Universidad Internacional de La Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Algoritmos de inteligencia artificial para el desarrollo de controladores de conducción autónoma en TORCS

Trabajo Fin de Máster

Presentado por: Hernández García, Mario

Director/a: Fuentes Lorenzo, Damaris

Ciudad: Madrid
Fecha: 6/05/2021

Resumen

En este trabajo de fin de máster, se propone una comparativa entre diferentes algoritmos de inteligencia artificial destinados a resolver el problema de la conducción autónoma siendo implantados en el simulador de carreras TORCS. Los algoritmos involucrados en esta comparativa son: la lógica borrosa (FL), las redes neuronales artificiales (ANN) y las redes neuronales convolucionales (CNN).

Se ha seguido un proceso de desarrollo y optimización de los algoritmos propuestos, implementándolos en el simulador con la finalidad de analizar las ventajas y desventajas de la implantación y los resultados obtenidos en cada uno de ellos.

Por último, se ha realizado un análisis exhaustivo de los distintos controladores atendiendo a su desempeño en entorno cerrado y controlado. Aunque los experimentos propuestos han demostrado que la lógica borrosa es el algoritmo que presenta los mejores resultados en conjunto, no se ha podido determinar el mejor algoritmo de inteligencia artificial para la resolución del problema de conducción autónoma, ya que influye en gran medida tanto el entorno de pruebas como la optimización de los algoritmos en la fase de desarrollo.

Palabras Clave: Inteligencia Artificial, conducción autónoma, lógica borrosa, red neuronal.

Abstract

In this work, we propose a comparison between different artificial intelligence algorithms designed to solve the problem of autonomous driving, implemented in the racing simulator TORCS. The algorithms involved in this comparison are: fuzzy logic (FL), artificial neural networks (ANNs) and convolutional neural networks (CNNs).

Proposed algorithms development and optimization process in addition to the implementation in the simulator, will provide advantages and disadvantages of each one and results analysis.

An different controllers analysis has been made according to the performance inside a closed and controlled environment. The proposed experiments have shown that fuzzy logic is the algorithm that presents the best initial results. In any case, It has not been possible to determine the best artificial intelligence algorithm for solving the autonomous driving problem, since the testing environment and the optimization of the algorithms have a great influence in later results.

Keywords: Artificial intelligence, autonomous driving, fuzzy logic, neural network, TORCS.

Índice de contenidos

Índice de tablas

Índice de figuras

1. Introducción

En este apartado se describe de forma breve cada una de las partes del trabajo con el fin de obtener una idea general de qué se quiere hacer, por qué se hace y cómo se va a estructurar su desarrollo. Se divide en tres partes: motivación, planteamiento y estructura del trabajo.

1.1. Motivación

El rápido desarrollo de la inteligencia artificial y el aumento de popularidad en los últimos años han revolucionado numerosas áreas de investigación. Una de ellas es la de los coches autónomos, incorporando complejos modelos y algoritmos.

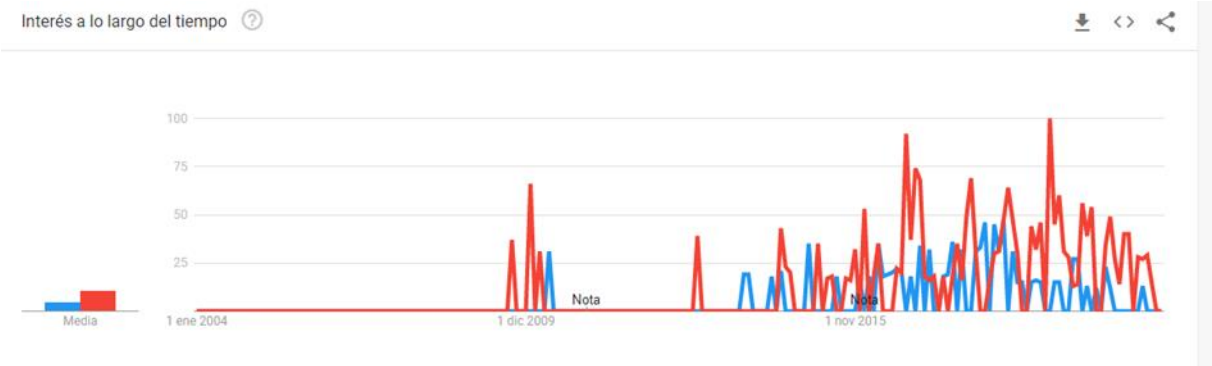

Figura 1. Búsquedas de 'conducción autónoma' y 'autonomuos car' en Google desde 2004.

Fuente: Google trends

Dejando de lado las implicaciones éticas o morales de la conducción autónoma, la implantación de robustos algoritmos que facilitan el funcionamiento de estos vehículos reducirá muchos problemas asociados a la conducción, como puede ser el peligro que supone el uso de un automóvil por parte de un conductor ebrio o con somnolencia.

Soluciones como la implantación de sensores LiDAR (Hecht, 2018) para la obtención de datos, o de técnicas más antiguas como las de RADAR (Göhring, Wang, Schnürmacher, & Ganjineh, 2011) que utiliza la empresa de automóviles Tesla en sus coches autónomos, hacen ver que cada vez se está más cerca de conseguir el objetivo de implantar un sistema de conducción autónoma viable y seguro, conviviendo con sistemas de conducción manuales.

Esas soluciones expuestas anteriormente, hacen referencia al método con el que se obtienen parámetros internos y del entorno (distancia a diferentes objetos, tamaño de estos), para más

tarde poder aplicar un algoritmo determinado. Este trabajo de fin de máster se centrará en los algoritmos de inteligencia artificial que pueden ser aplicados cuando ya se conocen a priori todas las variables necesarias, por lo que no trataremos con este tipo de sensores (LiDAR y RADAR). Las variables dadas permiten tener un conocimiento de todos los parámetros necesarios para la conducción autónoma y así, el foco de este trabajo consistirá en comparar la efectividad de los distintos algoritmos de inteligencia artificial en este caso concreto, con independencia de los errores o inexactitudes que se puedan derivar de los datos obtenidos por los sensores.

1.2. Planteamiento del trabajo

Aunque la implantación de vehículos auto tripulados no es nueva, no está documentada una solución que la aborde, realizando una comparativa entre diferentes algoritmos de inteligencia artificial y que además, mantenga fijos los demás parámetros de entorno. El mantener fijos estos parámetros a los que nos referimos, como puede ser el uso de los mismos circuitos de carreras, mismas condiciones ambientales, vehículos etc. permitirá una evaluación mucho más rigurosa, produciendo una independencia del resultado que proporcione el algoritmo de todas esas variables y evitando que puedan confundir o aleatorizar los resultados obtenidos.

Este trabajo determinará qué algoritmo de inteligencia artificial puede ser potencialmente más adecuado para resolver el problema de la conducción autónoma. Para ello, se implementarán los diferentes modelos en el simulador de carreras de código abierto TORCS, donde se verá el resultado de la aplicación de cada uno de estos algoritmos, estableciéndose conclusiones en base a los resultados obtenidos. Los modelos (en este caso también llamado controladores) son unos sistemas de control del automóvil que, a través de unas entradas proporcionadas tanto por sus parámetros internos como por su entorno físico, proporcionarán unas salidas acordes y que permitirán hacer una conducción del vehículo.

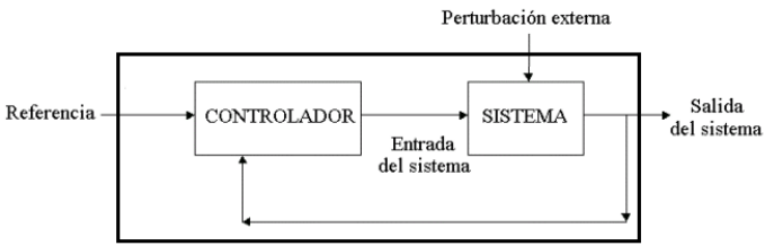

Figura 2. Descripción de un sistema de control con realimentación negativa.

Fuente: Google sites.

Primero, se realizará la implantación de un controlador borroso, que utiliza la denominada **lógica borrosa o difusa**. Esta técnica permite realizar una inferencia dados un conjunto de reglas y unas funciones de pertenencia a las variables asociadas. La principal ventaja de esta técnica es que no se necesita tener un conocimiento exhaustivo de la planta, ni tampoco tener en cuenta el modelo matemático que hay detrás de ella. Con la definición de unas reglas, que se concretan de manera lógica e intuitiva, se puede formar un primer modelo de lógica borrosa del que partir y mejorar.

Segundo, proponemos la implementación de **redes neuronales convolucionales** (CNN, del inglés *Convolutional Neural Network*). Estas redes se valen de las imágenes para obtener sus resultados. En este caso concreto de conducción autónoma, las imágenes serán obtenidas desde la perspectiva del conductor, son analizadas para determinar la acción a realizar, modificando las salidas de la red (acelerador, freno, giros, etc.). Este sistema, que podríamos clasificar dentro de la categoría de visión artificial, actúa de manera parecida a como lo haría un ser humano. A diferencia de la lógica borrosa, que hace uso de un conjunto de reglas para determinar su comportamiento y que se pueden enunciar de manera condicional (p.ej. si estamos en una curva cerrada y la velocidad es grande, entonces el vehículo frena), las redes neuronales convolucionales precisan de una gran cantidad de imágenes que hacen que funcionen el algoritmo de *backpropagation*. Este algoritmo es el encargado de corregir las salidas que no sean correctas del modelo actual, modificando cada uno de los pesos de la red y haciendo que disminuya la función de coste. El proceso de entrenamiento e incluso de predicción de esta técnica puede extenderse en el tiempo, ya que se trabaja con cientos de imágenes que contienen grandes cantidades de información cada una, es computacionalmente más costoso que las otras técnicas evaluadas en este trabajo.

Por último, se implementará la técnica de **redes neuronales artificiales** (ANN, del inglés Artificial Neural Network). A diferencia de las redes neuronales convolucionales, estas se valen de datos de entrada, en este caso proporcionados por los sensores del coche (velocidad, inercia, posición en la pista, ángulo respecto a la paralela de la pista, etc.) para determinar su salida. Al igual que las redes neuronales convolucionales, precisan de una gran cantidad de datos para poder hacer la función de entrenamiento. A mayor cantidad de datos, mayor será la corrección que se pueda hacer en la conducción frente al modelo base.

Ambas técnicas de redes neuronales precisan de los datos obtenidos por modelos que ya hacen una conducción correcta dentro de un circuito. Esto es debido a que se necesita saber cuál es una posición correcta dentro de la pista para poder “aprender” y corregir la función actual. La función o la red obtenida de modelos entrenados previamente, se aplicarán en unos

circuitos distintos al de entrenamiento para comprobar si en la fase de inferencia o predicción, el modelo es capaz de tomar buenas decisiones en entonos de conducción diferentes.

En términos generales, el objetivo del TFM (Trabajo de Fin de Máster) es **determinar qué algoritmo de inteligencia artificial es capaz de realizar una mejor conducción**, teniendo en cuenta aspectos como el trazado (que no produzca una conducción errática), el tiempo por vuelta o si se produce alguna salida de pista.

1.3. Estructura de la memoria

La estructura del trabajo consta de nueve bloques diferenciados:

- **Contexto y estado del arte:** Se hará una introducción teórica para aclarar conceptos de cada uno de los algoritmos de inteligencia artificial aplicados en la comparativa. Además, se realizará una breve contextualización de la situación actual de cada uno de los algoritmos de inteligencia artificial tanto de manera general (últimos avances y aplicaciones) como las aportaciones recientes en el problema de la conducción autónoma.

- **Objetivos concretos y metodología de trabajo:** Se describen una serie de objetivos a cumplir tanto a nivel general como específico, además de un plan de desarrollo para completar dichos objetivos.

- **Planteamiento de la comparativa:** Se propone un marco común de pruebas, de tal manera que todos los algoritmos partan de las mismas condiciones (mismos coches, mimos circuitos de pruebas) para así poder hacer una comparación justa.

- **Preparación del trabajo:** En este capítulo, se sigue todo el proceso de instalación y creación de los Robots que van a suponer una estructura donde implementar los controladores.

- Desarrollo del controlador de **lógica borrosa.**

- Desarrollo del controlador de **redes neuronales artificiales.**

- Desarrollo del controlador de **redes neuronales convolucionales.**

- **Discusión y análisis de resultados:** A raíz del desarrollo propuesto en el apartado anterior, se realizará una comparativa exhaustiva de cada uno de los algoritmos

propuestos. La información se propondrá en forma de tablas para que se presente la información útil de forma clara y concisa.

- **Conclusiones y trabajos futuros:** En este apartado se responderá a si se cumplieron los objetivos tanto generales como específicos propuestos. Además, se extraerán conclusiones de la información aportada en los resultados de la comparativa, intentando determinar si existe un mejor algoritmo para este contexto de aplicación o si, por el contrario, existen tanto ventajas como desventajas para cada uno de los modelos de conducción autónoma propuestos. Por último, se desarrollará de manera teórica una forma de realizar un desarrollo posterior al trabajo de fin de máster y se planteará de manera teórica la forma de implementar los desarrollos realizados en un coche real.

2. Contexto y estado del arte

Este capítulo se va a describir el contexto de aplicación del software necesario para realizar el trabajo, centrándolo en el simulador de carreras TORCS. También se estudiará más en profundidad el marco teórico y la situación del desarrollo existente de los algoritmos propuestos en la comparativa. Estos son el de lógica borrosa, las ANN y por último, las CNN.

