consecuencia, extender y difundir los alcances a futuro que se podrían desempeñar para continuar con la investigación presentada.

7.1. Conclusiones

Los resultados se dividen en dos partes, en primer lugar los resultados de entrenamiento, y tomando en cuenta que se crearon cinco bases de datos, cada base de datos fue entrenada individualmente, puesto que las tareas se tomaron como independientes, en ese contexto, se buscó obtener el menor error posible, partiendo de los resultados de la primer tarea en simulación, eso dio pauta para conocer el error al que se debía aproximar los resultados, y evidentemente, que funcionara el modelo en el prototipo realizando la tarea para la que fue creada. Por otro lado, los resultados de navegación autónoma muestran gráficas del movimiento en el circuito propuesto para cada tarea, estos resultados no solo demuestran que el prototipo realiza las tareas propuestas satisfactoriamente, sino que también son comparadas con lo que se obtiene manejando al prototipo utilizando un mando, por último, esta infor-

mación es contenida en tablas, comparando el error (PWM) para cada caso y en cada carril del circuito, a modo de interpretar de otra manera los datos respecto a los resultados.

Así mismo, en la investigación previa se notó la poca información sobre la generación de algoritmos para la navegación autónoma, por otro lado, esta información tenía mayor peso en configuraciones de prototipos tales como robots diferenciales, bajo simulación y en un circuito de un solo carril, además de contar con colores en el carril para una mejor percepción en el camino. En consecuencia, se decidió emplear el AutoNOMOS V2 para estas actividades ya que era un robot con configuración Ackerman (el mismo que un auto real), y que igualmente es un prototipo a escala (1:10).

Se decidió emplear redes neuronales convolucionales para la ejecución de este proyecto, en ese sentido, la mayor herramienta utilizada fue la cámara situada en la parte superior del prototipo, tanto en simulación cómo en el prototipo real.

Uno de los principales problemas para el aprendizaje profundo es la selección de características, así pues, se realizó un trabajo previo donde se propusieron dos áreas de interés para la estimación del ángulo de dirección empleando dos CNN, teniendo como finalidad realizar la navegación autónoma en un entorno controlado bajo simulación, y en consecuencia, observar que área de interés era necesaria emplear para las actividades siguientes en este proyecto. Por lo tanto, los datos recogidos fueron suficientes para confirmar que los modelos propuestos realizaran la actividad de mantenerse en su carril sin salirse del mismo. Sin embargo, se observaron algunas diferencias entre las dos áreas de interés, puesto que en la primera se generaba una vista de águila, lo que ocasionaba que la imagen fuese limpia del ruido ocasionado por información innecesaria, mientras que en la otra se utilizó la imagen original proveniente de la cámara, la primer CNN (vista de águila) aprendía más rápido y el error era menor, a diferencia de la segunda CNN (imagen original proveniente de la cámara), el cual no estaba mal pero le costaba un par de épocas más llegar a su menor error el cual no era mejor que la vista de águila pero tampoco estaba alejado, si bien la vista de águila era mejor, la falta de información podría ser relevante para problemas en particular. En otro orden de cosas, aunque el proceso fue más tardado para la segunda, la información adicional es de utilidad para otras tareas (como la evasión de obstáculos), por lo tanto, se decidió utilizar la imagen proveniente de la cámara con algunos recortes en la parte inferior y superior para eliminar información innecesaria, cabe destacar que los resultados obtenidos de este trabajo previo fue publicado en el congreso COMROB 2023 y la evidencia se puede encontrar en A.

El prototipo físico contaba con algunas limitaciones para implementar las CNN, en ese sentido, se optó primeramente por cambiar la computadora encargada de realizar los pro-

cesos en el prototipo, la computadora utilizada para estas tareas es una Jetson Nano de 4 GB, capaz de realizar procesos más rápidos que otras computadoras ya que cuenta con una tarjeta gráfica, además, este cambio agregó dos actualizaciones más, por un lado se tenía la actualización del software completo, cambiando de Ubuntu 14.04 a 18.04, lo cual implicaba mayor estabilidad y una mayor compatibilidad con librerías y paquetes más actuales, mientras que por el otro, este cambio de computadora ayudaba a emplear una red 5G para el envío de datos (imágenes) a otra computadora, ya que anteriormente solamente se podía emplear la red2.4 G, esto contribuyó en gran medida a realizar la navegación autónoma sin la perdida de datos ocasionada al enviar datos por Wi-Fi. Estos cambios no solo ayudaron a realizar las tareas propuestas en este proyecto, sino que dan pie a desarrollar algoritmos más robustos, por ejemplo, el intercambio de información más pesada y además bilateral entre computadora-robot o incluso robot-robot se puede realizar sin tener una pérdida de información.

El procesamiento de los datos lo realizaba una computadora externa con fines de investigación, ya que era necesario guardar información pesada, y antes de emplear la información de la cámara y el LiDAR, estos pasan por un pre procesamiento, en el que se ajusta la imagen de la cámara y recibe algunos recortes, por otro lado, el sensor LiDAR también sufre algunos cambios, dado que esta información es recibida en vectores, para los fines de este proyecto era necesario convertirlo a imágenes, una de las ventajas de emplear ROS es la fácil compatibilidad que tiene con librerías de Python, en ese sentido, se convertía cada lectura del LiDAR en imágenes.

Para cada tarea se creó una base de datos independiente una de otra, esta decisión fue con la intensión de analizar independientemente cada tarea, debido a que es el primer contacto con el prototipo y el uso del aprendizaje supervisado, como ya se mencionó, aún no hay suficiente información de este prototipo con el uso de redes neuronales, por lo tanto, buscar realizar el mayor aporte al área es necesario, por otro lado, la Federación Mexicana de Robótica (FMR), en su concurso anual que realiza sobre el AutoNOMOS, específicamente en el año 2022 dio la oportunidad de realizar cambios físicos, electrónicos e internos a este robot (anteriormente estaba prohibido), por lo que información nueva en cuanto a cambios realizados es muy escasa, en ese sentido, la realización de las tareas, independiente una de otra fue necesario para futuras implementaciones.

El entrenamiento de cada CNN obtuvo buenos resultados tanto en simulación como en la implementación real, la percepción la realizó sin pérdida de información y para cada una de las actividades propuestas, el prototipo en ningún caso sale de su carril, la evasión de obstáculos se realiza de manera satisfactoria, y la evasión de obstáculos móviles de igual manera fue cumplida. No obstante, algo interesante es que en una primer instancia al inicio

el entrenamiento se realizó con imágenes que correspondían a un solo carril, sin embargo, la falta de información del carril restante hizo que el modelo perdiera el control en los tramos largos del circuito y, si bien la red sí aprendía, ésta se confundía, por lo tanto, después de agregar la información de ambos carriles este error desapareció.

Tomando en cuenta a la hipótesis propuesta la cual menciona lo siguiente "Utilizando la información obtenida por los sensores del vehículo AutoNOMOS mini V2, es posible desarrollar un sistema de percepción con CNN para la navegación autónoma considerando obstáculos estáticos y en movimiento sobre un ambiente controlado, así como cumplir con los retos establecidos por el TMR", se puede decir que la hipótesis se cumple, puesto que las tres tareas propuestas son cumplidas en la simulación, por otra parte, en la implementación real el tercer objetivo no fue completado debido a algunas fallas en el sistema del prototipo, además, una construcción de un robot complementario para que se mueva en la pista como obstáculo móvil no fue planteado en los objetivos. Sin embargo, con lo obtenido se puede corroborar la suposición planteada. Es decir, los resultados respaldan la hipótesis.

7.2. Trabajo futuro

En el contexto de esta investigación, se han logrado avances significativos en el desarrollo de un robot autónomo equipado con redes neuronales convolucionales (CNN) para la navegación. Sin embargo, para continuar mejorando la capacidad y versatilidad de este prototipo, es necesario considerar trabajos futuros.

Un primer desafío se encuentra en la navegación autónoma en entornos con obstáculos dinámicos para la implementación real. Esto implica agregar robots con algún sistema de control de posición como fue el caso en la simulación, estos robots deberán tener la facilidad de cambiar velocidades y de que no se salgan del carril en el que navegan.

Otro aspecto relevante es mantener la mejora del hardware del robot. La posibilidad de cambiar el motor y desarrollar un control puede incrementar la eficiencia y la capacidad de movimiento en diversas condiciones. Esto no solo afecta la movilidad del robot, sino que también puede influir en la velocidad y estabilidad, aunado a que se podrá agregar otras técnicas, como conocer la odometría del prototipo.

En el contexto de la aplicación del robot en entorno de estacionamiento, es fundamental desarrollar algoritmos que permitan al robot estacionarse de manera autónoma. Esto implica la identificación de espacios de estacionamiento disponibles y la ejecución de maniobras para ingresar en ellos.

La integración de técnicas de análisis semántico de imágenes con redes neuronales profundas es otro camino de investigación, esto incluye el reconocimiento de objetos, la interpretación del contexto y la capacidad de comprender señales visuales, por lo que puede aumentar la comprensión del entorno y la toma de decisiones del robot.

La validación del sistema en entornos menos controlados y más realistas es un tópico hacia la implementación del robot en situaciones del mundo real, esto implica la simulación y la implementación física en entornos que cuenten con desafíos como variabilidad de terreno, cambios de iluminación y condiciones no tan predecibles.

La investigación en comunicación e interacción con otros robots y dispositivos inteligentes en el entorno es una línea de investigación que puede llevar a aplicaciones colaborativas y en consecuencia, más complejas. Por otro lado, la implementación de técnicas de aprendizaje por refuerzo puede mejorar la toma de decisiones del robot en situaciones dinámicas y desconocidas, lo que es fundamental para aumentar la navegación autónoma.

Estos trabajos futuros representan una visión de las áreas de investigación las cuales van acorde a lo desarrollado en este trabajo, y reflejan un compromiso con el avance de la robótica autónoma en el futuro.

Referencias

Allen, B. (2012). Think bayes. En *Bayesian stadistics in python* (Vol. 1, pp. 1–10).

Andrés, R. (2020). *El coche autónomo de waymo ya ve más que un conductor humano.* https://computerhoy.com/noticias/motor/coche-autonomo-waymo-ya-ve-conductor-humano-594415. (Accesado: 26.01.2022)

Arevalo, V., González, J., y Ambrosio, G. (2005a, 01). La librería de visión artificial opencv, aplicación a la docencia e investigación. En (Vol. s.n, pp. 1–3).