2.1. TORCS

El simulador de carreras 3D de código abierto (TORCS, por sus siglas en inglés The Open Racing Car Simulator - Wymann et al., 2000), es un simulador modular, multiagente y multijugador fundado en 2007 por Guionneau Christophe y Eric Espie.

El desarrollo de TORCS empezó en 1997 como un juego en dos dimensiones llamado Racing Car Simulator (RCS). Estaba influenciado en el simulador RARS (Robot Auto Racing Simulator). Posteriormente se lanzó una versión en tres dimensiones nombrada como Open Racing Car Simulator (ORCS). Este simulador de coches no implementaba motores dentro de los coches, por lo que no fue hasta la implantación de estos cuando empezó a llamarse TORCS, nombre que mantiene en la actualidad y que le dota de más relevancia por su relación con la palabra torque o par en inglés.

Algunas de las características más reseñables son las siguientes:

- Orientado a jugadores, investigadores, ingenieros y profesores.
- Gran cantidad de circuitos, coches y oponentes.
- Modelo físico sofisticado.
- Soporta un gran número de dispositivos de entrada (volantes, joysticks, mandos de consolas etc.)
- Arquitectura modular
- Facil de modificar, añadir y crear contenido

Gracias a ese modularidad, a su simplicidad en el código base y a su estabilidad entre otras características, es ideal para investigaciones industriales y científicas, como lo abala el hecho de la cantidad de investigaciones de distinta índole utilizando este software.

En TORCS, los conductores tanto reales como NPCs (Non Playable Characters) son denominados robots. Estos robots son cargados de manera externa en el juego, de forma que

se pueda introducir nuevos módulos de robots para en este caso, implementar soluciones de inteligencia artificial de una manera estructurada.

Existe una gran comunidad alrededor de este software, como demuestra la continua aportación de los usuarios. Algunos ejemplos son los generadores de pistas online (Cardamone, Loiacono, & Lanzi, 2011), por los que presentaron un *framework* para la generación procedimental de pistas de una manera automática. Además, existen otras variantes a ser mencionadas. Una de ellas es Speed Dreams (Ande Gaetan et al, 2013), que modifica el código fuente de TORCS para implementar nuevas características, pistas y oponentes con la finalidad de proporcionar una mejor experiencia de usuario, además de proporcionar una mejora constante del realismo visual y de las físicas del juego. PyTorcs (Keith Curtis, 2013) es un *port* de TORCS para Python que reemplaza algunos módulos del software de código abierto, para ser compatible con el lenguaje de programación Python.

También existen numerosas competiciones en TORCS. Una de ellas es la llamada *Simulated Car Racing Championship* (Loiacono, Cardamone, & Lanzi, 2013), que está organizada por organismos de apoyo a la inteligencia artificial. Otra competición es el evento anual llamado TORCS *Endurance Word Championship* (Bernhard Wymann, 2013), que tiene una duración de alrededor de 6 meses. Debido a su larga duración, este evento está orientado tanto a personas aficionadas al juego, como a investigadores que tienen como finalidad continuar con la investigación o modificación de alguna parte del simulador.

Por todo lo mencionado anteriormente (estructura modular y competiciones anuales), existen más de 300 artículos académicos que hayan empleado TORCS como base, utilizando algoritmos de inteligencia artificial. En cualquier caso, TORCS ha encontrado otros usos más allá de la implementación en el simulador. Por ejemplo, ha sido usado como banco de pruebas para las unidades electrónicas de los automóviles (Drolia, Wang, Pant, & Mangharam, 2011). También se han realizado estudios de control de la atención y del estrés de un conductor al volante (Benoit et al., 2009) y para desarrollar un controlador económico de conducción autónoma para tractores (Bogoni & Pinho, 2012).

Aunque, el avance de TORCS se produce a menor ritmo que hace unos años, este simulador de coches continúa evolucionando, ofreciendo contenido tanto a aficionados del juego como a la comunidad de investigación académica. La arquitectura modular que presenta y el hecho de tener una licencia de código abierto, permite a este juego continuar desarrollándose a pesar de que los desarrolladores originales no sigan formando parte activa del proyecto.

2.2. Lógica Borrosa

En el trabajo Razonamiento con Imprecisión: Lógica borrosa (Manrique Gamo & Suárez de Figueroa Baonza, María del Carmen, 2015), esta se define como una disciplina contenida en la inteligencia artificial que permite representar el conocimiento acerca de un dominio y realizar procesos de inferencia o razonamiento, proporcionando un lenguaje formal de representación del conocimiento basado en la lógica y las matemáticas.

La lógica borrosa es capaz de dar solución al control de plantas de difícil modelado matemático aplicando conocimiento heurístico humano al control de procesos. Este conocimiento heurístico se puede relacionar con descripciones lingüísticas que no están cuantificadas y que representan un conocimiento borroso del problema a solventar (p.e una persona es alta o baja sin especificar donde se pone el límite o valor numérico de estas dos consideraciones). De esta manera, es capaz de crear un modelo lógico-matemático para crear sistemas que permiten el tratamiento de conocimiento impreciso o difuso.

Algunas de las principales ventajas de aplicar lógica borrosa son las siguientes:

- Es conceptualmente fácil de entender, ya que proporciona un modelo de caja blanca fácilmente interpretable y fácil de definir.
- Es flexible y tolerante a la imprecisión de datos.
- Permite modelar funciones no lineales a través de sus funciones de pertenencia.
- Se describe a partir del conocimiento e intuición de expertos. En este caso, un experto puede ser cualquier persona que tenga un conocimiento del funcionamiento cualitativo de proceso a controlar.

Para el diseño de un controlador borroso es necesario definir cada de sus elementos y procesos básicos que la envuelven, desde la captación de la información hasta que se produce una respuesta en consecuencia.

El elemento más importante es la **base de conocimiento** o *rule base*, que contiene la información de cómo controlar el sistema, en forma de un conjunto de reglas. Asocia unas entradas con unas salidas con asociaciones lingüísticas no cuantitativas.

También presenta unos **mecanismos de inferencia** o *inference mechanism*, que son los métodos con los que se evalúan las reglas de control que son relevantes en un momento especifico, y establece la salida.

Por último, existen dos elementos o procesos contrarios entre sí, uno la interfaz de **borrosificación** y otra la de **desborrosificación**. El proceso de borrosificación hace que las variables de entrada sean entendibles según nuestra base de conocimiento asociando etiquetas lingüísticas a valores numéricos. Una vez realizada la inferencia y de manera opuesta a la borrosificación, la interfaz de desborrosificación se coloca a la salida del sistema para dar una salida numérica a las etiquetas lingüísticas asociadas, de tal manera que puedan ser entendibles por la planta.

Tanto para el proceso de borrosificación como el de desborrosificación, se necesita una manera de convertir valores numéricos a etiquetas lingüísticas y viceversa. Esto es posible gracias a los conjuntos borrosos. Estos conjuntos borrosos no presenten límites abruptos ni claramente definidos. De esta manera, puede existir elementos con un cierto grado de pertenencia a una categoría o etiqueta concreta. El conjunto borroso está asociando a un valor lingüístico, definido por una palabra, adjetivo o etiqueta lingüística (p.e muy joven, joven, adulto, mayor, muy mayor). La certeza o certidumbre con la que una variable se le puede asignar el valor lingüístico, se indica por una función de pertenencia. Estas funciones de pertenencia se definirán con una forma bidimensional específica y que dependerá del problema a tratar. Se suelen definir funciones triangulares, trapezoidales, cuadradas o variantes menos lineales, como puede ser una función gausiana.

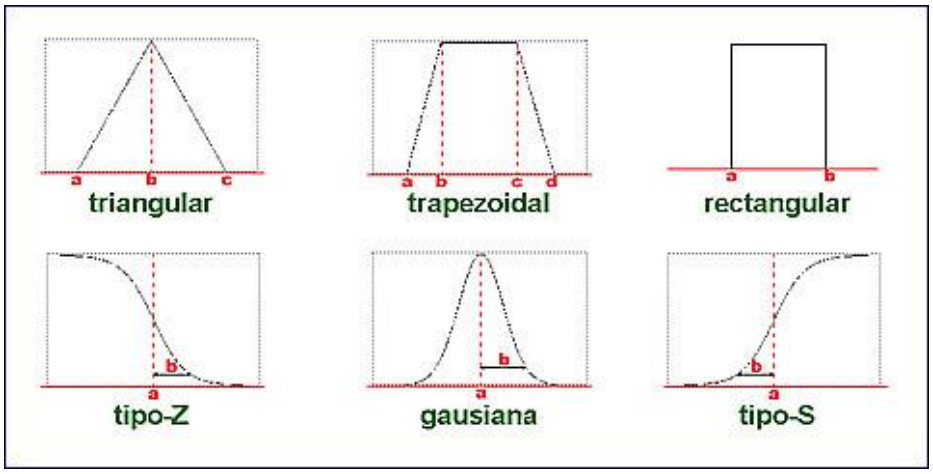

Figura 3. Distintos tipos de funciones de pertenencia que se asocian a las etiquetas lingüísticas.

Fuente: http://www.imse-cnm.csic.es/

Existen dos sistemas de inferencia principales para aplicar controladores borrosos. El primero de ellos es llamado **Mamdani**, y fue el primer método en crear un sistema de control sintetizando un conjunto de reglas de control obtenidas de la experiencia de operarios humanos. En un sistema Mamdani, la salida de cada regla es un conjunto borroso. El sistema

Mamdani es el más intuitivo y fácil de entender, además de encajar perfectamente para partir de reglas creadas desde la experiencia humana.

Por otro lado, tenemos el sistema **Sugeno** de lógica borrosa para la creación de controladores. También referida como Takagi-Sugeno-Kang, usa funciones delta a la salida o de valor constante, con independencia de los valores de entrada. El proceso de desborrosificación de un sistema Sugeno es computacionalmente más eficiente que un sistema Mamdani, ya que para la desborrosificación, usa una suma ponderada de la salida, en vez de realizar el cálculo de los centroides de áreas bidimensionales.

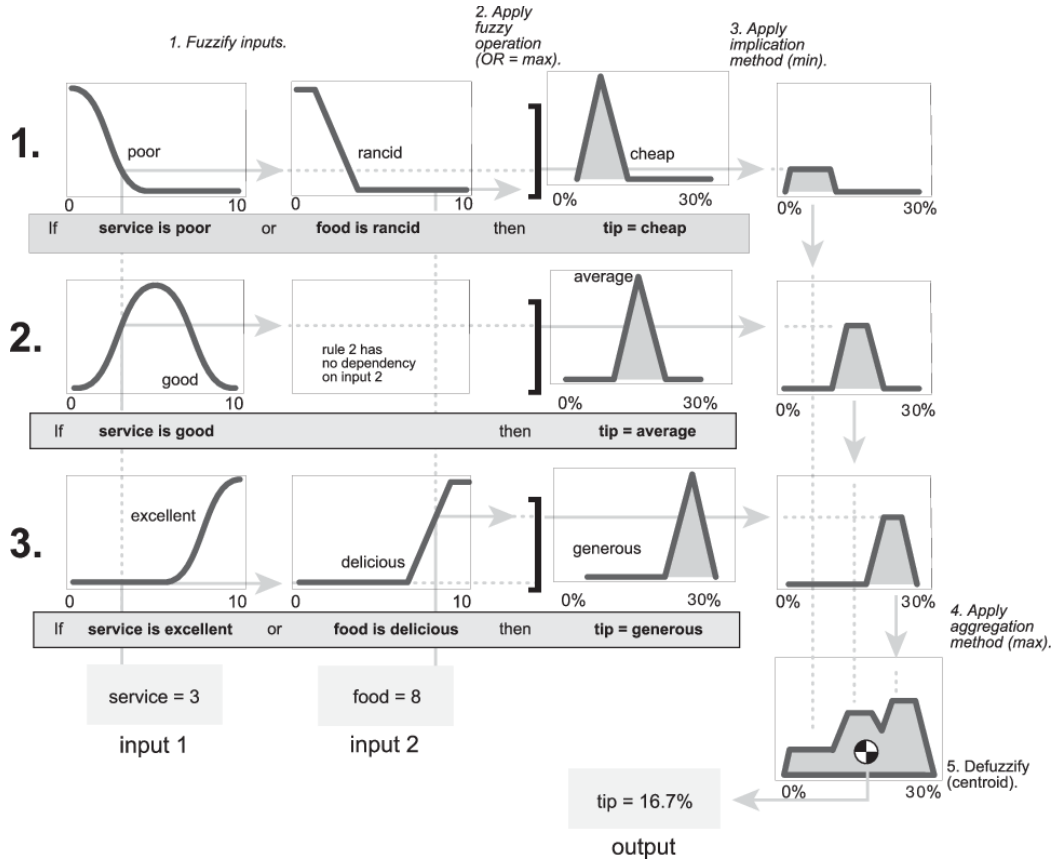

Figura 4. Proceso de la aplicación de todos los pasos de un proceso de inferencia en un sistema Mamdami.

Fuente: es.mathworks.com

En la figura 4 se puede apreciar los diferentes pasos que componen el proceso de inferencia de un sistema borroso, desde la definición de reglas con las distintas funciones de pertenencia hasta la desborrosificación (en este caso para un sistema Mamdami), pasando por el proceso de agregación de las salidas.

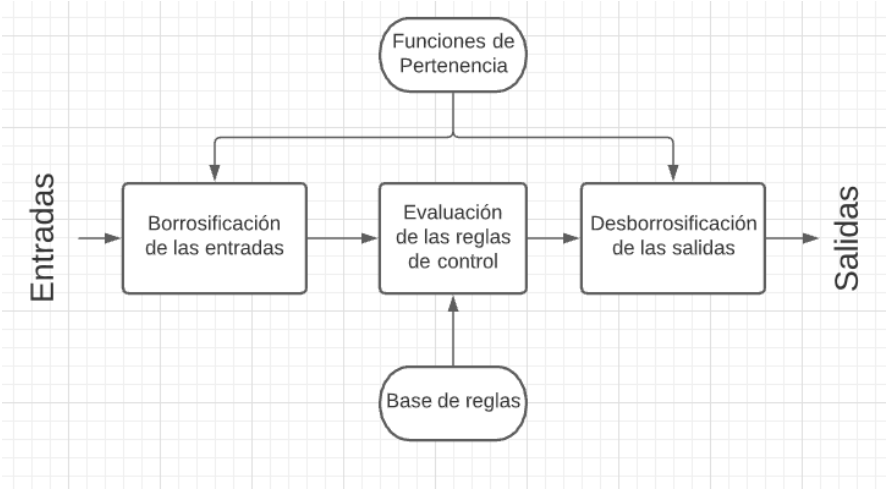

Figura 5. Estructura de un controlador borroso

La lógica borrosa ha sido utilizada de manera prolífica para abordar el problema de la conducción autónoma. Esta técnica ha sido implementada de manera recurrente y en un principio con mayor frecuencia que otras técnicas de inteligencia artificial debido a que no es necesario tener una capacidad computacional elevada si se compara con el coste de la implementación de ANN y CNN y respectivos procesos de entrenamiento.

Por ejemplo, se ha examinado el desempeño de un sistema borroso basado en reglas de decisiones de alto nivel en un controlador, transformando la información de entrada proporcionada por los sensores (Fujii, Nakashima, & Ishibuchi, 2008). También se ha propuesto el diseño de un controlador borroso para calcular la posición deseada del coche, aunque no se usó para provocar el movimiento del volante (Onieva, Pelta, Alonso, Milanés, & Pérez, 2009). Se ha propuesto además para la utilización de lógica borrosa, la implementación de dos controladores borrosos que funcionan de manera simultánea pero independiente, unos para el control de la velocidad, y otro para controlar la dirección del volante (Salem, Mora, Merelo, & García-Sánchez, 2017). Cogiendo de referencia esta última solución, se realizará el desarrollo del control borroso del coche utilizando este tipo de implementación que se puede considerar más clásica o antigua dentro del ámbito de la inteligencia artificial. Con la información aportada por investigaciones anteriores, se hará un ajuste más preciso y acorde a los sensores de entrada de los que se dispone. Se han propuesto adicionalmente otros sistemas híbridos que utilizan técnicas de machine learning con lógica borrosa (Ho & Garibaldi, 2008), o lógica borrosa con algoritmos genéticos (Pérez, Recio, Sáez, & Isasi, 2009) para la optimización de hiperparámetros[1].

[1] Los algoritmos genéticos son aquellos que se basan en las leyes de la selección natural o que son inspirados por el comportamiento de ciertas especies, donde se sigue un esquema de adaptación basado en la naturaleza (Moliner & Tanda, 2016).

2.3. Redes Neuronales Artificiales

Las redes neuronales artificiales tienen la finalidad de aproximar cualquier función matemática. Son especialmente útiles cuando el proceso que relaciona las entradas con la salida es complejo o presenta ruido (perturbaciones de alta frecuencia que interfieren sobre una tendencia estable), ya que es capaz de computar grandes cantidades de variables de entrada para modelar una salida.

Las redes neuronales artificiales, como su nombre indica, están inspiradas (aunque de una manera ligera) por el cerebro biológico y su sistema nervioso. Este cerebro biológico es muy diferente en términos de su estructura y la manera en la que procesa información. La característica más distintiva de un cerebro biológico es la capacidad de “aprender” y “adaptar”, mientras que un ordenador convencional no tiene tales habilidades, ya que completan tareas específicas basadas en unas instrucciones definidas previamente. Las redes neuronales artificiales rompen con esa concepción de la computación clásica ya que, al igual que los cerebros biológicos, son capaces de “aprender” de los errores a través de un ajuste de sus parámetros internos para adaptarse al problema. De esta manera, se puede decir que las redes neuronales artificiales encuentran la relación entre un conjunto de señales de entrada y una señal o conjunto de señales de salida utilizando un modelo que simula la forma en que el cerebro responde a los estímulos.

Las ANN emergieron de un popular algoritmo de aprendizaje automático llamado perceptrón (Rosenblatt, 1958). El perceptrón fue desarrollado en los años 50 y 60 por el científico Frank Rosenblatt, inspirado anteriormente por el trabajo de Warren McColloch y Walter Pitts en 1943 referente al funcionamiento del sistema nervioso desde una perspectiva computacional. Tal y como se puede observar en la siguiente fórmula, el perceptrón es un clasificador lineal que consta de unas entradas x(n), ponderadas por unos pesos w(n), siendo n el número de entradas a la neuronal. Más tarde, se aplica la función f() al sumatorio de las multiplicaciones de las variables de entrada con los pesos, obteniendo de esta manera la salida y(x).

$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

Combinando varias capas de perceptrones se crea lo que es conocido como perceptrón multicapa o redes neuronales artificiales (ver figura 6). A diferencia de un perceptrón unitario, el perceptrón multicapa es capaz de modelar funciones no lineales. Debido a esta

característica, han sido utilizadas ampliamente para tareas de regresión y clasificación en aprendizaje supervisado.

Existen tres características de una red neuronal artificial que hacen que existan infinidad de ellas y que puedan ajustarse mejor al problema en el que se apliquen. La primera de ellas es la topología o arquitectura de la red, que define tanto el número de neuronas como el número de capas de la red. El número de capas y de neuronas óptimo, pueden venir dados por la complejidad de la red ya que, a mayor número de estos parámetros, la red será capaz de aprender patrones más complejos. La arquitectura de la red se debe ajustar a la complejidad del problema, ya que una arquitectura muy compleja puede producir un sobreajuste a los datos de entrada, mientras que una arquitectura simple puede producir lo que se conoce como *underfitting*, es decir, la red no es capaz de encontrar un patrón común para los valores de entrada propuestos. El mínimo de capas necesario será el de la capa de entrada con tantos números de neuronas como variables de entradas tenga el problema y una de salida. Las capas que se sitúan entre la capa de entrada y la de salida son las llamadas capas ocultas y hacen que aumente la profundidad de la red.

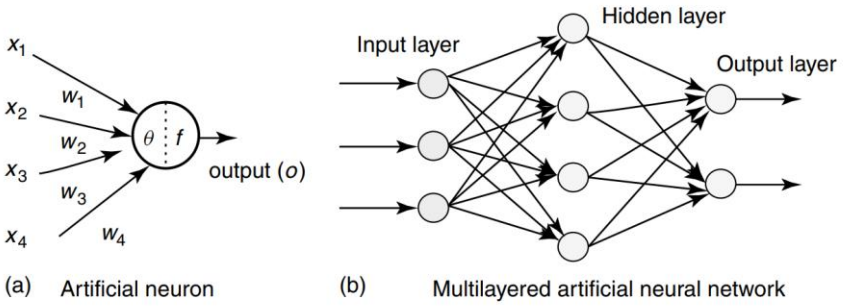

Figura 6. Arquitectura de un perceptrón y una red neuronal multicapa.

Fuente: Abraham, 2005

La función de activación o *transfer functions* define cómo se computan los valores de entrada a un nodo y de qué manera se convierten en la salida. La elección de la función de activación tiene un gran impacto en la capacidad y el rendimiento de la red neuronal. Todas las capas ocultas utilizan típicamente las mismas funciones de activación en un mismo modelo, mientras que en la capa de salida se suele utilizar una distinta pero ligada al tipo de predicción requerida (regresión, clasificación, etc.) Algunas de las funciones de activación típicas para las capas ocultas son la ReLU (*Rectified Linear Activation*), la sigmoidea o la tangente hiperbólica tal y como están representadas en la figura 7 mientras que a la salida se suelen utilizar funciones de activación lineales o Softmax.

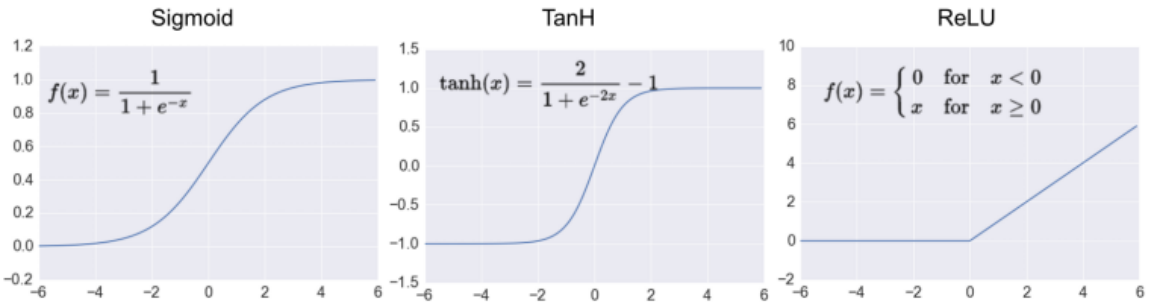

Figura 7. Tipos de funciones de activación en las capas ocultas más utilizadas.

Fuente: www.KDnuggets.com

Otro parámetro determinante en el funcionamiento de las ANN, es la del algoritmo de entrenamiento. Para que la red pueda hacer una predicción correcta a partir de unos datos de entrada, se necesita un proceso de entrenamiento (que puede ser tanto supervisado como no supervisado con aplicación en la detección de anomalías, aunque este apartado estará centrado en la primera opción) en el que la red, a través de unos casos ya conocidos de relación entre entrada y salida, sea capaz de ajustar sus pesos de tal manera que busque un patrón común entre las entradas y las salidas propuestas. Por norma general, si se dispone de un número mayor de datos de entrenamiento, la red será capaz de ajustar sus pesos de manera más precisa, mejorando la exactitud del modelo. Otros factores que pueden ser determinante es el tratamiento previo de las imágenes de entrenamiento para corregir problemas como el desbalanceo entre clases o la calidad de los datos con los que se trabaja.

La práctica más común es que el ajuste de los pesos se haga a través del algoritmo de *backpropagation*. Este algoritmo utiliza la regla de la cadena para su funcionamiento, de tal manera que cada ciclo de entrenamiento tiene una fase llamada *fordward pass*, seguida de un *backward pass* donde ajusta el modelo a través de los pesos existentes. A través del algoritmo del descenso del gradiente, la red es capaz calcular la dirección en la que se producirá una mayor reducción del error de la función de coste (función que pone en relación la salida real y el valor actual) hasta poder alcanzar un mínimo local o absoluto. Ajustando el parámetro del *learning rate*, se controla el tamaño del paso que se toma en cada ciclo o *epoch* del entrenamiento. Un *learning rate* pequeño puede hacer que el tiempo de convergencia hasta un mínimo local pueda ser elevado, mientras que un *learning rate* grande puede producir la convergencia del modelo, alejándonos de los mínimos locales y absolutos. Una alternativa a estos problemas es la del uso de un *leaning rate* adaptativo que, tal y como se muestra en la figura 8, a medida que transcurren un número determinado de *epochs*, va reduciendo su valor, mejorando así las prestaciones del modelo entrenado.

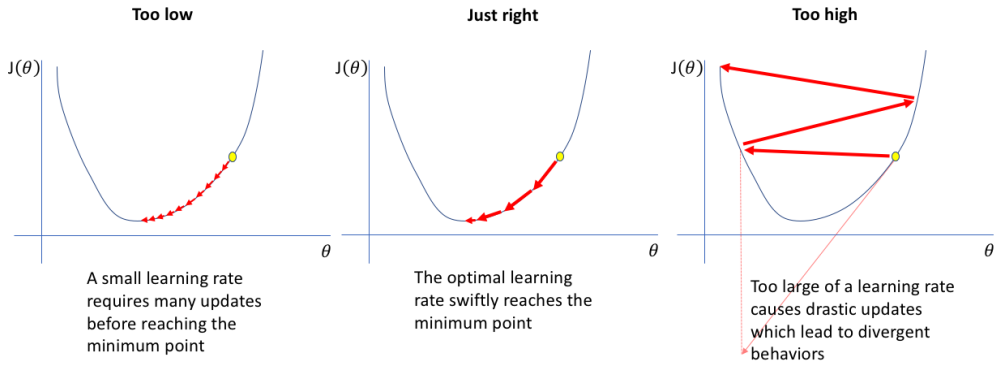

Figura 8. Algoritmo del descenso del gradiente con diferentes valores de Learning Rate.

Fuente: https://deepai.org/

Este tipo de algoritmo de inteligencia artificial ha sido implementado en numerosas ocasiones para abordar el tema de la conducción autónoma. Para realizar la fase de entrenamiento del automóvil, se necesitan datos que le permitirán a la red neuronal ajustar sus pesos para tener un mejor desempeño en este problema. Este algoritmo recibe como entrada datos proporcionados por el propio simulador, tanto de datos referentes a los parámetros internos del vehículo (velocidad, inercia, aceleración) como en relación con la pista (ángulo, posición respecto al centro de la pista). Gracias a esos datos de entrada y a la información de salida (giro del volante, acción de acelerar, acción de frenar), son capaces de aprender esos conocimientos y ponerlos en práctica en circuitos donde no han sido entrenados. Para la recolección de estos datos, se ha hecho uso de los robots que están previamente implementados en el simulador TORCS por defecto (Athanasiadis, Galanopoulos, & Tefas, 2012; Albelihi & Vrajitoru, 2015), mientras que en (Munoz, Gutierrez, & Sanchis, 2009), los datos se extraen de un automóvil conducido por una persona, ya que su propósito de esta investigación es el de diseñar un robot que sea capaz de imitar en cualquier pista la conducción humana.