Arevalo, V., González, J., y Ambrosio, G. (2005b, 01). Opencv, la librería open source de visión artificial (in spanish). En (Vol. 10, pp. 141–147).

AutoNOMOS. (2017). https://github.com/AutoModelCar/AutoModelCarWiki/wiki/Hardware. (Accesado: 07.11.2021)

AWS. (2021, 03). *Aws deep racer.* https://pages.awscloud.com/LATAM-field-OE-DeepRacer-Mx-F1-2021-reg-event.html. (Accesado: 28.05.2022)

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... others (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Bravo, C. (2018). Navegación autónoma de un robot tipo automóvil en pista de carreras con obstáculos. En (Vol. s.n., p. s.n.).

Broggi, A., Cerri, P., Debattisti, S., Chiara, M., Panciroli, M., y Prioletti, A. (2015). Proud–public road urban driverless-car test. En *Transactions on intelligent transportation systems* (Vol. 16, pp. 3508–3519).

Brown, B. (2017). The social life of autonomous cars. En *Computer* (Vol. s.n., pp. 92–96).

Calvo, J., Guzmán, M., y Ramos, D. (2018). Machine learning, una pieza clave en la transformación de los modelos de negocio. En (Vol. s.n., pp. 20–22).

Cognex. (2018). Introducción a la visión artificial: Una guía para la automatización de procesos y mejorar la calidad. En (Vol. s.n., pp. 3–5).

Collado, J. (2009). Detección y modelado de carriles de vías interurbanas mediante análisis de imágenes para un sistema de ayuda a la conducción. En (pp. 1–3).

de Robótica, F. M. (2021). *Tmr.* www.femexrobotica.org. (Accesado: 20.10.2021)

Diaz, E. (2020). Generación del mapa métrico de una pista de competencias para la navegación del robot móvil autonomos mini v2. En (Vol. s.n., p. s.n.).

Gershenson, C. (2012). Artificial neural networks for beginners. En (Vol. s.n., pp. 1–7).

Gonzalez, R., y Woods, R. (2018, 01). Digital image processing. En (Vol. 4, pp. 964–987).

Gonzalez, R., y Woods, R. (2019). Digital image processing. En (Vol. 2, pp. 49–51).

González, A., Martínez, F., Pernía, A., Alba, F., Castejón, M., Ordieres, J., y Vergara, E. (2006). Técnicas y algoritmos básicos de visión artificial. En (Vol. 24, pp. 11–17).

Grisleri, P., y Fedriga, I. (sf). The braive autonomous ground vehicle platform. En (Vol. s.n, pp. 1–6).

Hagan, M., Demuth, H., Hudson, M., y Jesus, O. (s.f.). Neural network design. En (Vol. 2, pp. 18–19).

Hernández, R., Fernández, C., y Baptista, M. (2010). Metodología de la investigación. En (Vol. 5, pp. 50–75).

IMCO. (2019). Índice de movilidad urbana. En (pp. 11–13).

Ladero, R. (2019). Detección de señales y líneas de carril para conducción autónoma con vehículo a escala. En (pp. 9–12).

Moya, R. (2016). *¿qué es el clustering?* https://jarroba.com/que-es-el-clustering/. (Accesado: 09.01.2022)

Nolte, M., Form, T., Ernst, S., Graubohm, R., y Maurer, M. (2018). The carolo-cup student competition: Involving students with automated driving. En *Proceedings of the 12th european workshop on microelectronics education* (Vol. 1, pp. 95–99).

Ortego, D. (2017). *Qué es ros (robot operating system).* https://openwebinars.net/blog/que-es-ros/. (Accesado: 08.12.2021)

Popescu, M., Balas, V., Popescu, L., y Mastorakis, N. (2009). *Perceptrón multicapa y redes neuronales.* https://www.researchgate.net/publication/228340819_Multilayer_perceptron_and_neural_networks. (Accesado: 10.05.2022)

Riva, J. (2021). *Ros.* http://wiki.ros.org/es. (Accesado: 08.12.2021)

Robert, J. (2020). *Hands-on introduction to robot operating system (ros).* https://trojrobert.github.io/hands-on-introdution-to-robot-operating-system(ros)/. (Accesado: 08.12.2021)

Rojas, R. (2021). *Model cars for autonomous driving.* https://www.femexrobotica.org/tmr2021/portfolio-item/autos-autonomos/. (Accesado: 20.10.2021)

Roman, V. (2019). *Introducción al machine learning: Una guía desde cero.* https://medium.com/datos-y-ciencia/introduccion-al-machine-learning-una-guia-desde-cero-b696a2ead359. (Accesado: 09.01.2022)

Sandoval, L. (2018). Algoritmos de aprendizaje automático para análisis y predicción de datos. En (Vol. 11, pp. 36–40).

Silva, D. (2021). *Diferencia entre machine learning y deep learning.* https://www.zendesk.com.mx/blog/machine-learning-deep-learning-diferencias/#:~:text=La%20principal%20diferencia%20entre%20el,incluyen%20esta%20extraccion%20de%20caracteristicas. (Accesado: 14.05.2022)

Sotaquirá, M. (2019, 03). *La convolución en las redes convolucionales.* https://www.codificandobits.com/blog/convolucion-redes-convolucionales/. (Accesado: 17.05.2022)

Stöckle, C., Herrmann, S., Dirndorfer, T., y Utschick, W. (2020). Automated vehicular safety systems, robust function and sensor design. En *Signal processing magazine* (Vol. 37, pp. 24–33).

Tesla. (2021). *El futuro de la conducción.* https://www.tesla.com/es_MX/autopilot. (Accesado: 06.11.2021)

TMR. (2018). *Automodelcar.* https://www.femexrobotica.org/tmr2018/portfolio-item/autos-autonomos/. (Accesado: 02.03.2022)

Valentín, M. (2018). Visión artificial, una herramienta para los sistemas inteligentes. En (Vol. s.n., pp. 3–6).

Waymo. (2019). *Waymo driver.* https://waymo.com/intl/es/waymo-driver/. (Accesado: 07.11.2021)

Apéndice A

Artículo publicado

Durante la investigación para el desarrollo de esta tesis fue necesario elegir un área de interés del cual partir para crear las bases de datos. Como resultado, se publicó un artículo en el **XXV Congreso Mexicano de Robótica COMRob 2023**. Este artículo tiene el titulo de: *Implementación de dos áreas de interés para la predicción del ángulo de dirección de un auto a escala en tiempo real*. A continuación el diploma y la primer página del artículo:

Implementación de dos áreas de interés para la predicción del ángulo de dirección de un auto a escala en tiempo real

Filiberto-E. Martinez-Hernandez
División de Estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapan de León, Oaxaca, México
https://orcid.org/0009-0004-0064-3871

José Anibal Arias-Aguilar
División de Estudios de Posgrado
Universidad Tecnológica de la Mixteca
Huajuapan de León, Oaxaca, México
https://orcid.org/0000-0002-8838-877X

Edgar Macias-Garcia
Intelligent System Research
Intel Labs
Zapopan, Jalisco, México
https://orcid.org/0000-0003-2571-9460

***Resumen*—La predicción del ángulo de dirección en vehículos inteligentes es un tópico ampliamente estudiado en la literatura, sin embargo, en la mayoría de las aplicaciones se requiere de la intervención de un usuario externo para operar el vehículo, proveyendo así un grado parcial de autonomía. En este trabajo, se proponen algoritmos basados en redes neuronales artificiales para determinar el ángulo de giro de un vehículo en movimiento, con aplicaciones en tiempo real para modelos a escala sobre una pista de dos carriles. Empleando una cámara frontal centrada en el vehículo se determina un área de interés localizada a nivel de piso, la cual es rectificada usando un cambio de perspectiva. Posteriormente, se usa la tupla de imágenes para construir una base de datos etiquetada con el ángulo deseado de giro, que es usada para entrenar una red neuronal convolucional. Una vez entrenada, la red neuronal es implementada en tiempo real para guiar a un vehículo a escala sobre una pista libre de obstáculos, permitiendo su navegación autónoma dentro del carril designado.**

***Index Terms*—Aprendizaje supervisado, área de interés, CNN, sistema de dirección, vehículos inteligentes.**

I. Introducción

En el contexto de la conducción autónoma, existen aún áreas de oportunidad para desarrollar algoritmos que brinden a los vehículos la capacidad de ser completamente autónomos. En este sentido, de acuerdo con [1], los Sistemas Avanzados de Asistencia a la Conducción (ADAS, por su sigla en inglés) permiten al vehículo desarrollar diferentes tipos de tareas individuales tales como la detección de peatones, la activación del freno en casos de emergencia, la detección de ángulo de dirección, o el control de crucero adaptativo. De manera similar [2], el brindar autonomía a los vehículos permite mejorar la mecánica de la conducción en diversas situaciones; sin embargo, el problema se encuentra en que dicha conducción siempre busca estadísticamente evitar una colisión, lo cual no siempre es la mejor decisión dadas las circunstancias del entorno. Por otro lado, la investigación de [3] establece que la navegación autónoma requiere capturar la mayor cantidad de información posible del entorno que rodea al vehículo, a fin de tomar decisiones basadas en aspectos de seguridad, eficiencia y condiciones del entorno.

En el contexto académico y de investigación también se han desarrollado contribuciones importantes para el área, tal es el caso de [4], donde los autores introdujeron el proyecto BRAiVE desarrollado en la Universidad de Parma, a partir del cual se creó un sistema de navegación que permitió a un vehículo conducir de forma autónoma por 13 kilómetros. El vehículo contaba con dos escáneres láser montados en la parte delantera del vehículo, mientras que el sistema de posicionamiento empleaba un dispositivo de navegación cinética satelital; sin embargo, fue necesario que un conductor apoyara en algunas tareas que el sistema no podía realizar por sí solo. Por otra parte, de acuerdo con [5], NVIDIA logró desarrollar una red neuronal profunda que permite guiar y predecir el ángulo de dirección para la navegación autónoma. Esta topología es capaz de reconocer objetos que se encuentran en el camino, vehículos, postes e incluso objetos a los costados.

Posteriormente, la Universidad Libre de Berlín, Alemania, introdujo AutoNOMOS [6] como plataforma física de un automóvil a escala 1:10, el cual incorporaba un modelo para simulación con propiedades similares, tanto físicas como de adquisición de datos y diseño de un auto real. Contaba con una computadora de 64 GB para el sistema y almacenamiento, un servomotor para lograr el ángulo de dirección, un motor sin escobillas para la tracción, un sensor IMU para la orientación, una cámara Intel 3D y un sensor lidar.