Un caso de combinación de distintos algoritmos de inteligencia artificial se encuentra en las llamadas redes NEAT (*NeuroEvolution of Augmenting Topologies*). Este tipo de redes puede optimizar y evolucionar la estructura entera de una red neuronal (Stanley & Miikkulainen, 2002), para producir unos resultados que minimicen la función de coste. En el campo de la conducción autónoma, han sido implementados junto a las redes neuronales artificiales en (Muñoz, Gutiérrez, & Sanchís, 2009).

2.4. Redes Neuronales Convolucionales

Las redes neuronales convolucionales o CNN (del inglés *Convolutional Neural Networks*), es un algoritmo de *Deep Learning* (DL) que constituye el estado del arte para muchas de las

tareas de visión artificial (VA) por su capacidad de extraer y aprender patrones espaciales (relaciones entre los píxeles que definen una imagen).

La visión artificial o visión por computador consiste en la capacidad que tiene la computadora de realizar tareas similares a la visión humana, principalmente centrándose en la obtención/extracción de información numérica y simbólica a partir de imágenes estáticas y/o de secuencias (vídeos). Esta extracción de información a partir de imágenes, suele depender de que se completen diferentes tareas, como la adquisición, el procesado, el análisis y la compresión de la imagen (Forsyth, 2011; Szeliski, 2010). La adquisición de la imagen consiste en la obtención de datos numéricos a partir de una señal de entrada recibida desde un dispositivo capturador de imágenes (es decir, la digitalización de una imagen de entrada). El procesado de la imagen se centra en realizar transformaciones sobre una imagen para mejorarla y facilitar diferentes operaciones sobre la misma (por ejemplo, realzarla). La salida que se produce tras el procesado de la imagen suele ser la entrada para otra tarea específica posterior. El análisis de la imagen se emplea para la extracción de características. Finalmente, la compresión se emplea para la reducción de tamaño y/o complejidad de la información encapsulada en imagen.

Gracias a sus grandes prestaciones dentro del campo de la visión artificial, las CNN se han utilizado para numerosas tareas dentro de esta: la clasificación, la localización, la detección, la identificación, la segmentación y el seguimiento de objetos. La clasificación categoriza una imagen en función de un conjunto de etiquetas predefinidas. La localización ubica un objeto que se sabe dentro de la imagen. La detección ubica un objeto en una imagen si este está presente. La segmentación particiona la imagen en segmentos (o áreas) que delimitan objetos. Por último, el seguimiento, se encarga de ubicar y mantener un único identificador por cada objeto mostrado (Forsyth, 2011; Hassaballah, & Awad, 2020; Szeliski, 2010).

La arquitectura básica de una CNN de clasificación, como la necesaria en el problema de conducción autónoma, puede ser diseccionada en cuatro áreas clave:

- Como se puede encontrar en otras formas de redes neuronales, la primera capa contiene los valores de entrada, en este caso, los pixeles de la imagen.

- La capa principal dentro de estas redes y en la que se centra su funcionamiento, es la de convolución. Como su propio nombre indica, realiza una operación de convolución con la imagen o capa de entrada a través de un filtro o *kernel* definido por el usuario que recorre todos los píxeles de entrada. Al igual que en las redes neuronales artificiales, en estas capas también se utilizan funciones de activación que producen un mayor ajuste de los pesos de la red, siendo la más común la función de activación ReLU.

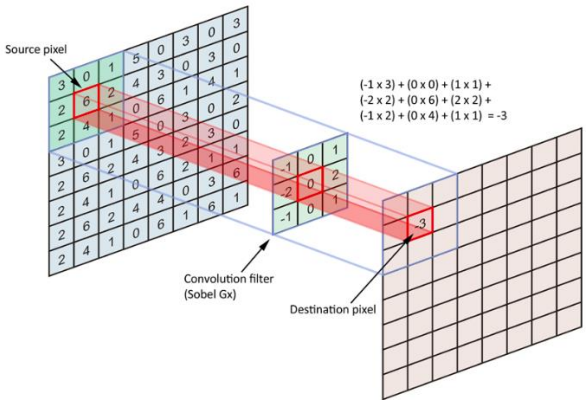

Figura 9. Ejemplo de la operación de convolución.

Fuente: www.medium.com

- La capa de *pooling* realiza una reducción de dimensionalidad espacial de la entrada dada, aminorando de esta manera los parámetros de esta e intentando minimizar la pérdida de información.
- La capa *fully connected* es una red neuronal artificial donde todos los nodos están conectados entre sí. Tiene la función de vectorizar su entrada (eliminando en este caso la información espacial entre sus valores) para producir la salida de toda la red.

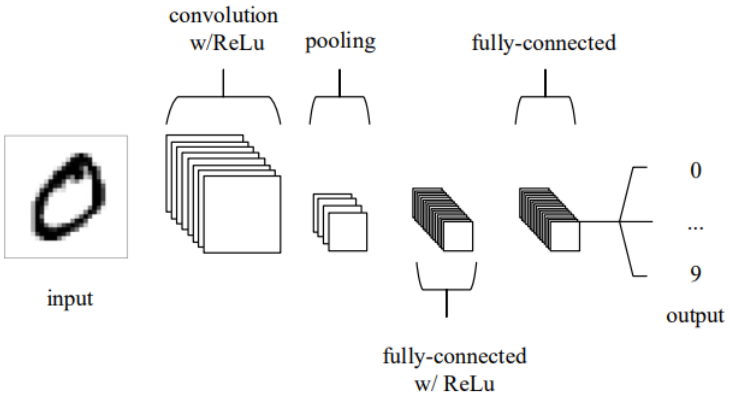

Figura 10. Estructura básica de una red neuronal convolucional de clasificación.

Fuente: (O'Shea & Nash, 2015)

Al ser las redes neuronales convolucionales como las artificiales un subtipo de red neuronal, presentan similitudes en sus algoritmos, como puede ser que ambas utilizan el algoritmo de *backpropagation* y del descenso del gradiente para minimizar una función de coste. De hecho,

como se ha mostrado anteriormente, las CNN presentan una red neuronal artificial en el último paso para determinar una clasificación final. La diferencia más notoria entre ambas, es que las CNN, mediante su operación de convolución, son capaces de guardar relaciones espaciales entre la información de entrada, lo que la hace idónea para trabajar con imágenes. Por otro lado, las redes neuronales artificiales, al no presentar esta característica, hacen que sean más dados a trabajar con otro tipo de datos de entrada, como pueden ser variables independientes.

Aunque las CNN tengan la ventaja de poder conservar la relación espacial de sus valores de entrada, estas suelen ser computacionalmente más costosas, necesitando una capacidad de cómputo muy elevada tanto en la fase de entrenamiento como en la de inferencia. Para procesar una imagen mediana de un alto y ancho de 200 píxeles en color, que suponen la existencia de tres canales del mismo tamaño, se deben procesar 120.000 parámetros de entrada para una sola imagen.

La eficacia de las CNN en las tareas de VA y la mayor capacidad de cómputo han permitido que aparezcan un número importante de variaciones sobre esta arquitectura. Uno de los factores que ha permitido la evolución de este tipo de redes neuronales convolucionales es la promoción de competiciones en las que, dados un conjunto de datos, se pretende obtener la mayor precisión posible en las tareas de VA.

Entre estas competiciones destaca ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015) organizada desde 2010 por el proyecto ImageNet. En ella se proporcionaba un conjunto de datos de millones de imágenes etiquetadas manualmente y divididas en 1000 categorías. Así, permitió comparar diferentes propuestas para la clasificación de imágenes a gran escala. En 2012 la arquitectura CNN llamada AlexNet (Krizhevsky, Sutskever, & Hinton, 2012) mejoró la precisión de los modelos de clasificación predecesores. Otro hito lo marca IncepcionV3 (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2016), que superó la precisión humana en un 1,5 %. En el año 2017 con SENet (Hu, Shen, & Sun, 2018) con una tasa de error de 2,25 %, se dio por terminada esta competición, puesto que se entendía que el problema de clasificación estaba resuelto. La figura 11 muestra los resultados de las propuestas vencedoras para cada uno de los años.

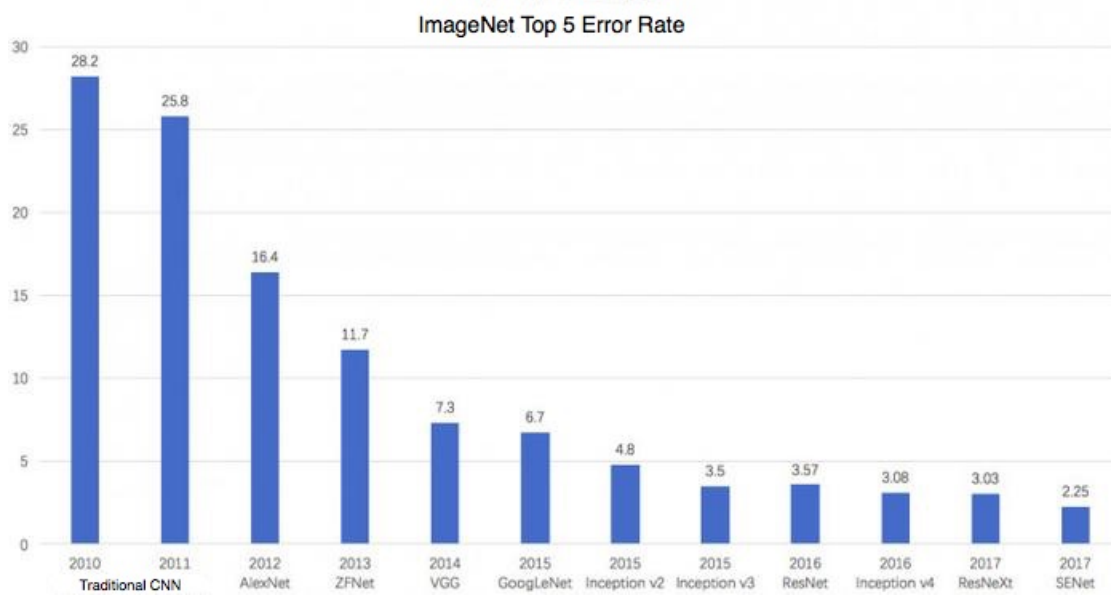

Figura 11. Resultados de las propuestas vencedoras para cada uno de los años para la competición ILSVRC.

Fuente: Google images

Otras arquitecturas a lo largo del tiempo han mostrado variaciones sobre las redes presentadas anteriormente tratando de resolver problemas de clasificación más específicos. Por ejemplo, Restnet (He, Zhang, Ren, & Sun, 2016), VGG (Simonyan & Zisserman, 2014) o Google leNet (Szegedy et al., 2015). Cabe destacar EfficientNet (Tan & Le, 2019), una CNN ligera (arquitectura menos compleja) que provee de una precisión muy competitiva.

Referente al estado del arte de las CNN aplicado a resolver el problema de la conducción autónoma, se han propuesto numerosos casos de cámaras, tanto frontales y traseras de asistencia al conductor. Algunos ejemplos de esta aplicación puede ser la ayuda al aparcamiento (Gamal, Imran, Roth, & Wahrburg, 2020) o la observación del conductor para detectar patrones de distracción de la carretera (Tran, Do, Lu, & Sheng, 2020). Adicionalmente, se han desarrollado en menor medida, algunos sistemas que implementan esta tecnología para la propia conducción del automóvil para el simulador de carreras TORCS. El mayor avance en el uso de CNN para la conducción autónoma implementado en el simulador TORCS viene dado por la universidad de *Princeton* con el proyecto nombrado *Deepdriving* (Chen, Seff, Kornhauser, & Xiao, 2015). En este proyecto, se utilizan pistas modificadas con líneas divisorias que ayudan al automóvil a predecir tanto la dirección de la pista con mayor exactitud. Se propone una solución tanto para la conducción autónoma, como para la detección de otros automóviles para evitar posibles colisiones.

La menor aplicación o investigación de las CNN frente a otros algoritmos como pueden ser las ANN o la lógica borrosa dentro de este campo, puede estar dada por la gran capacidad de cómputo necesaria para llevar a cabo la tarea de entrenar una red de este tipo, desde la recolección de datos (imágenes) y su almacenamiento, hasta el proceso de entrenamiento con esos datos. En cualquier caso, el avance que se está dando dentro del mundo tecnológico y en concreto en las tarjetas gráficas o GPU (*Graphics Processing Unit*), necesarias para todo el proceso con redes neuronales, está permitiendo una mayor accesibilidad de estos sistemas y, por lo tanto, se espera un aumento del número de investigaciones sobre este tema.

2.5. Otros algoritmos de inteligencia artificial

Existen numerosas propuestas adicionales dentro del marco de la inteligencia artificial para resolver el problema de la conducción autónoma. Muchos de ellos combinan algoritmos vistos anteriormente con otros nuevos, como puede ser (Pérez, Recio, Sáez, & Isasi, 2009), que utiliza tanto lógica borrosa como algoritmos evolutivos para la optimización de los hiperparámetros o las mencionadas anteriormente redes NEAT, que tienen el mismo propósito de optimización.

Otros algoritmos también utilizados de manera continuada e implementados en simuladores de conducción, son las redes neuronales recurrentes y el aprendizaje por refuerzo. Las redes neuronales recurrentes o RNN del inglés *Recurrent Neural Network* (Rumelhart, Hinton, & Williams, 1986), son un tipo de red neuronal que almacena valores pasados para hacer inferencias futuras, lo que se puede considerar una función de memoria. Para el caso de la condición autónoma, puede ser especialmente útil el tener información de datos pasados, como el cambio de velocidad para hacer predicciones de manera más acertadas. Un ejemplo de uso de este algoritmo aplicado en TORCS es el de (McNeill, 2018).

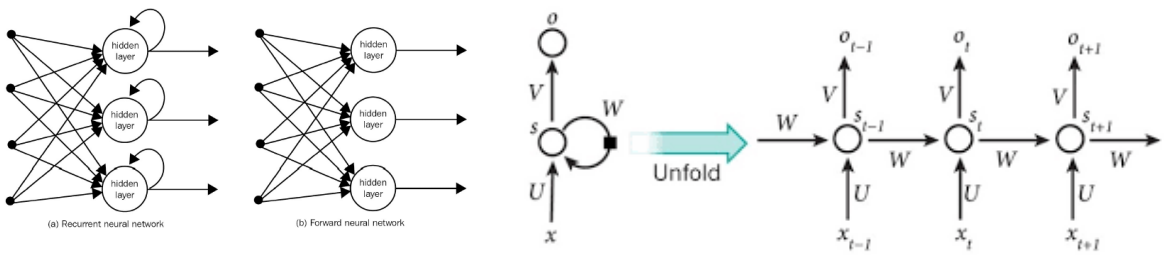

Figura 12. Comparación entre las capas ocultas de una red neuronal recurrente y una red neuronal artificial y su posterior despliegue.

Fuente: (Barrero, 2018)

El aprendizaje por refuerzo o *Reinforcement Learning* (Kaelbling, Littman, & Moore, 1996) es otro algoritmo de inteligencia artificial que intenta minimizar una función de coste a través acciones producidas por un agente (el coche) que interactúa con un entorno (la pista de carreras). Según las acciones que toma este agente, se producen recompensas si son correctas o castigos (salirse de la pista, producir una colisión) si no son las adecuadas. De esta manera, en (Karavolos, 2013), se presenta un castigo de dos puntos sobre la función de *fitness* si el coche se queda atascado, de un punto si el coche se sale de la pista y si se mantiene en la carretera, una recompensa que depende de la posición y ángulo dentro de la pista de un máximo de un punto.

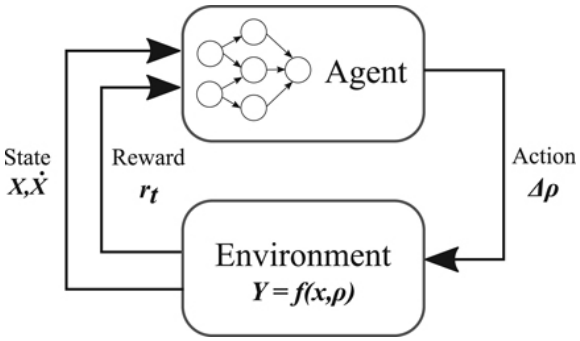

Figura 13. Estructura básica de un algoritmo de aprendizaje por refuerzo.

Fuente: (Vashishtha & Verma, 2020)

Aunque estas últimas propuestas son igual de válidas para hacer una comparativa sobre el problema, la elección de utilizar los tres primeros algoritmos viene motivada por el hecho de cubrir una gran variedad de algoritmos que existen hoy en día dentro de la inteligencia artificial. La lógica borrosa es una técnica más clásica que, aunque necesita menos carga computacional, necesita el conocimiento de un experto para definir las reglas de actuación. Por otro lado, tanto las ANN como las CNN son redes neuronales, pero con grandes diferencias entre sí. Mientras que las ANN computan parámetros de entrada que son datos numéricos que no presentan relación espacial entre sí, la aplicación de las CNN sí tiene esta característica que hace que se incluya dentro de la categoría de visión artificial.

3. Objetivos y metodología de trabajo

Este tercer capítulo sirve como puente entre el estudio del dominio de la conducción autónoma y la contribución que se va a realizar. Se compone de objetivo general, objetivos específicos y por último, de la metodología de trabajo, donde se definirán una serie de pasos que ayudarán a la resolución de los objetivos propuestos anteriormente, orientado en todo momento a la resolución del problema de la conducción autónoma.

3.1. Objetivo general

El objetivo principal del trabajo de fin de máster es mostrar una comparativa de manera extensa y detallada de cuál es la técnica o algoritmo de inteligencia artificial que más se adecúa al problema de la conducción autónoma. Los algoritmos desarrollados se probarán en el simulador de coches de código abierto TORCS por su estructura modular y recursos de investigación disponibles. Para concluir, también plantearemos una manera en la que se puedan trasladar los conocimientos adquiridos y los resultados obtenidos en el simulador, a la conducción de un automóvil real.

3.2. Objetivos específicos

Para conseguir el objetivo principal de la comparación y determinación de las ventajas y desventajas de cada una de las técnicas de inteligencia artificial aplicadas, definimos una serie de objetivos parciales o específicos:

- Desarrollar en profundidad cada una de las técnicas o algoritmos de inteligencia artificial a implementar elegidos, proporcionando la configuración óptima de ese algoritmo ajustándose al problema propuesto.
- Evaluar los algoritmos aplicados en 3 circuitos diferentes, manteniendo las mismas condiciones para cada una de las pruebas hechas en los circuitos.

3.3. Metodología del trabajo

De cara a alcanzar los objetivos específicos y con ellos, el objetivo general, definimos una metodología de trabajo, en la que se describen paso a paso las acciones a seguir, y el orden en el tiempo de estas.

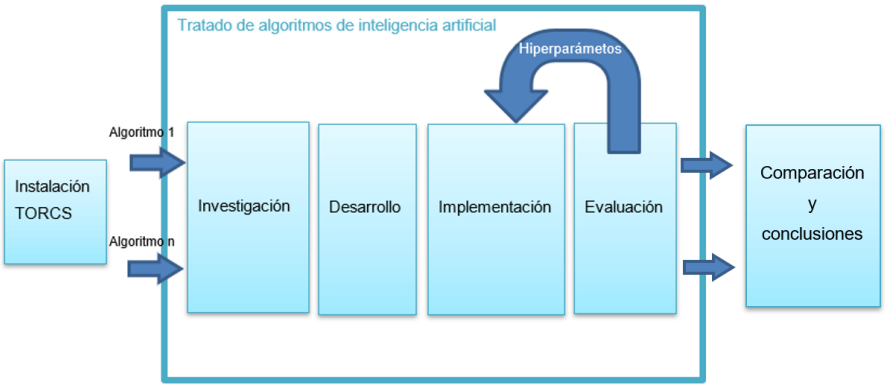

Figura 14. Metodología de trabajo

En la figura 14, se muestra un diagrama que pretende ser una guía visual de los pasos a seguir. Se empezará con la instalación del simulador de coches TORCS en el sistema operativo Linux dentro de una máquina virtual. Esto es debido a que la mayoría de las investigaciones sobre este software están en este sistema operativo, además de que también existe la mayor comunidad de resolución de problemas derivados de su instalación y modificación. Una vez instalado el software y todas sus dependencias necesarias, se escogerá una técnica o algoritmo de inteligencia artificial para ver si es apta su implementación. Si determinamos que, en efecto, ese algoritmo puede ser útil para el problema de la conducción autónoma, se plantean una serie de pasos a alto nivel que van a compartir todos los algoritmos que se vayan a probar en el simulador.

El primer paso describe la **investigación** sobre el algoritmo, donde determinaremos el funcionamiento de este y los recursos necesarios que se pueden buscar en la web (por ejemplo, un *dataset* público con imágenes para la implantación de redes neuronales convolucionales). Una vez completada la fase de investigación, estableceremos una fase de **desarrollo** de ese algoritmo. Para esta fase de desarrollo se buscará la manera de hacer uso de los recursos disponibles (uso de distintos lenguajes de programación, de distintos *frameworks* de aprendizaje automático o de software específico como Matlab). Hecho un modelo primigenio tras una fase de **implementación** dentro del simulador, se evaluará su comportamiento e intentaremos ajustar los hiperparámetros de la red dependiendo del algoritmo utilizado (modificación del *dataset* en la redes neuronales o cambio de las funciones de pertenencia en el controlador borroso) de manera que busquemos la mejor solución teniendo en cuenta la persecución de los objetivos del trabajo. Tras haber repetido el proceso de ajuste de hiperparámetros hasta que se haya encontrado una solución óptima, se hará una comparación de todos los resultados de todos los algoritmos en una tabla, de manera que se pueda extraer conclusiones, si hay algún algoritmo que sea claramente mejor que otro, teniendo en cuenta los criterios de evaluación o si, por el contrario, cada algoritmo tiene ventajas y desventajas respecto al resto.

4. Planteamiento de la comparativa

En este capítulo se identifica, con todo lo expuesto anteriormente, en qué va a consistir la comparativa de los controladores de inteligencia artificial elegidos y desarrollados en los capítulos posteriores. Se va a determinar de manera cuantitativa cuáles son los parámetros a medir en cada uno de los controladores para asignarle una nota global al desempeño de dicho controlador.

4.1 Coche de pruebas

Es importante que se utilice un mismo coche, que será el mismo en todas las pruebas. De esta manera, se elimina la importancia que pueda tener la diferencia de los distintos vehículos en cuanto a motor, adherencia a la pista o control que pueda alterar la extracción de conclusiones de una manera clara. El coche que se va a utilizar para estas pruebas, es el Alfa Romeo 155 DTM. Se trata de un coche de carreras que, aunque tiene gran potencia, presenta tracción a las 4 ruedas y una gran manejabilidad. Esto ayudará en la labor de mantener dentro de la pista el vehículo con los diferentes controladores desarrollados posteriormente.

Figura 15. Alfa Rome DTM-155

Fuente: Google imágenes

4.2 Circuitos de pruebas

Se definen también 3 circuitos de donde se harán las pruebas de los controladores. Estos circuitos deben ser diferentes entre sí para poder determinar diferencias en los controladores para distintos tipos de circuitos. Se establecen los circuitos de *E-Track5*, *CG track 2* y *CG-Speedway Number 1* como los circuitos donde se realizará la comparativa de los distintos controladores aplicados.

		Nombre pista		
		E-Speedway	**CG track 2**	**CG-Speedway Number 1**
				
Forma	**Tipo de pista**	Oval	Road	Road
	Longitud	4130.84 m	3185.83 m	2057.56 m
	Anchura	30 m	15 m	15 m

Tabla 1. Circuitos de pruebas de los controladores

4.3 Variables de análisis

Se establecen las variables a tener en cuenta para determinar el funcionamiento de cada uno de los controladores. La variable más importante para controlar será la del tiempo por vuelta, ya que este aspecto proporciona una idea acertada de la efectividad del algoritmo aplicado. Se hará una prueba de 3 vueltas con cada controlador y se obtendrá el resultado de la media por vuelta a partir de la segunda (la primera vuelta el automóvil empieza parado en el modo contrarreloj, por lo que la media se aplica desde la vuelta 2 hasta la 5).

Se hará uso de otras dos clasificaciones. Primero, se establecerá si el coche ha tenido alguna salida de pista y se tendrá en cuenta en la valoración final del controlador (en cualquier caso, el objetivo en el desarrollo de los controladores debe ser que se mantengan en pista sin problemas).

El tipo de conducción que haga el algoritmo también es un factor determinante para asignar una puntuación final, ya que, como se quiere implementar la simulación en la implementación en un vehículo real, no es lo mismo que el modelo creado conduzca haciendo cambios de dirección bruscos o de manera suave. Se observarán dos gráficas de error angular y de posición según el centro de la pista y dependiendo del resultado observado se ordenarán los distintos controladores a los que se le asignarán tres, dos y un punto, siendo el puntaje más alto para el controlador que presente una mejor conducción.

Dejando a un lado las variables numéricas y una vez realizados la implantación y desarrollo de los algoritmos, también se expondrán las ventajas y desventajas de cada uno de ellos. De

esta manera, se determinará en qué contexto de aplicación es mejor uno que otro, teniendo en cuenta aspectos como la facilidad de desarrollo e implantación, el coste computacional o el coste material y económico asociado.

Con todo lo dicho anteriormente, se determinará cuál es el algoritmo de inteligencia artificial elegidos que presenta mejores prestaciones con las condiciones propuestas.

4.4. Instalación de TORCS

La instalación del software se hará en Linux, debido a que existe una mayor comunidad de jugadores e investigadores que proporcionan información tanto para la instalación para la para modificación en este sistema operativo.

Se instalan distintas librerías necesarias, como pueden ser el acelerador HW OpenGL, GLOT 3.7, OpenAL. etc. Dependiendo de la distribución de Linux instalada, estas dependencias pueden cambiar. En este trabajo de fin de máster se utiliza Zorin OS. Se trata de una distribución de Linux que parte de Ubuntu como núcleo, mejorando aspectos como la apariencia, la optimización y la seguridad de esta distribución, pero conservando gran parte de su núcleo.

```
sudo apt install g++
sudo apt install build-essential
sudo apt install freeglut3-dev
sudo apt install libplib-dev
sudo apt install libopenal-dev
sudo apt install libalut-dev
sudo apt install libvorbis-dev
sudo apt install libxi-dev
sudo apt install libxmu-dev
sudo apt install libxrender-dev
sudo apt install libxrandr-dev
sudo apt install libpng-dev
```

Figura 16. Instalación de librerías necesarias

Posteriormente, se descarga el archivo “all in one” de TORCS, que está disponible para su descargar en la página oficial de TORCS. Una vez descargado este archivo, se descomprime, se compila y se procede posteriormente a su instalación con estas líneas de código:

```
tar xvjf torcs-1.3.7.tar.bz2
cd torcs-1.3.7
./configure
make
sudo make install
sudo make datainstall
```

Figura 17. Descompresión e instalación del archivo "all in one" de TORCS

La ejecución del archivo *configure* se asegura de que se hayan descargado todas las dependencias necesarias para su correcto funcionamiento. Con el comando *make* se instalan el resto de las dependencias.