En México también se han desarrollado proyectos sobre esta plataforma, como el trabajo desarrollado en [7] donde se propone una metodología para implementar la navegación autónoma y evasión de obstáculos, la cual consiste en operaciones morfológicas para búsqueda de máximos locales que detecten puntos sobre la línea de carril cuantificados en un histograma desarrollado a partir de la imagen, siendo eficaz para diferentes condiciones de iluminación. Así mismo, la Universidad Tecnológica de la Mixteca [3], desarrolló un algoritmo para la reconstrucción de mapas métricos generados a partir de la cámara Intel Real Sense colocada en la parte frontal superior del AutoNOMOS Mini V2.

Apéndice B

Programa computacional para inicialización del robot prototipo

```
#!/bin/bash

password="robotica"
source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash

echo $password | sudo -S chmod 777 /devel/bus/ussb/001/*
echo $password | sudo -S chmod 777 /dev/ttyMotor
source /robotica/.bashrc
roscore &
sleep 17
roslaunch manual_control manual_odroid.launch &
roslaunch astra_camera astra_pro.launch &
rosrun ROI2 Deteccion.py &
export ROS_MASTER_URI=hhtp://192.168.43.179:11311
export ROS_IP=192.168.43.179
```

Programa B.1: Auto-inicio

Apéndice C

Programas computacionales para adquisición de datos

```
#!/usr/bin/env python

from tokenize import Double
import rospy
from ackermann_msgs.msg import AckermannDrive
from sensor_msgs.msg import Joy
from std_msgs.msg import Int16
from std_msgs.msg import Float32
import sys

class AckermannDriveJoyop:
    def __init__(self, args):
        if len(args)==1 or len(args)==2:
            self.max_speed = float(args[0])
            self.max_steering_angle = float(args[len(args)-1])
            vel_topic= 'set_velocity'
            angle_topic= 'set_angle'
        elif len(args) == 3:
            self.max_speed = float(args[0])
            self.max_steering_angle = float(args[1])
            vel_topic= 'set_velocity'
            angle_topic= 'set_angle'
        else:
            self.max_speed = 500
            self.max_steering_angle =180
            self.min_steering_angle =0.41853095368099036
            vel_topic= 'set_velocity'
```

```
28              angle_topic= 'set_angle'
29              angulo_pub='Angulo'
30          self.speed = 0
31          self.steering_angle = 0
32          self.joy_sub = rospy.Subscriber('/joy', Joy, self.joy_callback)
33          self.vel_pub=rospy.Publisher('/AutoNOMOS_mini/manual_control/speed
    ',Int16,queue_size=10)
34          self.angle_pub=rospy.Publisher('/AutoNOMOS_mini/manual_control/
    steering',Int16,queue_size=10)
35          self.angulo_pub=rospy.Publisher('/angulo',Int16,queue_size=10)
36          rospy.Timer(rospy.Duration(1.0/30.0), self.pub_callback, oneshot=
    False)
37          rospy.loginfo('ackermann_drive_joyop_node initialized')
38
39      def joy_callback(self, joy_msg):
40          gamma_positivo=90
41          gamma_negativo=89
42          self.speed = int(-1*(joy_msg.axes[1] * (self.max_speed)))
43          if(joy_msg.axes[3]>0):
44              self.steering_angle = int(90+(joy_msg.axes[3] * gamma_positivo
    ))
45          if(joy_msg.axes[3]<0):
46              self.steering_angle = int(89+(joy_msg.axes[3] * gamma_negativo
    ))
47          if(joy_msg.axes[3]==0):
48              self.steering_angle = int(90)
49
50      def pub_callback(self, event):
51          self.vel_pub.publish(self.speed)
52          self.angle_pub.publish(self.steering_angle)
53          self.angulo_pub.publish(self.steering_angle)
54          self.print_state()
55      def print_state(self):
56          sys.stderr.write('\x1b[2J\x1b[H')
57          rospy.loginfo('\x1b[1M\r''\033[31;1m''Salida')
58          rospy.loginfo('\x1b[1M\r'
59                        '\033[34;1mSpeed: \033[32;1m%0.2f m/s, '
60                        '\033[34;1mSteering Angle: \033[32;1m%0.2f rad\033[0
    m',
61                        self.speed, self.steering_angle)
62      def finalize(self):
63          rospy.loginfo('Halting motors, aligning wheels and exiting...')
64          sys.exit()
65
66  if __name__ == '__main__':
67      rospy.init_node('ackermann_drive_joyop_node')
68      joyop = AckermannDriveJoyop(sys.argv[1:len(sys.argv)])
69      rospy.spin()
```

Programa C.1: Manejo con mando (Simulación)

```python
#!/usr/bin/env python

from tokenize import Double
import rospy
from ackermann_msgs.msg import AckermannDrive
from sensor_msgs.msg import Joy
from std_msgs.msg import Int16
from std_msgs.msg import Int32
from std_msgs.msg import Float32
import sys

class AckermannDriveJoyop:
    def __init__(self, args):
        if len(args)==1 or len(args)==2:
            self.max_speed = float(args[0])
            self.max_steering_angle = float(args[len(args)-1])
            vel_topic= 'set_velocity'
            angle_topic= 'set_angle'
        elif len(args) == 3:
            self.max_speed = float(args[0])
            self.max_steering_angle = float(args[1])
            vel_topic= 'set_velocity'
            angle_topic= 'set_angle'
        else:
            self.max_speed = 500
            self.max_steering_angle =180
            self.min_steering_angle =0.41853095368099036
            vel_topic= 'set_velocity'
            angle_topic= 'set_angle'
            angulo_pub='Angulo'
        self.speed = 0
        self.steering_angle = 0
        self.joy_sub = rospy.Subscriber('/joy', Joy, self.joy_callback)
        self.vel_pub=rospy.Publisher('/brushless_control/speed',Int32,
    queue_size=10)
        self.start_pub=rospy.Publisher('/brushless_control/stop_start',
    Int16,queue_size=10)
        self.angle_pub=rospy.Publisher('/manual_control/steering',Int16,
    queue_size=10)
        self.angulo_pub=rospy.Publisher('/angulo',Int16,queue_size=10)
        rospy.Timer(rospy.Duration(1.0/20.0), self.pub_callback, oneshot=
    False)
        rospy.loginfo('ackermann_drive_joyop_node initialized')

    def joy_callback(self, joy_msg):
        gamma=499
        if(joy_msg.axes[3]>0):
            self.steering_angle = int(1499-(joy_msg.axes[3] * gamma))
        if(joy_msg.axes[3]<0):
```

```
            self.steering_angle = int(1501-(joy_msg.axes[3] * gamma))
        if(joy_msg.axes[3]==0):
            self.steering_angle = int(1500)
        if(joy_msg.buttons[2]==1):
            self.start_pub.publish(0)
        if(joy_msg.buttons[0]==1):
            self.start_pub.publish(1)
    def pub_callback(self, event):
        self.angle_pub.publish(self.steering_angle)
        self.angulo_pub.publish(self.steering_angle)
        self.print_state()
    def print_state(self):
        sys.stderr.write('\x1b[2J\x1b[H')
        rospy.loginfo('\x1b[1M\r''\033[31;1m''Salida')
        rospy.loginfo('\x1b[1M\r'
                      '\033[34;1mSpeed: \033[32;1m%0.2f m/s, '
                      '\033[34;1mSteering Angle: \033[32;1m%0.2f rad\033[0
    m',
                      self.speed, self.steering_angle)
    def finalize(self):
        rospy.loginfo('Halting motors, aligning wheels and exiting...')
        sys.exit()
if __name__ == '__main__':
    rospy.init_node('ackermann_drive_joyop_node')
    joyop = AckermannDriveJoyop(sys.argv[1:len(sys.argv)])
    rospy.spin()
```

Programa C.2: Manejo con mando (Implementación física)

```
bag = rosbag("rosbag.bag");
imageBag = select(bag,'Topic','/imageROI');
anglesBag = select(bag,'Topic','/angulo');
imageMsgs = readMessages(imageBag);
anglesMsgs = readMessages(anglesBag);

ts1 = timeseries(imageBag);
ts2 = timeseries(anglesBag);
t1 = ts1.Time;
t2 = ts2.Time;

k = 1;
if size(t2,1) > size(t1,1)
    for i = 1:size(t1,1)
        [val,indx] = min(abs(t1(i) - t2));
        if val <= 0.1
            idx(k,:) = [i indx];
            k = k + 1;
        end
    end
else
```

```
    for i = 1:size(t2,1)
        [val,indx] = min(abs(t2(i) - t1));
        if val <= 0.1
            idx(k,:) = [indx i];
            k = k + 1;
        end
    end
end

steeringFilesPath = fullfile('D:\filib\Documents\Maestria','steeringReal')
    ;
imageFilesPath = fullfile('D:\filib\Documents\Maestria','
    ImagesCompletoDentro2');
if ~exist(imageFilesPath,'dir')
    mkdir(imageFilesPath);
end
if ~exist(steeringFilesPath,'dir')
    mkdir(steeringFilesPath);
end

for i = 1:length(idx)
    Im = readImage(imageMsgs{idx(i,1)});
    St = anglesMsgs{idx(i,2)}.Data;
    n_strPadded = sprintf('%04d',i+10239);
    vector(i,1)=i+10239;
    vector(i,2)=St;
    imageFileName = strcat(imageFilesPath,'/',n_strPadded,'.png');
    imwrite(Im,imageFileName);
end
writematrix(vector,'steering.xls')
```

Programa C.3: Extracción y guardado para imagen con ángulo de dirección

```
bag = rosbag("fuera_obs.bag");

imageBag = select(bag,'Topic','/imageROI_CUT');
anglesBag = select(bag,'Topic','/angulo');
cloudBag = select(bag,'Topic','/scan');
imageMsgs = readMessages(imageBag);
anglesMsgs = readMessages(anglesBag);
lidarMsgs = readMessages(cloudBag);

ts1 = timeseries(imageBag);
ts2 = timeseries(anglesBag);
ts3=timeseries(cloudBag);
t1 = ts1.Time;  %image
t2 = ts2.Time;  %angle
t3 = ts3.Time;  %lidar
k = 1;

```

```
%image
if size(t2,1) > size(t1,1) && size(t3,1) > size(t1,1)
    for i = 1:size(t1,1)
        [val1,indx1] = min(abs(t1(i) - t2));
        [val2,indx2] = min(abs(t1(i) - t3));
        if val1 <= 0.1 && val2 <=0.1
            idx1(k,:) = [i indx1];
            idx2(k,:)=[i indx2];
            k = k + 1;
        end
    end
end
%angle
if size(t1,1) > size(t2,1) && size(t3,1) > size(t2,1)
    for i = 1:size(t2,1)
        [val1,indx1] = min(abs(t2(i) - t1));
        [val2,indx2] = min(abs(t2(i) - t3));
        if val1 <= 0.1 && val2<=0.1
            idx1(k,:) = [i indx1];
            idx2(k,:)=[i indx2];
            k = k + 1;
        end
    end
end
%lidar
if size(t1,1) > size(t3,1) && size(t2,1) > size(t3,1)
    for i = 1:size(t3,1)
        [val1,indx1] = min(abs(t3(i) - t1));
        [val2,indx2] = min(abs(t3(i) - t2));
        if val1 <= 0.1 && val2<=0.1
            idx1(k,:) = [i indx1];
            idx2(k,:)=[i indx2];
            k = k + 1;
        end
    end
end
steeringFilesPath = fullfile('D:\filib\Documents\Maestria\data','steering3
    ');
imageFilesPath = fullfile('D:\filib\Documents\Maestria\data','Images3');
lidarFilesPath = fullfile('D:\filib\Documents\Maestria\data','lidar3');
if ~exist(imageFilesPath,'dir')
    mkdir(imageFilesPath);
end
if ~exist(steeringFilesPath,'dir')
    mkdir(steeringFilesPath);
end
if ~exist(lidarFilesPath,'dir')
    mkdir(lidarFilesPath);
end
```

```
%Comparacion respecto a imagen
if size(t2,1) > size(t1,1) && size(t3,1) > size(t1,1)
    for i = 1:length(idx1)
        Image = readImage(imageMsgs{i});
        Steering = anglesMsgs{idx1(i,2)}.Data;
        lidar=lidarMsgs{idx2(i,2)}.Ranges;
        n_strPadded = sprintf('%04d',i+21097);
        vector(i,1)=i+21097;
        vectorLidar(i,1)=i+21097;
        vector(i,2)=Steering;
        vectorLidar(i,2:361)=lidar';
        imageFileName = strcat(imageFilesPath,'/',n_strPadded,'.png');
        imwrite(Image,imageFileName);
    end
end
%Comparacion respecto a angulo
if size(t1,1) > size(t2,1) && size(t3,1) > size(t2,1)
    for i = 1:length(idx1)
        Image = readImage(imageMsgs{idx1(i,2)});
        Steering = anglesMsgs{i}.Data;
        lidar=lidarMsgs{idx2(i,2)}.Ranges;
        n_strPadded = sprintf('%04d',i+21097);
        vector(i,1)=i+21097;
        vectorLidar(i,1)=i+21097;
        vector(i,2)=Steering;
        vectorLidar(i,2:361)=lidar';
        imageFileName = strcat(imageFilesPath,'/',n_strPadded,'.png');
        imwrite(Image,imageFileName);
    end
end
%Comparacion respecto a lidar
if size(t1,1) > size(t3,1) && size(t2,1) > size(t3,1)
    for i = 1:length(idx1)
        Image = readImage(imageMsgs{idx1(i,2)});
        Steering = anglesMsgs{idx2(i,2)}.Data;
        lidar=lidarMsgs{i}.Ranges;
        n_strPadded = sprintf('%04d',i+21097);
        vector(i,1)=i+21097;
        vectorLidar(i,1)=i+21097;
        vector(i,2)=Steering;
        vectorLidar(i,2:361)=lidar';
        imageFileName = strcat(imageFilesPath,'/',n_strPadded,'.png');
        imwrite(Image,imageFileName);
    end
end
writematrix(vector,'steeringfuera_obs.xls')
writematrix(vectorLidar,'Lidarfuera_obs.xlsx')
```

Programa C.4: Extracción y guardado para imagen de cámara con ángulo de dirección y LiDAR

```
lidarFilesPath = fullfile('D:\filib\Documents\Maestria\data','lidar3');
lidarName = 'Lidarfuera_obs.xlsx';
lidarData = readtable(lidarName);

lidarData = lidarData{:,:};

for k=1:size(lidarData,1)
    label=lidarData(k,1);
    matrizImg=zeros(20,18);
    flag=2;
    for i=1:size(matrizImg,1)
        for j=1:size(matrizImg,2)
            if lidarData(k,flag)>1
                matrizImg(i,j)=0;
            else
                value=(lidarData(k,flag)/1)*255;
                matrizImg(i,j)=value;
            end
            flag=flag+1;
        end
    end
    label= sprintf('%04d',label);
    lidarFileName = strcat(lidarFilesPath,'/',label,'.png');
    imwrite(uint8(matrizImg), lidarFileName)

end
```

Programa C.5: Conversión de vectores LiDAR a imágenes

Apéndice D

Programas computacionales para entrenamiento de CNN

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

from keras.models import Sequential
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Activation, Flatten, Dropout,
    BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers
import pandas as pd
import numpy as np
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
import random
from keras.utils.vis_utils import plot_model

random.seed(0)
SEED = 0

def set_seeds(seed=SEED):
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    tf.random.set_seed(seed)
    np.random.seed(seed)
```

```
27
28 def set_global_determinism(seed=SEED):
29     set_seeds(seed=seed)
30     os.environ['TF_DETERMINISTIC_OPS'] = '1'
31     os.environ['TF_CUDNN_DETERMINISTIC'] = '1'
32     tf.config.threading.set_inter_op_parallelism_threads(1)
33     tf.config.threading.set_intra_op_parallelism_threads(1)
34 set_global_determinism(seed=SEED)
35 def append_ext(fn):
36     return str(fn)+".png"
37 def normalize(fn):
38     return float(float(fn)/180.)
39
40 train=pd.read_excel('trainROI/steering.xls',dtype=str)
41
42 train["id"]=train["id"].apply(append_ext)
43 train["label"]=train["label"].apply(normalize)
44
45 datagen=ImageDataGenerator(rescale=1./255.,validation_split=0.1)
46
47 train_generator=datagen.flow_from_dataframe(
48                                                 dataframe=train,
49                                                 directory="trainROI/Images/",
50                                                 x_col="id",
51                                                 y_col="label",
52                                                 color_mode='rgb',
53                                                 subset="training",
54                                                 batch_size=66,
55                                                 seed=42,
56                                                 shuffle=True,
57                                                 class_mode='other',
58                                                 target_size=(64,64)
59                                                 )
60
61 valid_generator=datagen.flow_from_dataframe(
62                                                 dataframe=train,
63                                                 directory="trainROI/Images/",
64                                                 x_col="id",
65                                                 y_col="label",
66                                                 color_mode='rgb',
67                                                 subset="validation",
68                                                 batch_size=1,
69                                                 seed=42,
70                                                 shuffle=True,
71                                                 class_mode='other',
72                                                 target_size=(64,64))
73
74 initializer = tf.keras.initializers.HeNormal(seed = 0)
75 model = Sequential()
```