4.5. Generación de robots en TORCS

Antes de implementar una solución de un controlador aplicando inteligencia artificial, se debe crear el esqueleto del robot. En este proceso, se copian las características físicas de uno de los robots existentes previamente en el simulador. Existen más de 50 coches que tienen características físicas muy variadas, desde un coche de *rallies* a un fórmula 1.

Figura 18. Ejemplo de coches en el simulador TORCS

Fuente: sourceforge.net/projects/torcs

Como se ha especificado en la base de requisitos de la comparativa, el coche elegido para todas las pruebas es el Alfa Romeo DTX-155, escogido por su tracción a las cuatro ruedas y 6 cilindros, proporcionan gran potencia, pero con una buena adherencia a la pista que facilitará la labor del diseño de los controladores.

Para generar este esqueleto del robot se ejecuta el comando *robotgen*, especificando el nombre del coche y el modelo de coche tal y como se muestra a continuación:

./robotgen –n RobotFL –a Mario Hernandez -c 155-DTM –gpl

Con esta línea desde la terminal de comandos, se generan varios archivos en la carpeta *drivers* entre el que destaca *RobotFL.cpp*, que es el script que se modifica para introducir los diferentes controladores que se van a diseñar en este TFM. El comando *robotgen* se ejecuta una vez por cada algoritmo de inteligencia artificial creado, creando así los robots llamados RobotFL (Fuzzy logic), RobotANN (Artificial Neural Network) y por último RobotCNN (Convolutional Neural Network).

5. Diseño del controlador borroso

El diseño del controlador borroso se realiza a través del software Matlab con la toolbox llamada Fuzzy Logic. Esta herramienta permite una manipulación gráfica e intuitiva de las reglas y las funciones de pertenencia. Será necesario hacer uso de un fichero que proporciona MATLAB llamado fis.c, que convierte los archivos.fis (extensión propia de MATLAB para trabajar con lógica borrosa) para que puedan ser entendidos en lenguaje C y C++, que es el lenguaje que utiliza el motor de TORCS.

Se decide diseñar dos controladores para este apartado de lógica borrosa, un controlador de volante y otro de velocidad. Ambos tendrán que trabajar de manera conjunta para el correcto funcionamiento del automóvil, pero la información de entrada que tienen en cuenta para sus respectivas salidas puede ser diferente. Se podría haber definido solo una salida que controlase tanto el volante como el freno del coche, pero esto habría añadido mucha complejidad a la definición de las reglas y una de las mayores ventajas que tiene esta técnica es que la definición de reglas permite crearlas de manera intuitiva.

5.1. Diseño del controlador borroso del volante

Para llevar a cabo un diseño adecuado del controlador de volante que se implementará en el vehículo elegido, primero se seleccionan, de las variables de entrada que son proporcionadas por simulador, cuáles se van a tener en cuenta en la entrada para producir una salida.

Se define que las variables de entrada sean:

- Error angular del vehículo (***angle***): está expresado en radianes y representa el ángulo entre la dirección del eje mayor del coche y el eje central del segmento en el que se encuentra el coche. Las pistas se dividen en segmentos o secciones que tienen un ángulo asociado a ellas. El rango de este movimiento puede variar desde 0 (en dirección a la carrera) hasta $\pm \pi$ (en dirección opuesta a la carrera).

- Error de posición lateral con respecto al centro de la pista (***trackPos***). Está expresada en metros y representa la distancia entre el centro del coche y el eje central del segmento en el que se encuentra el coche.

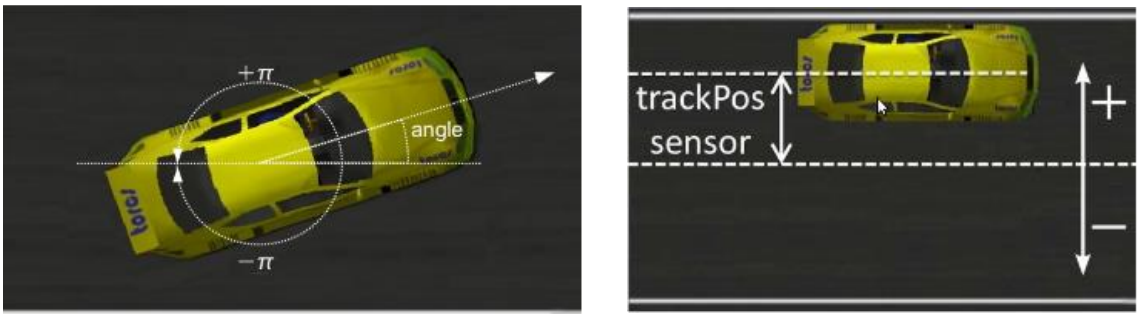

Figura 19. Visualización del error de ángulo (angle) y de posición (trackPos)

Fuente: sourceforge.net/projects/torcs

En el caso de la variable de salida, se define el giro del volante que deberá llevar a cabo nuestro vehículo con el fin de no salirse de la trazada. Esta salida, podrá variar entre -1 y 1, siendo cada uno de estos valores equivalentes a un giro del volante de 180 º en sentido horario o antihorario.

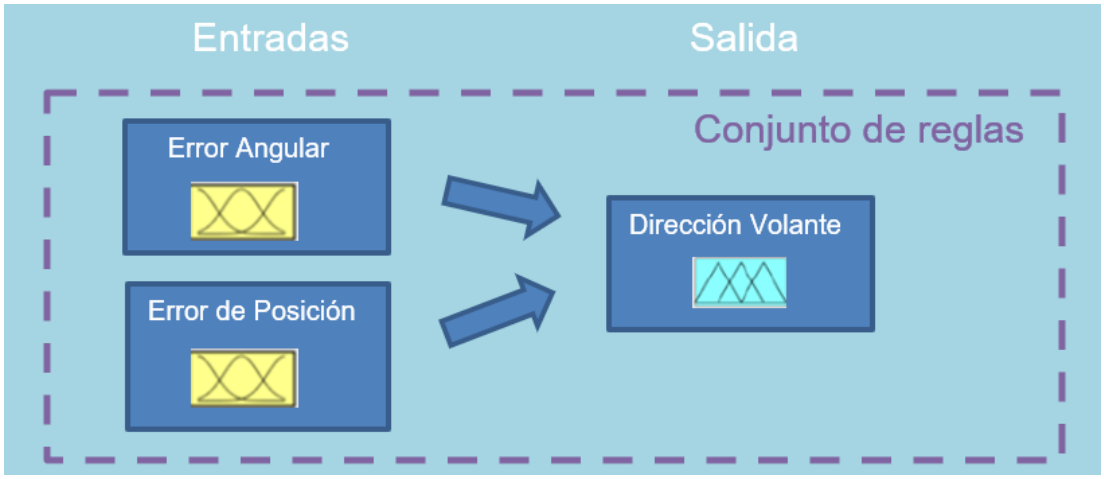

Figura 20. Esquema de entradas y salida del controlador del volante

5.1.1. Error angular

Para esta primera variable de entrada, se define un rango de entrada entre -1.74 y 1.74 radianes (+-100º). Sin embargo, las funciones de pertenencia que se utilizan en los extremos son del tipo triangular derecha e izquierda respectivamente de forma que para valores de entrada mayores a 100 º o menores a 100 º, la entrada que habrá saturado para un valor de 100º, tomará el valor máximo del conjunto borroso activo.

Para esta entrada, se definen los siguientes 3 conjuntos borrosos:

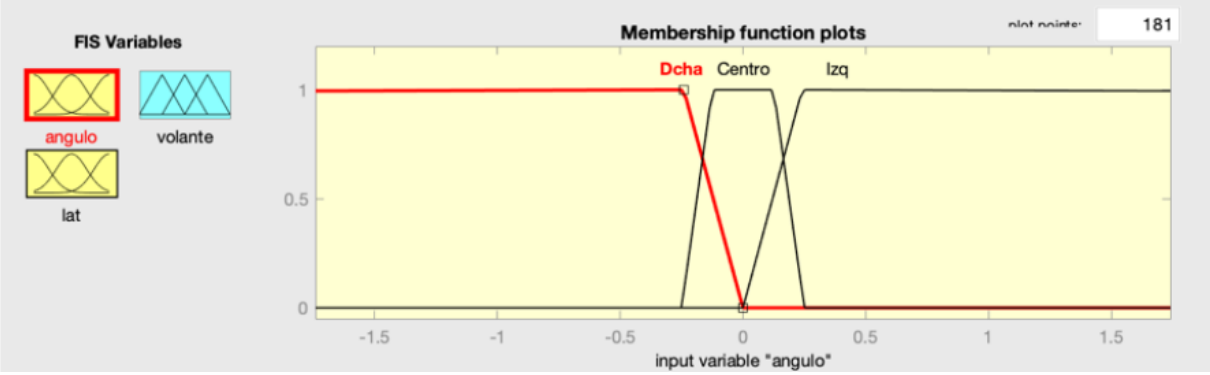

Figura 21. Función de pertenencia de la variable ángulo

Donde se aprecia que:

- La función de pertenencia “Dcha” (derecha) es del tipo triangular izquierda y sus parámetros son [-277 -0.241 0]
- La función de pertenencia “izq” (izquierda) es del tipo triangular derecha y sus parámetros son [0 0.241 277] (ambos conjuntos borrosos saturan a partir de un error angular de 0.241 rad = 13.08º).
- La función de pertenencia “Centro” es del tipo trapezoidal y sus parámetros son [-0.25-0.125 0.125 0.25].

El sentido de que tanto la función de pertenencia de la izquierda como de la derecha siempre estén activadas para una mínima desviación angular es el habitual; ir corrigiendo la posición para permanecer dentro de la pista, en vez de corregir la posición cuando el cambio de dirección tenga que ser muy brusco.

También es importante destacar cómo los valores asociados a la derecha son los negativos, mientas los propios de la izquierda son positivos. Esto es debido a que están definidos de esta forma en los parámetros internos del simulador.

5.1.2. Error de posición

Para esta segunda variable de entrada, se define un rango de [-7.5 7.5] (metros). Sin embargo, al igual que en el caso anterior, al utilizar funciones triangular derecha y triangular izquierda, la señal de entrada se saturará a partir de un valor determinado, con el fin de evitar que el coche se salga demasiado de la parte central de la carretera. Si se permitiesen movimientos

muy amplios, sería difícil definir el giro a implementar en nuestro volante, con el fin de retornar a la línea central de la carretera, sin que el giro sea tan brusco que provoque oscilaciones.

Para esta entrada, se definen también 3 conjuntos borrosos, que se pueden apreciar en la siguiente imagen:

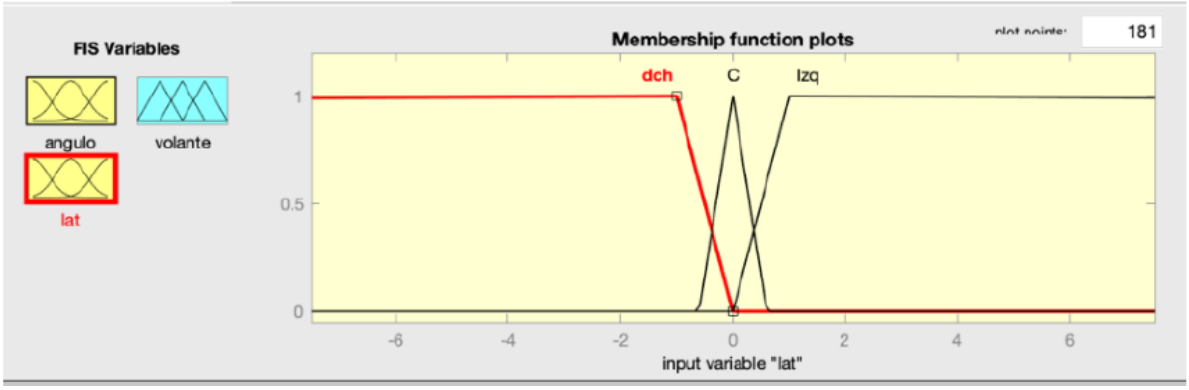

Figura 22. Funciones de pertenencia de la variale "error lateral"

Donde se puede apreciar que:

- La función de pertenencia “dch” (derecha) es del tipo triangular izquierda y sus parámetros son [-750 -1 0].
- La función de pertenencia “izq” (izquierda) es del tipo triangular derecha y sus parámetros son [0 1 750].
- La función de pertenencia “C” (centro) es del tipo triangular y sus parámetros son [-0.60 0.6].

Las funciones descritas en este caso son muy parecidas a las del caso del error angular, ya que se pretende de nuevo que las correcciones al volante sean leves y así evitar oscilaciones para mantenerse dentro de la pista.

5.1.3. Giro del volante

En el caso de la variable de salida, a pesar de que los límites se encuentran entre -1 y 1, se puede apreciar que esta salida se encuentra limitada por el centro de la base de las funciones de pertenencia I y D, de forma que el giro del volante pueda variar entre un mínimo de 0.9317 y un máximo de 0.9317.