```
76
77 model.add(Conv2D(12, (5, 5), input_shape=(64,64, 3), activation='elu',
       kernel_initializer = initializer,padding='same'))
78 model.add(Conv2D(36, (5, 5), (2, 2), activation='elu',kernel_initializer =
       initializer,padding='same'))
79 model.add(Conv2D(48, (5, 5), (2, 2), activation='elu',kernel_initializer =
       initializer,padding='same'))
80 model.add(Conv2D(64, (3, 3), activation='elu',kernel_initializer =
       initializer,padding='same'))
81 model.add(Conv2D(64, (3, 3), activation='elu',kernel_initializer =
       initializer,padding='same'))
82
83 model.add(Flatten())
84 model.add(Dense(10, activation = 'elu',kernel_initializer = initializer))
85 model.add(Dense(5, activation = 'elu',kernel_initializer = initializer))
86 model.add(Dense(3, activation = 'elu',kernel_initializer = initializer))
87 model.add(Dense(1,kernel_initializer = initializer))
88
89 model.compile(Adam(lr=0.001),loss='mse',metrics=['mse'])
90
91 model.summary()
92 plot_model(model, show_shapes=True, to_file='CNNsimulacion1.png')
93 STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
94 STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size
95
96 history=model.fit(train_generator,
97                     steps_per_epoch=STEP_SIZE_TRAIN,
98                     validation_data=valid_generator,
99                     validation_steps=STEP_SIZE_VALID,
100                    epochs=23)
101
102 score=model.evaluate(valid_generator,
103 steps=1)
104 print("score",score)
105 hist=history.history
106
107 model.save('modelSimu.h5',
108                 overwrite=True,
109                 include_optimizer=True,
110                 save_format='h5')
111
112 plt.plot(hist['loss'], '.-')
113 plt.plot(hist['val_loss'], '.-')
114 plt.ylabel('loss')
115 plt.xlabel('epoch')
116 plt.legend(['train', 'validation'], loc='upper right')
117 plt.show()
```

Programa D.1: Modelo propuesto para CNN (tarea: Navegación autónoma sin obstáculos en simulación)

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

from keras.models import Sequential
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Input, Flatten, Conv2D, Dropout,
    MaxPooling2D
from keras import regularizers, optimizers
import pandas as pd
import numpy as np
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
import random
from keras.layers.merge import concatenate
from keras.models import Model
from keras.utils.vis_utils import plot_model

random.seed(0)
SEED = 0
def set_seeds(seed=SEED):
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    tf.random.set_seed(seed)
    np.random.seed(seed)
def set_global_determinism(seed=SEED):
    set_seeds(seed=seed)
    os.environ['TF_DETERMINISTIC_OPS'] = '1'
    os.environ['TF_CUDNN_DETERMINISTIC'] = '1'
    tf.config.threading.set_inter_op_parallelism_threads(1)
    tf.config.threading.set_intra_op_parallelism_threads(1)

set_global_determinism(seed=SEED)

def append_ext(fn):
    return str(fn)+".png"

def normalize(fn):
    return float(float(fn)/180.)

trainAngle=pd.read_excel('dataObs_simu/steering.xls')
trainAngle["id"]=trainAngle["id"].apply(append_ext)
trainAngle["label"]=trainAngle["label"].apply(normalize)

def get_flow_from_dataframe(subset):
    datagen1=ImageDataGenerator(rescale=1./255.,brightness_range
   =[0.3,0.9],validation_split=0.1)
```

```
    datagen2=ImageDataGenerator(rescale=1./255.,validation_split=0.1)
    trainImage_generator=datagen1.flow_from_dataframe(
                                            dataframe=trainAngle,
                                            directory="dataObs_simu/Images/",
                                            x_col="id",
                                            y_col="label",
                                            color_mode='rgb',
                                            subset=subset,
                                            batch_size=50,
                                            seed=42,
                                            shuffle=True,
                                            class_mode='other',
                                            target_size=(64,128)
                                            )
    trainLidar_generator=datagen2.flow_from_dataframe(
                                            dataframe=trainAngle,
                                            directory="dataObs_simu/lidar/",
                                            x_col="id",
                                            y_col="label",
                                            color_mode='grayscale',
                                            subset=subset,
                                            batch_size=50,
                                            seed=42,
                                            shuffle=True,
                                            class_mode='other',
                                            target_size=(20,18)
                                            )
    while True:
        x_1 = trainImage_generator.next()
        x_2 = trainLidar_generator.next()
        yield [x_1[0], x_2[0]], x_1[1]

initializer = tf.keras.initializers.HeNormal(seed = 0)

# define the model
## channel 1
inputs1 = Input(shape=(64,128, 3))
conv1_1 = Conv2D(12, (5,5), activation='elu',kernel_initializer = initializer,padding='same')(inputs1)
conv1_2 = Conv2D(36, (5,5), (2,2), activation='elu',kernel_initializer = initializer,padding='same')(conv1_1)
conv1_3 = Conv2D(48, (5, 5), (2, 2), activation='elu',kernel_initializer = initializer,padding='same')(conv1_2)
conv1_4 = Conv2D(64, (3, 3), (2,2),activation='elu',kernel_initializer = initializer,padding='same')(conv1_3)
flat1 = Flatten()(conv1_4)
## channel 2
```

```
inputs2 = Input(shape=(20,18, 1))
conv2_1 = Conv2D(12, (5,5), activation='elu',kernel_initializer =
    initializer,padding='same')(inputs2)
conv2_2 = Conv2D(36, (5,5), (2,2), activation='elu',kernel_initializer =
    initializer,padding='same')(conv2_1)
conv2_3 = Conv2D(48, (5, 5), (2, 2), activation='elu',kernel_initializer =
     initializer,padding='same')(conv2_2)
conv2_4 = Conv2D(64, (3, 3), (2,2),activation='elu',kernel_initializer =
    initializer,padding='same')(conv2_3)
flat2 = Flatten()(conv2_4)
 # merge
merged = concatenate([flat1, flat2])
 # interpretation
dense1 = Dense(25, activation='elu',kernel_initializer=initializer)(merged
    )
drop1 = Dropout(0.25,seed=42)(dense1)
dense2 = Dense(15, activation = 'elu',kernel_initializer = initializer)(
    drop1)
drop2 = Dropout(0.25,seed=42)(dense2)
dense3 = Dense(10, activation = 'elu',kernel_initializer = initializer)(
    drop2)
outputs = Dense(1,kernel_initializer = initializer)(dense3)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
 # compile
model.compile(Adam(lr=0.001),loss='mse',metrics=['mse'])
 # summarize
model.summary()
plot_model(model, show_shapes=True, to_file='multichannel.png')

train_gen = get_flow_from_dataframe(subset='training')
valid_gen = get_flow_from_dataframe(subset='validation')

STEP_SIZE_TRAIN=38178//100
STEP_SIZE_VALID=1
history=model.fit(train_gen,validation_data=valid_gen,steps_per_epoch=
    STEP_SIZE_TRAIN,validation_steps=STEP_SIZE_VALID,epochs=19)

hist=history.history

model.save('modelObsSimu.h5',
                overwrite=True,
                include_optimizer=True,
                save_format='h5')

plt.plot(hist['loss'], '.-')
plt.plot(hist['val_loss'], '.-')
```

Programa D.2: Modelo propuesto para CNN (tarea: Navegación autónoma con obstáculos estáticos en simulación)

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

from keras.models import Sequential
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Input, Flatten, Conv2D, Dropout,
    MaxPooling2D
from keras import regularizers, optimizers
import pandas as pd
import numpy as np
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
import random
from keras.layers.merge import concatenate
from keras.models import Model
from keras.utils.vis_utils import plot_model
from keras.regularizers import l2

random.seed(0)
SEED = 0

def set_seeds(seed=SEED):
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    tf.random.set_seed(seed)
    np.random.seed(seed)
def set_global_determinism(seed=SEED):
    set_seeds(seed=seed)
    os.environ['TF_DETERMINISTIC_OPS'] = '1'
    os.environ['TF_CUDNN_DETERMINISTIC'] = '1'
    tf.config.threading.set_inter_op_parallelism_threads(1)
    tf.config.threading.set_intra_op_parallelism_threads(1)
set_global_determinism(seed=SEED)

def append_ext(fn):
    return str(fn)+".png"
def normalize(fn):
    return float(float(fn)/180.)

trainAngle=pd.read_excel('steering.xls')
trainAngle["id"]=trainAngle["id"].apply(append_ext)
trainAngle["label"]=trainAngle["label"].apply(normalize)

def get_flow_from_dataframe(subset):
    datagen1=ImageDataGenerator(rescale=1./255.,validation_split=0.1)
    datagen2=ImageDataGenerator(rescale=1./255.,validation_split=0.1)
```

```
    trainImage_generator=datagen1.flow_from_dataframe(
                                                dataframe=trainAngle,
                                                directory="Images/",
                                                x_col="id",
                                                y_col="label",
                                                color_mode='rgb',
                                                subset=subset,
                                                batch_size=50,
                                                seed=42,
                                                shuffle=True,
                                                class_mode='other',
                                                target_size=(64,128)
                                                )
    trainLidar_generator=datagen2.flow_from_dataframe(
                                                dataframe=trainAngle,
                                                directory="lidar/",
                                                x_col="id",
                                                y_col="label",
                                                color_mode='grayscale',
                                                subset=subset,
                                                batch_size=50,
                                                seed=42,
                                                shuffle=True,
                                                class_mode='other',
                                                target_size=(20,18)
                                                )
    while True:
        x_1 = trainImage_generator.next()
        x_2 = trainLidar_generator.next()
        yield [x_1[0], x_2[0]], x_1[1]
initializer = tf.keras.initializers.HeNormal(seed = 0)

# define the model
## channel 1
inputs1 = Input(shape=(64,128, 3))
conv1_1 = Conv2D(12, (5,5), activation='elu',kernel_initializer =
   initializer,padding='same')(inputs1)
conv1_2 = Conv2D(36, (5,5), (2,2), activation='elu',kernel_initializer =
   initializer,padding='same')(conv1_1)
conv1_3 = Conv2D(48, (5, 5), (2, 2), activation='elu',kernel_initializer =
    initializer,padding='same')(conv1_2)
conv1_4 = Conv2D(64, (3, 3), (2,2),activation='elu',kernel_initializer =
   initializer,padding='same')(conv1_3)
conv1_5 = Conv2D(128, (3, 3), (2,2),activation='elu',kernel_initializer =
   initializer,padding='same')(conv1_4)
flat1 = Flatten()(conv1_5)
## channel 2
inputs2 = Input(shape=(20,18, 1))
conv2_1 = Conv2D(12, (5,5), activation='elu',kernel_initializer =
```

```
    initializer,padding='same')(inputs2)
conv2_2 = Conv2D(36, (5,5), (2,2), activation='elu',kernel_initializer =
    initializer,padding='same')(conv2_1)
conv2_3 = Conv2D(48, (5, 5), (2, 2), activation='elu',kernel_initializer =
     initializer,padding='same')(conv2_2)
conv2_4 = Conv2D(64, (3, 3), (2,2),activation='elu',kernel_initializer =
    initializer,padding='same')(conv2_3)
conv2_5 = Conv2D(128, (3, 3), (2,2),activation='elu',kernel_initializer =
    initializer,padding='same')(conv2_4)
flat2 = Flatten()(conv2_5)
 # merge
merged = concatenate([flat1, flat2])
dense0 = Dense(100, activation='elu',kernel_initializer=initializer,
    kernel_regularizer=l2(0.01))(merged)
dense1 = Dense(50, activation='elu',kernel_initializer=initializer,
    kernel_regularizer=l2(0.01))(dense0)
dense2 = Dense(15, activation = 'elu',kernel_initializer = initializer)(
    dense1)
dense3 = Dense(10, activation = 'elu',kernel_initializer = initializer)(
    dense2)
outputs = Dense(1,kernel_initializer = initializer)(dense3)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
 # compile
model.compile(Adam(lr=0.001),loss='mse',metrics=['mse'])
 # summarize
model.summary()
plot_model(model, show_shapes=True, to_file='multichannel.png')

train_gen = get_flow_from_dataframe(subset='training')
valid_gen = get_flow_from_dataframe(subset='validation')

STEP_SIZE_TRAIN=38178//100
STEP_SIZE_VALID=1
history=model.fit(train_gen,validation_data=valid_gen,steps_per_epoch=
    STEP_SIZE_TRAIN,validation_steps=STEP_SIZE_VALID,epochs=20)
hist=history.history

model.save('modelObsSimuR3.h5',
                overwrite=True,
                include_optimizer=True,
                save_format='h5')
plt.plot(hist['loss'], '.-')
plt.plot(hist['val_loss'], '.-')
```

Programa D.3: Modelo propuesto para CNN (tarea: Navegación autónoma con obstáculos móviles en simulación)

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

from keras.models import Sequential
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Activation, Flatten, Dropout,
    BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers
import pandas as pd
import numpy as np
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
import random
from PIL import Image, ImageFilter
from keras.regularizers import l2, l1_l2
from keras.utils.vis_utils import plot_model
random.seed(0)
SEED = 0

def set_seeds(seed=SEED):
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    tf.random.set_seed(seed)
    np.random.seed(seed)
def set_global_determinism(seed=SEED):
    set_seeds(seed=seed)
    os.environ['TF_DETERMINISTIC_OPS'] = '1'
    os.environ['TF_CUDNN_DETERMINISTIC'] = '1'
    tf.config.threading.set_inter_op_parallelism_threads(1)
    tf.config.threading.set_intra_op_parallelism_threads(1)
set_global_determinism(seed=SEED)
def append_ext(fn):
    return str(fn)+".png"
def normalize(fn):
    return float((float(fn)/1000.)-1)
train=pd.read_excel('steeringReal/steering1.xls')
train["id"]=train["id"].apply(append_ext)
train["label"]=train["label"].apply(normalize)

datagen1=ImageDataGenerator(rescale=1./255.,validation_split=0.1)
train_generator=datagen1.flow_from_dataframe(
                                            dataframe=train,
                                            directory="imagesWifi/ImagesV4
    /",
                                            x_col="id",
```

```
                                                y_col="label",
                                                color_mode='rgb',
                                                subset="training",
                                                batch_size=90,
                                                seed=42,
                                                shuffle=True,
                                                class_mode='other',
                                                target_size=(64,64)
                                                )
valid_generator=datagen1.flow_from_dataframe(
                                                dataframe=train,
                                                directory="imagesWifi/ImagesV4
    /",
                                                x_col="id",
                                                y_col="label",
                                                color_mode='rgb',
                                                subset="validation",
                                                batch_size=1,
                                                seed=42,
                                                shuffle=True,
                                                class_mode='other',
                                                target_size=(64,64))

initializer = tf.keras.initializers.HeNormal(seed = 0)
model = Sequential()

model.add(Conv2D(12, (5, 5), input_shape=(64,64, 3), activation='elu',
    kernel_initializer = initializer,padding='same'))
model.add(Conv2D(36, (5, 5), (2, 2), activation='elu',kernel_initializer =
     initializer,padding='same'))
model.add(Conv2D(48, (5, 5), (2, 2), activation='elu',kernel_initializer =
     initializer,padding='same'))
model.add(Conv2D(64, (3, 3), (2,2),activation='elu',kernel_initializer =
    initializer,padding='same'))
model.add(Flatten())
model.add(Dense(27, activation = 'elu',kernel_initializer = initializer))
model.add(Dropout(0.5,seed=42))
model.add(Dense(15, activation = 'elu',kernel_initializer = initializer))
model.add(Dropout(0.25,seed=42))
model.add(Dense(10, activation = 'elu',kernel_initializer = initializer))
model.add(Dropout(0.2,seed=42))
model.add(Dense(5, activation = 'elu',kernel_initializer = initializer))
model.add(Dropout(0.1,seed=42))
model.add(Dense(1,kernel_initializer = initializer))

model.compile(Adam(learning_rate=0.001),loss='mean_squared_error',metrics
    =['mse'])
model.summary()
plot_model(model, to_file='CNNimplementacion1.png', show_shapes=False,
```

```
    show_layer_names=False)
STEP_SIZE_TRAIN=train_generator.n//train_generator.batch_size
STEP_SIZE_VALID=valid_generator.n//valid_generator.batch_size

history=model.fit(train_generator,
                  steps_per_epoch=STEP_SIZE_TRAIN,
                  validation_data=valid_generator,
                  validation_steps=STEP_SIZE_VALID,
                  epochs=37
)

score=model.evaluate(valid_generator,
steps=1)
print("score",score)
hist=history.history

model.save('modelWifiRealFinal3.h5',
               overwrite=True,
               include_optimizer=True,
               save_format='h5')

plt.plot(hist['loss'], '.-')
plt.plot(hist['val_loss'], '.-')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```

Programa D.4: Modelo propuesto para CNN (tarea: Navegación autónoma sin obstáculos en implementación física)

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

from keras.models import Sequential
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Input, Flatten, Conv2D, Dropout,
    MaxPooling2D, Add
from keras import regularizers, optimizers
import pandas as pd
import numpy as np
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
import random
from keras.layers.merge import concatenate
from keras.models import Model
from keras.utils.vis_utils import plot_model
```

```
from keras.regularizers import l2
from tensorflow.keras.utils import model_to_dot
import pydot
random.seed(0)
SEED = 0

def set_seeds(seed=SEED):
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    tf.random.set_seed(seed)
    np.random.seed(seed)
def set_global_determinism(seed=SEED):
    set_seeds(seed=seed)
    os.environ['TF_DETERMINISTIC_OPS'] = '1'
    os.environ['TF_CUDNN_DETERMINISTIC'] = '1'
    tf.config.threading.set_inter_op_parallelism_threads(1)
    tf.config.threading.set_intra_op_parallelism_threads(1)
set_global_determinism(seed=SEED)

def append_ext(fn):
    return str(fn)+".png"
def normalize(fn):
    return float((float(fn)/1000.)-1)

trainAngle=pd.read_excel('steering/steeringWifi.xls')
trainAngle["id"]=trainAngle["id"].apply(append_ext)
trainAngle["label"]=trainAngle["label"].apply(normalize)

def get_flow_from_dataframe(subset):
    datagen1=ImageDataGenerator(rescale=1./255.,validation_split=0.05)
    datagen2=ImageDataGenerator(rescale=1./255.,validation_split=0.05)

    trainImage_generator=datagen1.flow_from_dataframe(
                                            dataframe=trainAngle,
                                            directory="Images/",
                                            x_col="id",
                                            y_col="label",
                                            color_mode='rgb',
                                            subset=subset,
                                            batch_size=50,
                                            seed=42,
                                            shuffle=True,
                                            class_mode='other',
                                            target_size=(64,64)
                                            )
    trainLidar_generator=datagen2.flow_from_dataframe(
                                            dataframe=trainAngle,
                                            directory="lidarImg/",
                                            x_col="id",
```

```
                                                    y_col="label",
                                                    color_mode='grayscale',
                                                    subset=subset,
                                                    batch_size=50,
                                                    seed=42,
                                                    shuffle=True,
                                                    class_mode='other',
                                                    target_size=(20,18)
                                                    )
    while True:
        x_1 = trainImage_generator.next()
        x_2 = trainLidar_generator.next()
        yield [x_1[0], x_2[0]], x_1[1]
initializer = tf.keras.initializers.HeNormal(seed = 0)
# Valores de dropout que deseas probar para la primera capa Dropout
dropout_values_1 = [0]
# Valores de dropout que deseas probar para la segunda capa Dropout
neural_values=[100]
block=[200]
for neural_rate in neural_values:
    for block_rate in block:
        inputs1 = Input(shape=(64,64, 3))
        conv1_1 = Conv2D(12, (5,5), activation='elu',kernel_initializer =
    initializer,padding='same')(inputs1)
        conv1_2 = Conv2D(36, (5,5), (2,2), activation='elu',
    kernel_initializer = initializer,padding='same')(conv1_1)
        conv1_3 = Conv2D(64, (5, 5), (2, 2), activation='elu',
    kernel_initializer = initializer,padding='same')(conv1_2)
        conv1_4 = Conv2D(128, (5, 5), (2,2),activation='elu',
    kernel_initializer = initializer,padding='same')(conv1_3)
        conv1_5 = Conv2D(150, (5, 5), (2,2),activation='elu',
    kernel_initializer = initializer,padding='same')(conv1_4)
        conv1_6 = Conv2D(200, (5, 5), (2,2),activation='elu',
    kernel_initializer = initializer,padding='same')(conv1_5)
        flat1 = Flatten()(conv1_6)
        ## channel 2
        inputs2 = Input(shape=(20,18, 1))
        conv2_1 = Conv2D(12, (5,5), activation='elu',kernel_initializer =
    initializer,padding='same')(inputs2)
        conv2_2 = Conv2D(36, (5,5), (2,2), activation='elu',
    kernel_initializer = initializer,padding='same')(conv2_1)
        conv2_3 = Conv2D(64, (5, 5), (2, 2), activation='elu',
    kernel_initializer = initializer,padding='same')(conv2_2)
        conv2_4 = Conv2D(128, (5, 5), (2,2),activation='elu',
    kernel_initializer = initializer,padding='same')(conv2_3)
        conv2_5 = Conv2D(200, (5, 5), (2,2),activation='elu',
    kernel_initializer = initializer,padding='same')(conv2_4)
        flat2 = Flatten()(conv2_5)
        merged = concatenate([flat1, flat2])
```

```
106         dense42 = Dense(neural_rate, activation = 'elu',kernel_initializer
    = initializer,kernel_regularizer=l2(0.01))(merged)
107         dense41=Dense(50, activation = 'elu',kernel_initializer =
    initializer,kernel_regularizer=l2(0.01))(dense42)
108         dense422=Dense(20, activation = 'elu',kernel_initializer =
    initializer)(dense41)
109         dense44 = Dense(15, activation = 'elu',kernel_initializer =
    initializer)(dense422)
110         dense45 = Dense(10, activation = 'elu',kernel_initializer =
    initializer)(dense44)
111         dense46 = Dense(5, activation = 'elu',kernel_initializer=
    initializer)(dense45)
112         outputs = Dense(1,kernel_initializer = initializer)(dense46)
113         model = Model(inputs=[inputs1, inputs2], outputs=outputs)
114         model.compile(Adam(lr=0.001),loss='mse',metrics=['mse'])
115         ###################
116         model.summary()
117         plot_model(model, show_shapes=True, to_file='multichannel.png')
118
119         train_gen = get_flow_from_dataframe(subset='training')
120         valid_gen = get_flow_from_dataframe(subset='validation')
121
122         STEP_SIZE_TRAIN=56396//block_rate
123         STEP_SIZE_VALID=1
124         # Define el objeto EarlyStopping
125         early_stopping = EarlyStopping(monitor='loss', patience=5,
    restore_best_weights=True)
126         # Entrena el modelo con EarlyStopping como callback
127         history = model.fit(train_gen, validation_data=valid_gen,
    steps_per_epoch=STEP_SIZE_TRAIN, validation_steps=STEP_SIZE_VALID,
    epochs=33)
128         hist=history.history
129         print(f"Dropout rate 1: {block_rate}, Dropout rate 2: {neural_rate
    }, Test Loss: {hist['loss']}, Test Accuracy: {hist['loss']}")
130
131 model.save('modelObsWifi2V3.h5',
132                 overwrite=True,
133                 include_optimizer=True,
134                 save_format='h5')
135
136 plt.plot(hist['loss'], '.-')
137 plt.plot(hist['val_loss'], '.-')
138 plt.ylabel('loss')
139 plt.xlabel('epoch')
140 plt.legend(['train', 'validation'], loc='upper right')
141 plt.show()
```

Programa D.5: Modelo propuesto para CNN (tarea: Navegación autónoma con obstáculos estáticos en implementación física)