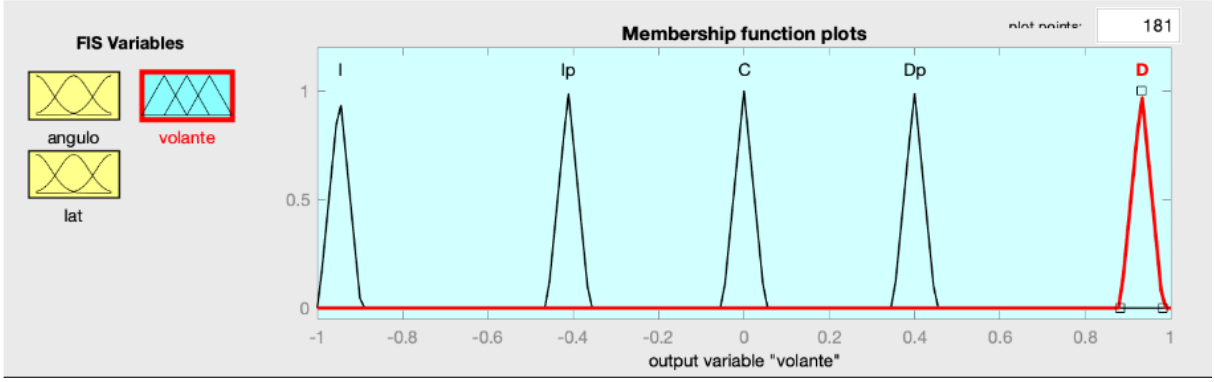

Figura 23. Función de pertenencia de la variable volante

Se establecen 5 conjuntos borrosos, con el fin de definir los valores de giro necesarios para nuestro controlador, los cuales son:

- I (Izquierda): Función de pertenencia triangular cuyos parámetros son [-0.9977 -0.9477 -0.8977]
- Ip (Izquierda pequeño): Función de pertenencia triangular cuyos parámetros son [-0.462 -0.412 -0.362]
- C (Centro): Función de pertenencia triangular cuyos parámetros son [-0.05 0 0.05]
- Dp (Derecha pequeño): Función de pertenencia triangular cuyos parámetros son [0.3494 0.3994 0.4494]
- D (Derecha): Función de pertenencia triangular cuyos parámetros son [0.8817 0.9317 0.9817]

Las funciones de pertenencia a la salida se han definido de forma simétrica, con el fin de que el área de cada conjunto borroso fuese el mismo, y así al utilizar el método de desborrosificación por centroide, evitar que unas funciones de pertenencia produzcan un desbalance respecto a la salida producida en una curva a derechas o un a izquierdas.

5.1.4. Base de reglas del controlador del volante

Puesto que se cuenta con un controlador con 2 entradas, y 3 conjuntos borrosos en cada una de las entradas, el número de reglas necesario para disponer de una base de conocimiento completa es 9 (todas las combinaciones tienen que estar contempladas).
Los principales casos que cabe observar son:

- Si el coche se encuentra en el centro, hacer que el volante no gire.
- Si el coche se encuentra a la izquierda del centro, hacer que el volante gire a la derecha.
- Si el coche se encuentra a la derecha del centro, hacer que el volante gire a la izquierda.

El principal problema, se produce cuando el vehículo se encuentra a la derecha del centro de la carretera, y el ángulo de inclinación es hacia la izquierda o al revés; es decir, el coche en posición lateral izquierda y orientación hacia la derecha. Cuando ocurría este caso, el vehículo giraba hacia el centro de la carretera, produciendo un movimiento de oscilación inestable. La solución consistió en girar hacia fuera del centro de la pista para minimizar las oscilaciones del coche.

Teniendo en cuenta todas estas premisas, se ha definido la siguiente base de reglas, que es capaz de dar respuesta a todas las necesidades de nuestro diseño.

Reglas del controlador del volante		Error angular		
		Derecho	Centro	Izquierdo
Error Lateral	Derecho	Izquierda	Derecha Peq.	Derecha peq.
	Centro	Izquierda	Centro	Derecha
	Izquierdo	Izquierda peq.	Izquierda Peq.	Derecha

Tabla 2. Reglas del controlador del volante

Se puede apreciar como la variable del error lateral queda en un segundo plano respecto a la del error angular, que es la que permite al vehículo mantenerse recto en la pista. Si se tiene la posición angular del coche desplazada a la derecha, el coche girará siempre a la izquierda (en distintos grados) con independencia del error actual en el que se encuentre el coche.

5.1.5. Superficie del controlador

La superficie del controlador permite tener una visión de cómo se accionará la salida, en este caso, el control del volante, teniendo en cuenta tanto el ángulo como la posición de la pista del vehículo. Esta representación tridimensional es posible debido a que se tienen solo tres variables a tener en cuenta ya que, si el número de variables fuese mayor, no se podrían observar la interacción entre entradas y salidas en un solo gráfico.

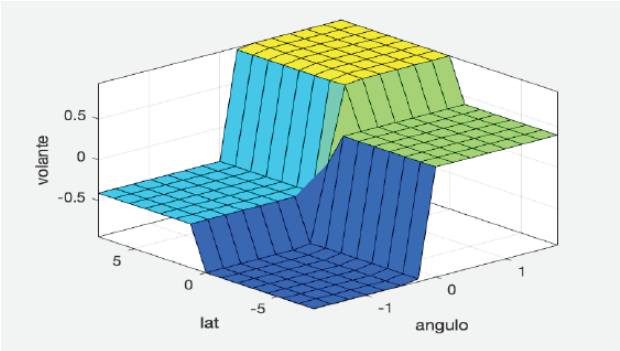

Figura 24. Superficie del controlador del volante

Se observan 4 superficies planas, que coinciden con los distintos giros de volante. El propósito de esta función es que el movimiento del volante no se mueva necesariamente en una de esas superficies planas, sino que se mueva la parte central, donde la salida se calcula haciendo medias entre las distintas funciones de pertenencia. Esto produce que el movimiento del volante no tenga una sensación de moverse en valores discretos.

5.2. Diseño del controlador borroso de velocidad

Para llevar a cabo un diseño adecuado del controlador de velocidad que implementaremos en el coche seleccionado, primero hemos de seleccionar cuáles serán las variables de entrada, y cuál será la variable de salida. Como variables de entrada se decide seleccionar:

- Tiempo al siguiente tramo (*TimetonextRadius*). Se mide en metros. Aunque esta variable no la proporciona de forma directa el simulador, se calcula haciendo una división entre los metros al siguiente tramo y velocidad actual. Poder agrupar estas dos variables en una sola, simplifica la complejidad a la hora de definir el conjunto de reglas. Se debe tener en cuenta que los tramos rectos pueden tener cientos de metros de longitud, mientras que los curvos son de longitudes mucho más reducidas.

Figura 25. División en tramos en los circuitos de TORCS

Fuente: sourceforge.net/projects/torcs

- Curvatura del tramo actual (*CurrentRadius*)
- Curvatura del siguiente tramo. (*NextRadius*)

Estas dos últimas variables se miden en radianes y permitirán saber si en una curva el vehículo está a la entrada o a la salida de la misma, viendo la relación entre la curvatura actual y la del siguiente tramo. En caso de que la curvatura actual sea menor que la siguiente, significa que se está entrando en una curva, por lo que habrá que frenar. Si ocurre de manera opuesta, se podrá acelerar saliendo de la curva, aunque todavía no se encuentre en un tramo completamente recto. Se obtienen valores altos para curvas amplias, valores bajos para curvas cerradas y 0 para rectas.

Con estas tres variables se es capaz de definir un conjunto de reglas que abarque todos los casos posibles encontrados en los circuitos a probar del simulador TORCS.

Para la salida se define la variable Velocidad con un rango de entre 0 y 300 Km\h, sin embargo, debido a las limitaciones físicas del vehículo, la velocidad máxima que alcanza será de aproximadamente 250 Km/h.

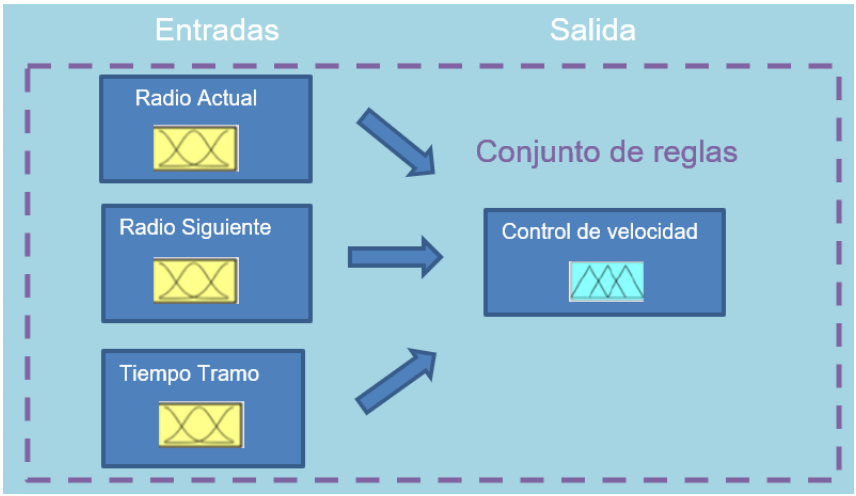

Figura 26. Esquema de las entradas y salidas del sistema

5.2.1. Tiempo hasta el próximo cambio de radio de curvatura

Esta variable va a ser de utilidad para saber cuánto tiempo hay hasta el siguiente cambio de radio de curvatura, permitiendo anticipar las frenadas y aceleraciones en función del tipo de tramo en el que se encuentra el vehículo y al que va a llegar.

En la figura 27, se observan las funciones de pertenencia de esta variable. Se han elegido dos funciones de pertenencia para optimizar el número de reglas ya que no se necesita diferenciar más allá de estar lejos o cerca del siguiente tramo. Cuando el coche se encuentra lejos del siguiente tramo, se especifica en las reglas que la velocidad se adapte al tramo actual y cuando se va acercando al siguiente tramo, se toma la decisión de frenar o acelerar.

El rango de esta variable se ha ajustado de 0 a 12 segundos, tomando 4 segundos como pequeño y 8 como grande. Estos valores se han seleccionado haciendo simulaciones en TORCS y viendo de una manera heurística el comportamiento del coche, hasta obtener una conducta adecuada a los valores elegidos.

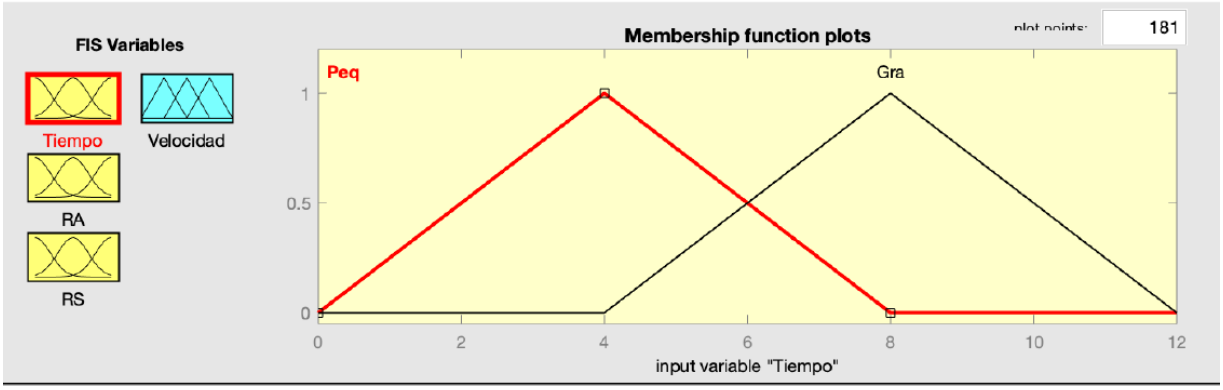

Figura 27. Función de pertenencia para la variable tiempo

5.2.2. Radio Actual y Radio Siguiente

Estas variables son idénticas pero que desempeñan funciones distintas en las reglas, aunque ambas miden la curvatura de los tramos del circuito.

El simulador TORCS interpreta 0 como tramo de recta, por lo que se define una función de pertenencia delta en 0 llamada “Recta”, que midiese esta situación.

Se definen otras tres funciones de pertenecía más: “Pequeño”, “Mediano” y “Grande”. Con estas tres funciones se pueden diferenciar tres tipos de curvas según su grado de curvatura, y junto a la función delta, son suficientes para definir un correcto funcionamiento del controlador.