Apéndice E

Topologías de redes neuronales convolucionales (CNN)

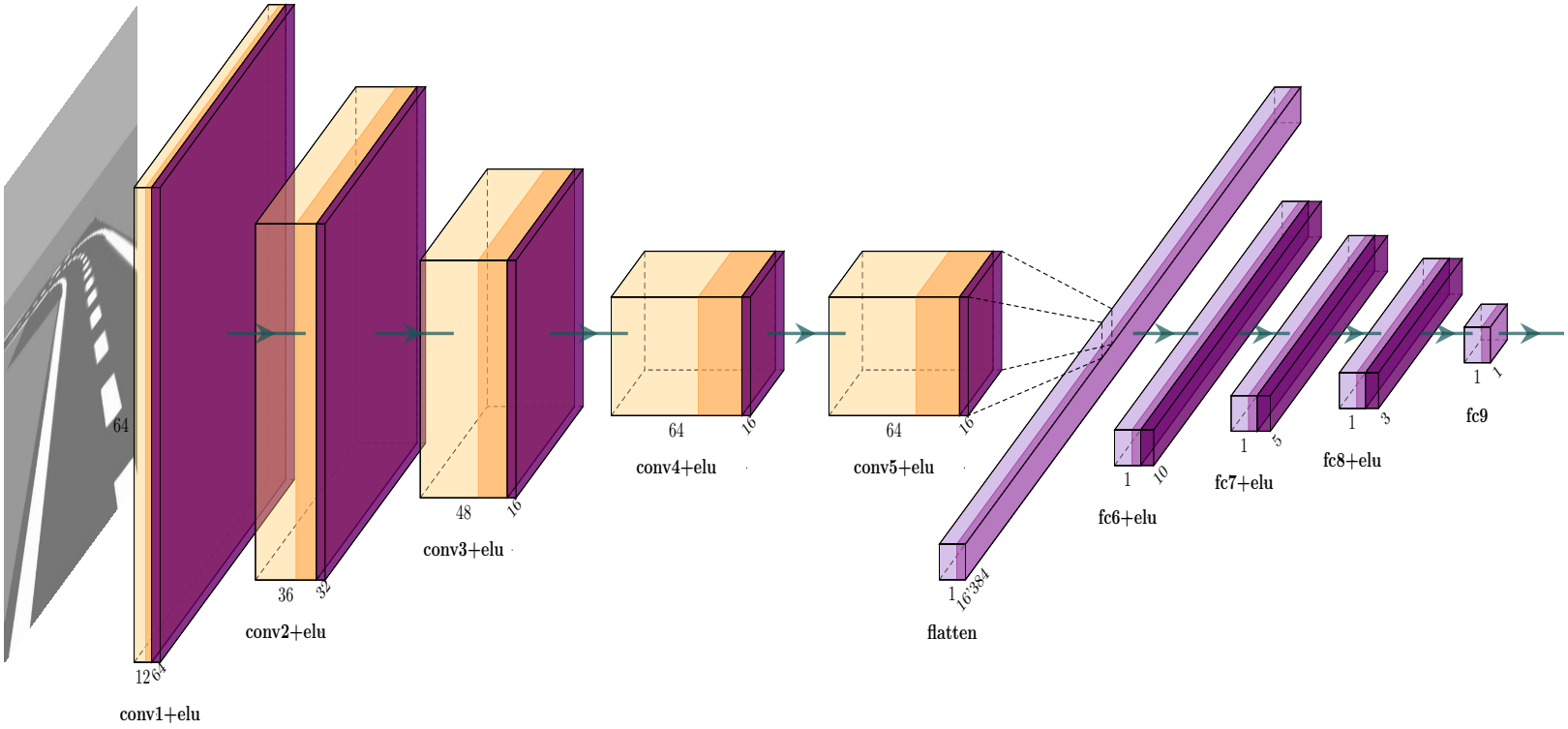


Figura E.1: CNN para la navegación autónoma sin obstáculos (Simulación)

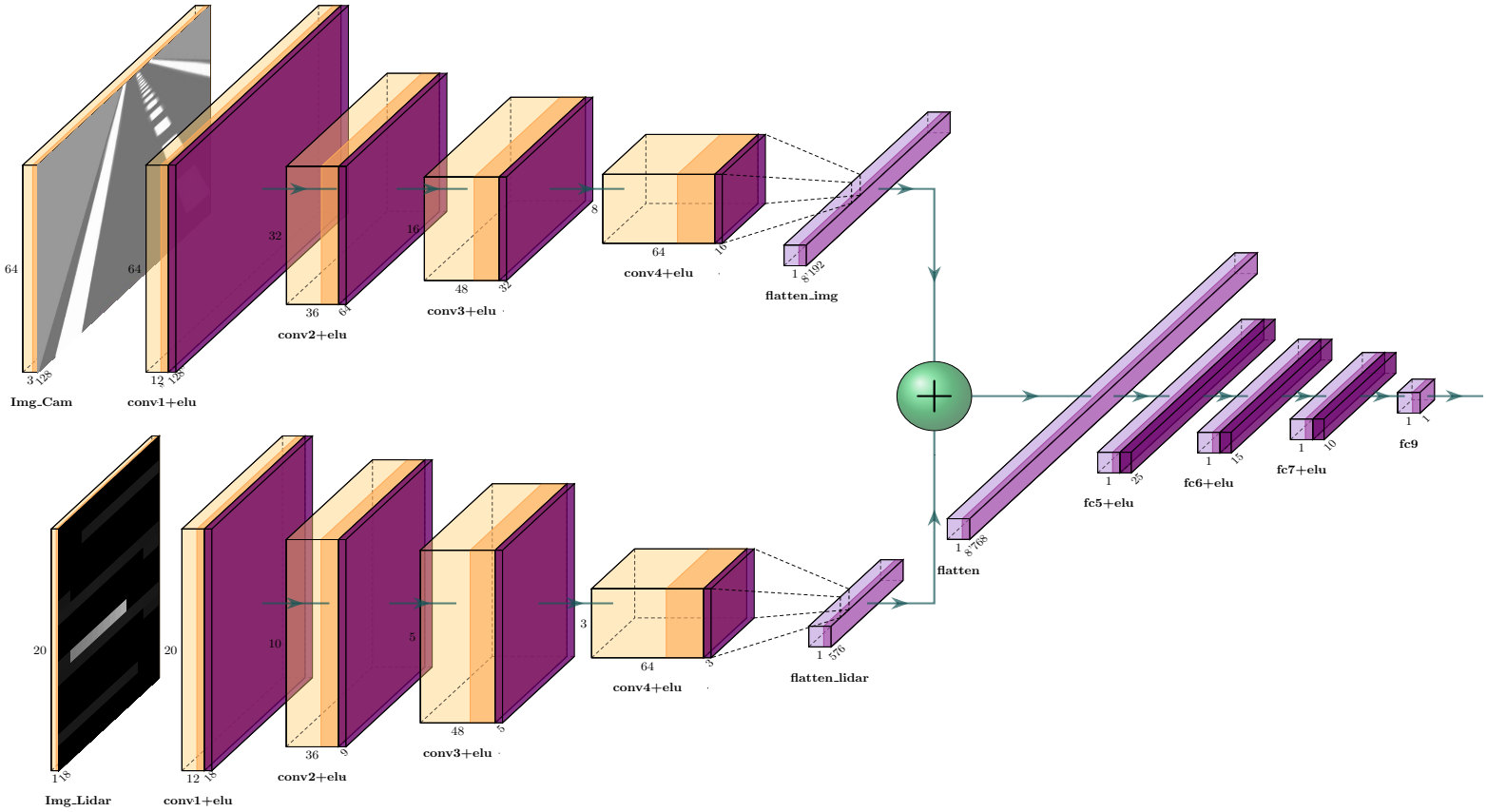


Figura E.2: CNN para la navegación autónoma con obstáculos estáticos (Simulación)

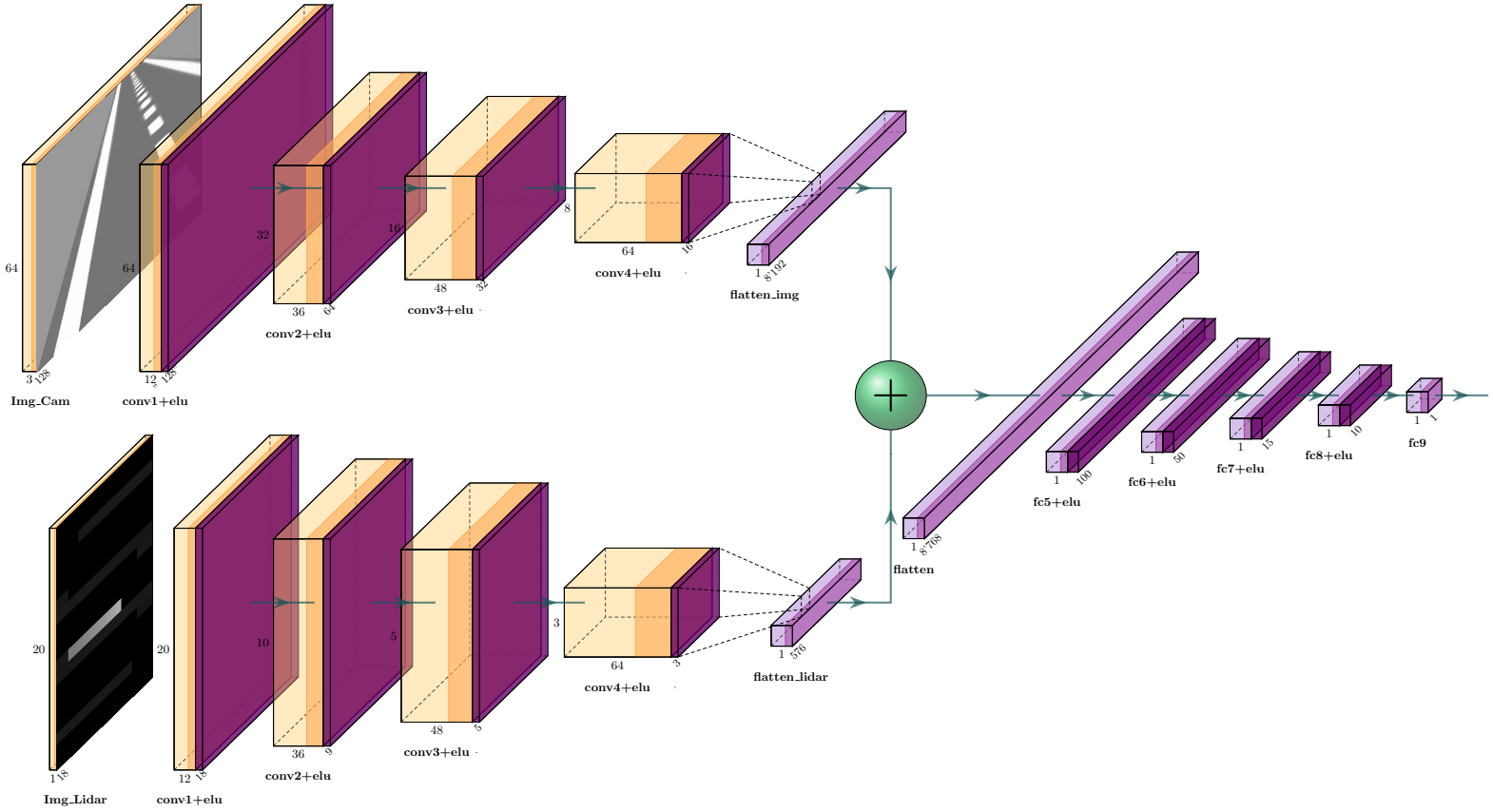


Figura E.3: CNN para la navegación autónoma con obstáculos móviles (Simulación)

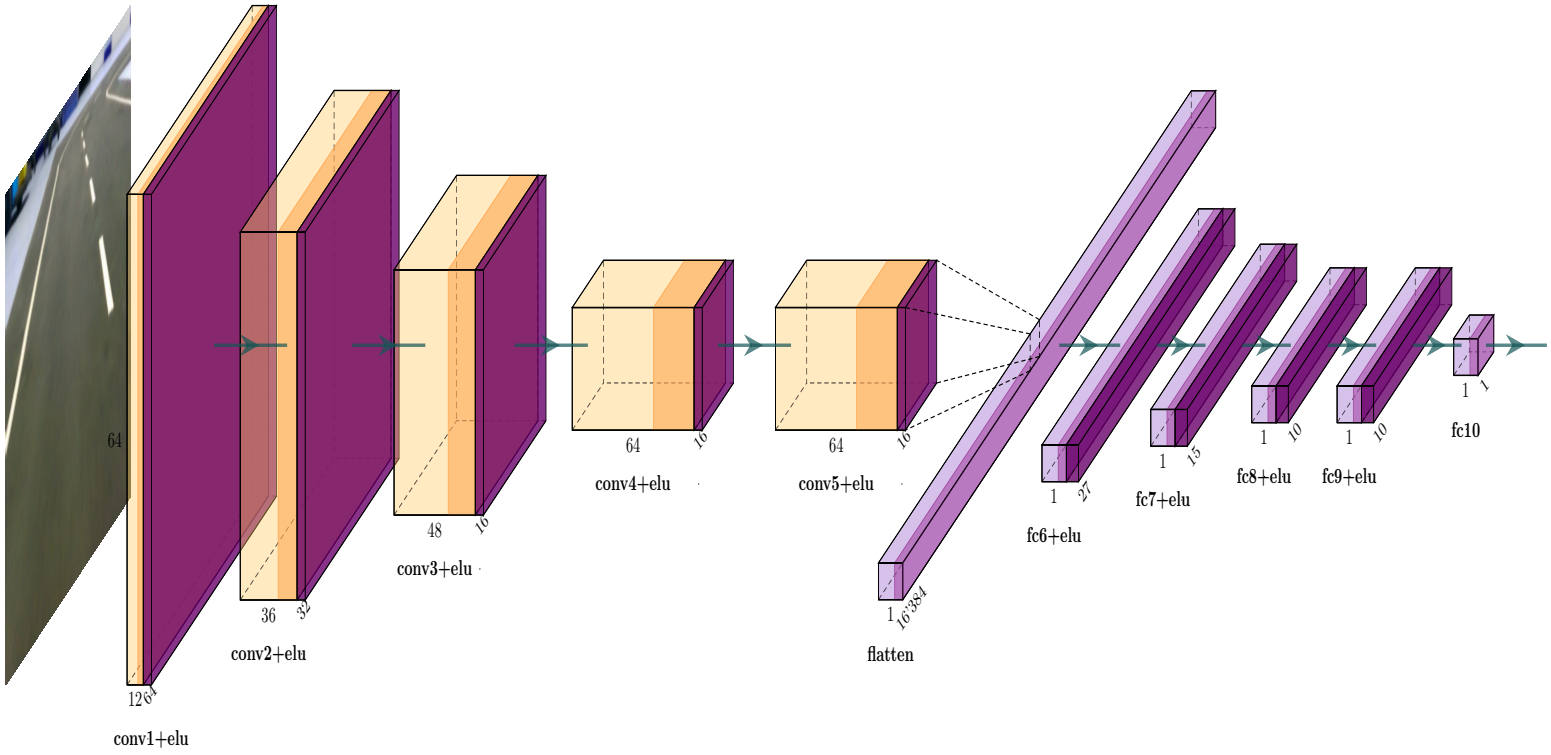


Figura E.4: CNN para la navegación autónoma sin obstáculos (Implementación)

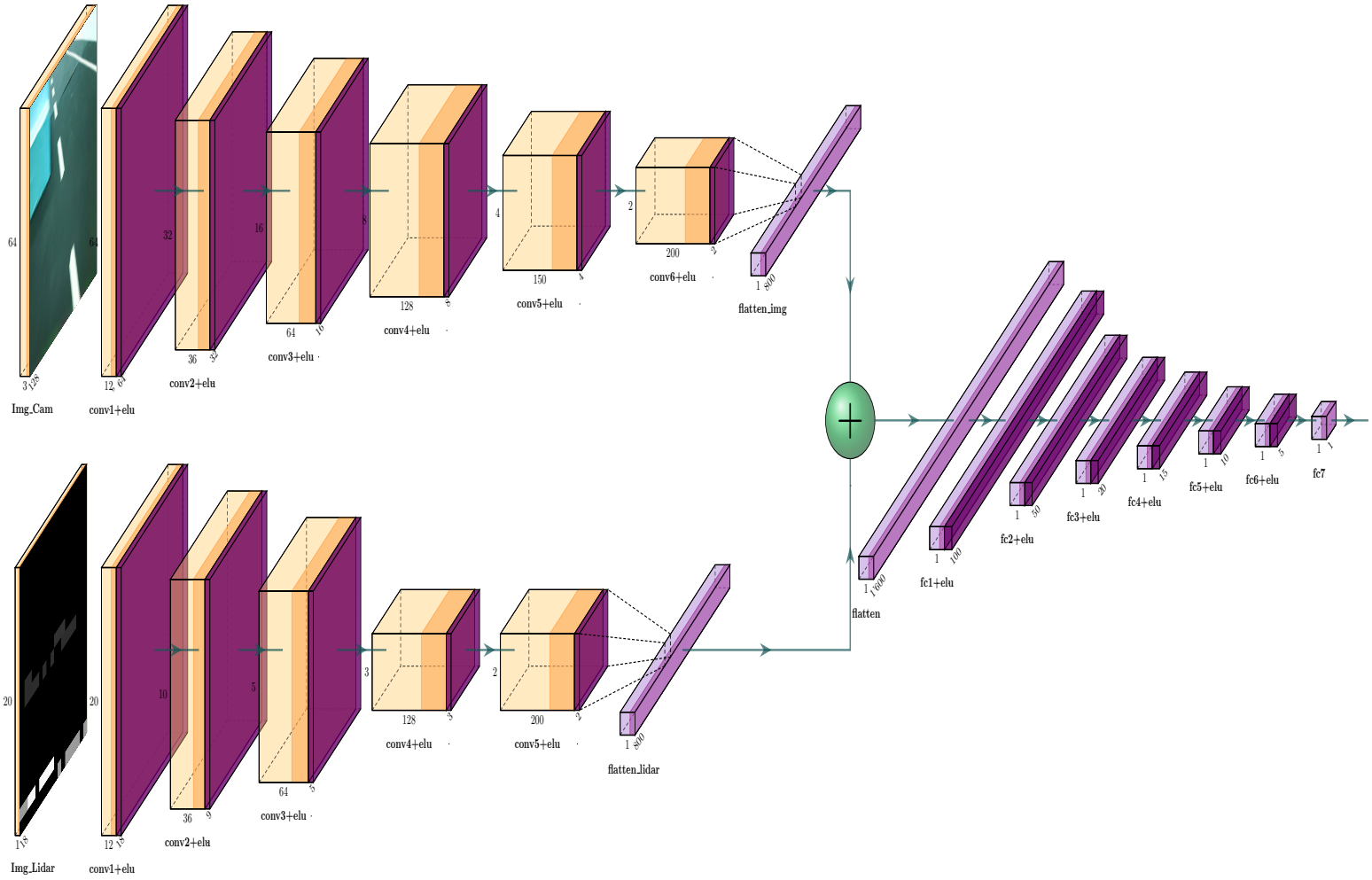


Figura E.5: CNN para la navegación autónoma con obstáculos estáticos (Implementación)

Apéndice F

Pruebas de hiperparámetros para las CNN

Neuronas	Menor error	Épocas
161	0.0203	14
86	0.0207	5
51	0.0203	9
46	0.0194	16
41	0. 0197	13
19	**0.0187**	**23**
9	0.04	20

Tabla F.1: Pruebas con diferentes neuronas en la CNN para simulación (reto uno)

Neuronas	Menor error	Épocas
100	0.015	13
60	0.012	15
51	**0.0101**	**19**
30	0. 0199	25

Tabla F.2: Pruebas con diferentes neuronas en la CNN para simulación (reto dos)

Neuronas	Menor error	Épocas
251	0.028	9
201	0.0150	16
186	**0.0103**	**20**
100	0. 0179	26

Tabla F.3: Pruebas con diferentes neuronas en la CNN para simulación (reto tres)

Neuronas	Menor error	Épocas
161	0.0409	18
86	0.0731	20
46	**0.0348**	**18**
41	0.0732	17

Tabla F.4: Pruebas con diferentes neuronas en la CNN para la implementación real (reto uno)

Neuronas	Menor error	Épocas
501	0.012	14
259	0.00901	18
201	**0.0052**	**33**
101	0.0701	25

Tabla F.5: Pruebas con diferentes neuronas en la CNN para implementación real(reto dos)