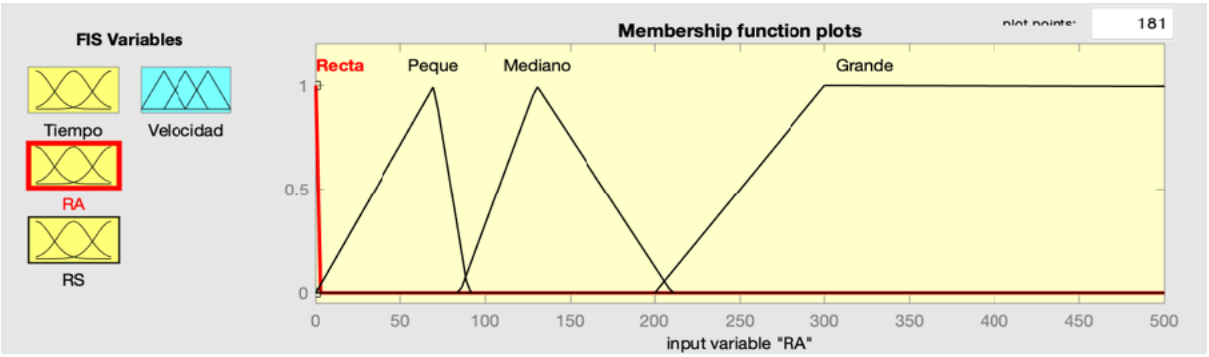

Figura 28. Funciones de pertenencia para la variable radio actual

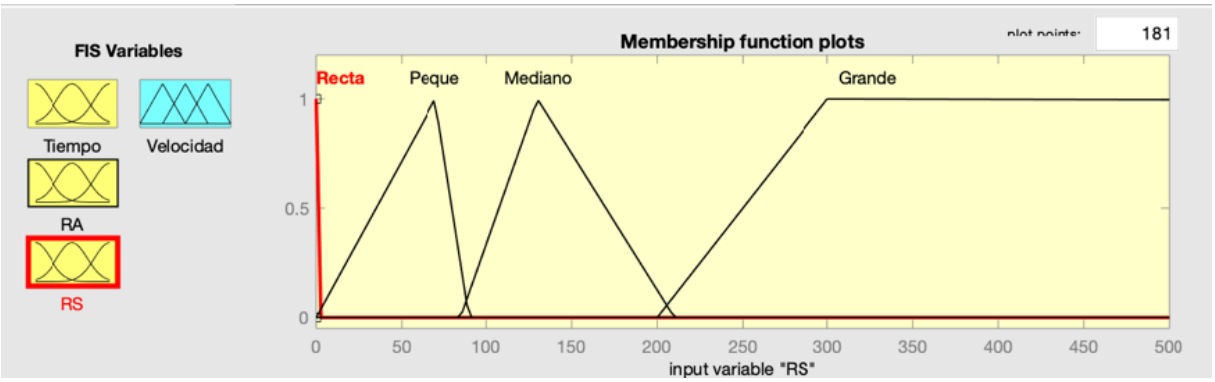

Figura 29. Función de pertenencia para la variable radio siguiente

5.2.3. Velocidad

Finalmente, se define para este controlador la variable de salida “Velocidad” con tres funciones de pertenencia, para precisar tres conjuntos borrosos: Velocidad baja, media y alta. Los valores de estas velocidades se obtienen de las velocidades del coche en cada tramo para otros controladores ya implementados en TORCS con este vehículo, observando que en tramos de curvas cerradas, el coche es capaz de ir a velocidades en torno a 110 km\h, en tramos de curvas amplias a 190 km\h y en rectas, a la máxima velocidad del coche, 250 km\h.

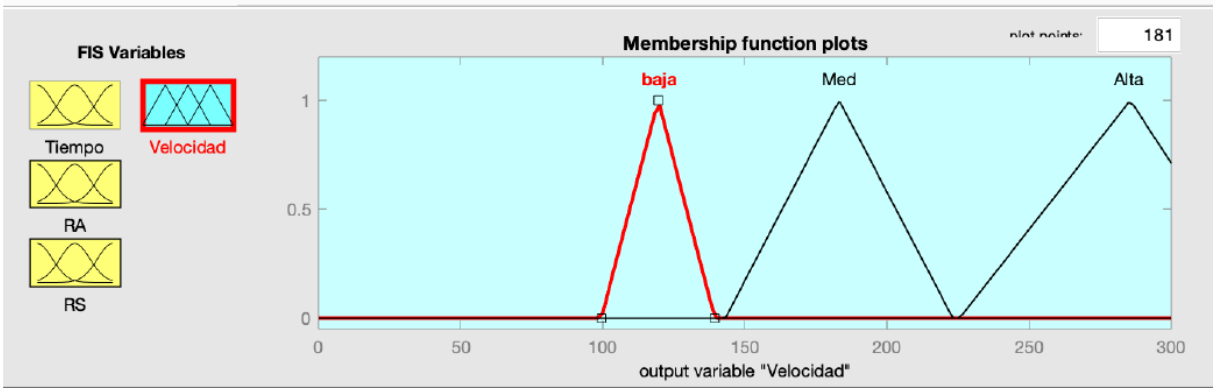

Figura 30. Funciones de pertenencia para la salida de velocidad

5.2.4. Base de reglas del controlador de velocidad

Puesto que contamos con un controlador de 3 entradas, dos de ellas con 4 conjuntos borrosos y una con 2, el número de reglas necesario para disponer de una base de conocimiento completa es 32.

Tiempo = Pequeño		Radio Siguiente			
		Recta	Pequeño	Mediano	Grande
Radio Actual	Recta	Alta	Baja	Alta	Alta
	Pequeño	Baja	Baja	Baja	Media
	Mediano	Media	Baja	Media	Alta
	Grande	Alta	Baja	Alta	Alta

Tabla 3. Salida de velocidad cuando la variable tiempo es igual a pequeño

Tiempo = Grande		Radio Siguiente			
		Recta	Pequeño	Mediano	Grande
Radio Actual	Recta	Alta	Alta	Alta	Alta
	Pequeño	Baja	Baja	Baja	Baja
	Mediano	Media	Media	Media	Media
	Grande	Alta	Alta	Alta	Alta

Tabla 4.Salida de velocidad cuando la variable tiempo es igual a grande

Con esta base de reglas definidas para el control de velocidad, se pueden apreciar dos paradigmas completamente diferenciados y que coinciden con cada una de las tablas propuestas. Cuando el tiempo a la siguiente curva es pequeño, se le da prioridad al radio siguiente, mientras que si el tiempo a la siguiente curva es grande, la prioridad pasa a ser el radio actual hasta acercarse a la próxima curva. Si por ejemplo el tiempo es grande y el radio actual es pequeño, la velocidad será siempre baja con independencia del valor que esté tomando el radio siguiente.

Inicialmente el controlador tenía 16 reglas. Esto era debido a que se intentó crear un controlador lo más sencillo posible en el que no se definía una función de pertenencia cuando el tramo era una recta. Esto imposibilitaba una aceleración total del coche en estos tramos y por lo tanto, penalizaba en gran manera el tiempo por vuelta. Finalmente, con 32 reglas abarcamos todo el control de velocidad teniendo un buen desempeño en los circuitos de prueba.

6. Diseño del controlador implementando redes neuronales artificiales

Tomando como referencia el controlador de redes neuronales artificiales realizado por Lex van Teeffelen (https://gitlab.com/Teeffelen/torcs-nn), se hará uso de la librería de aprendizaje automático Encog para la creación y desarrollo de las redes neuronales artificiales.

Encog es un *framework*[2] de aprendizaje automático escalable, adaptable y multiplataforma que fue creado en 2008 (Heaton, 2015). Está escrito principalmente en el lenguaje de programación Java con el objetivo de ayudar al desarrollo de la programación genética, *NEAT/HyperNEAT* y otras tecnologías de redes neuronales. Encog ha sido utilizado de manera frecuente desde su creación, tal y como lo demuestran las más de 1000 citas en artículos de investigación dentro de la plataforma G*oogle Scholar*. Fue precursor de *frameworks* de aprendizaje automático ampliamente utilizadas en la actualidad como TensorFlow, Keras o DeepLearning4J.

Encog soporta una gran variedad de algoritmos avanzados, entre las que se incluyen máquinas de vectores de soporte, redes neuronales, redes bayesianas, modelos ocultos de Markov, programación genética y algoritmos genéticos. Este *framework* sigue desarrollándose y corrigiendo errores no cubiertos por los otros frameworks proporcionando una implementación en Java de varias redes neuronales clásicas.

La elección de este *framework* de aprendizaje automático frente a otros, viene dada por su fácil implementación de redes neuronales artificiales. Para la formación de una red neuronal básica, solo se deben especificar los hiperparámetros de las neuronas de entrada, de salida y el número de capas ocultas con sus neuronas correspondientes (siempre pudiendo cambiar el resto de los parámetros si se requiere mayor grado de personalización).

Para poder utilizar este *framework* escrito en Java, se necesita una adaptación de la versión base del juego. A través del parche SCR_patch, instalamos el software de competición de TORCS que nos permitirá resolver esta necesidad.

El software de la competición, amplía la arquitectura original de TORCS en tres aspectos. En primer lugar, estructura TORCS como una aplicación cliente-servidor: los robots se ejecutan como procesos externos conectados al servidor de la competición a través de conexiones UDP (*User Datagram Protocol*). En segundo lugar, las acciones son ejecutadas en tiempo real: cada tic de juego (correspondiente a 20ms de tiempo simulado), el servidor envía las

[2] Un *framework* es un entorno de trabajo que facilita la labor de programación

entradas sensoriales actuales a cada robot y luego espera 10ms (de tiempo real) para recibir una acción del robot. Si no llega ninguna acción, la simulación continúa y se utiliza la última acción realizada. Por último, el software de competición crea una separación física entre el código del piloto y el servidor de carreras construyendo una capa de abstracción, un modelo de sensores y actuadores, que da total libertad de elección en cuanto al lenguaje de programación utilizado para los robots y restringe el acceso sólo a la información definida por el diseñador.

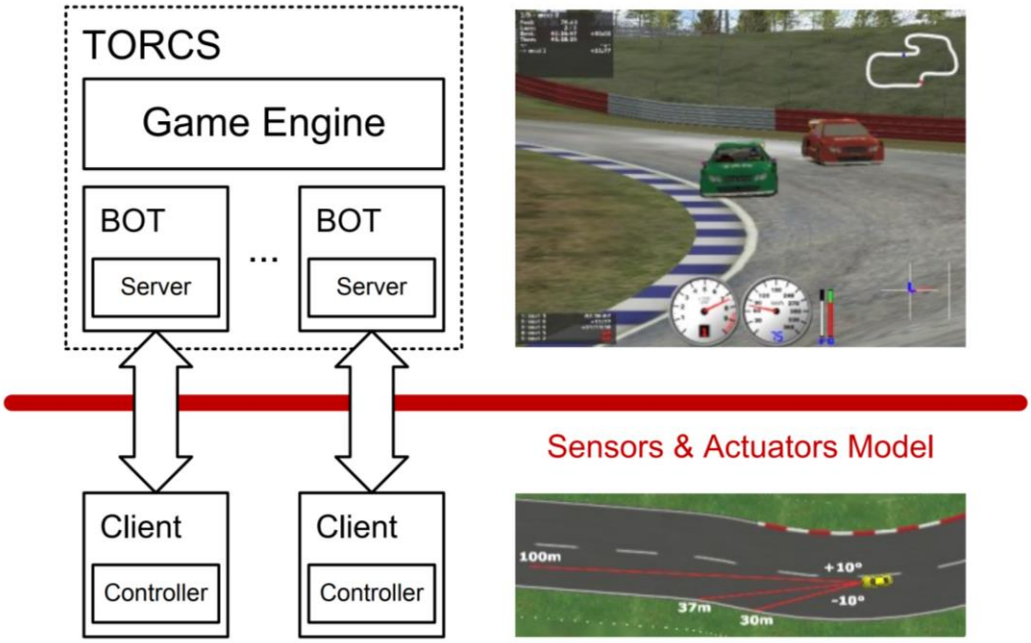

Figura 31. Arquitectura del software de competición de TORCS.

Fuente: (Loiacono, Cardamone, & Lanzi, 2013)

Tal y como se puede observar en la figura 31, la separación entre el servidor, llamado *scr_server* y el controlador desarrollado, permite que el lenguaje utilizado para realizar ese controlador sea distinto al lenguaje de programación C/C# en el que está desarrollado TORCS.

La instalación de este software requiere la implementación de TORCS 1.3.7 de igual manera que en el apartado de lógica borrosa, pero se le debe añadir el parche de competición para su correcto funcionamiento. Una vez añadido, se procederá a la construcción de TORCS:

```
$ export CFLAGS="-fPIC"
$ export CPPFLAGS=$CFLAGS
$ export CXXFLAGS=$CFLAGS
$ ./configure --prefix=$(pwd)/BUILD  # local install dir
$ make
$ make install
$ make datainstall
```

Figura 32. Comandos de instalación de TORCS Competition Software.

Fuente: https://github.com/fmirus/torcs-1.3.7

6.1. Tratamiento de los datos

En el parche SCR implementado, se añaden sensores variables adicionales, por lo que se hace una recopilación de los sensores y actuadores disponibles en esta versión.

Sensor (nombre)	Rango (unidades)	Descripción
angle	$[-\pi, +\pi]$ (rad)	Ángulo entre la dirección del coche y la dirección de los ejes de la pista
CurLapTime	$[0, +\infty)$ (s)	Tiempo transcurrido durante la vuelta actual
Damage	$[0, +\infty)$ (point)	Actual daño del coche (cuanto más alto es el valor, mayor es el daño)
distFromStartLine	$[0, +\infty)$ (m)	Distancia del coche desde la línea de salida a lo largo de la línea de la pista
distRaced	$[0, +\infty)$ (m)	Distancia recorrida por el coche desde el inicio de la carrera
focus	[0,200] (m)	Vector de 5 sensores de telemetría: cada sensor devuelve la distancia entre el borde de la pista y el coche en un rango de 200 metros. Cuando el coche está fuera de la pista (es decir, la variable racePos es menor que -1 o mayor que 1), la dirección está fuera del rango permitido ([-90,+90] grados) o los sensores ya han sido utilizados una vez en el último segundo, los valores devueltos no son fiables (normalmente se devuelve -1).
Fuel	$[0, +\infty)$ (l)	Niveles de gasolina actuales
Gear	$\{-1,0,1,\cdots 6\}$	Marcha actual: -1 es marcha atrás 0 es punto muerto y la marcha de 1 a 6
lastLapTime	$[0, +\infty)$ (s)	Tiempo para completar la última vuelta. Oponentes: Vector de 36 sensores que detecta la distancia del oponente en metros (el rango es [0,100]) dentro de un sector específico de 10 grados: cada sensor cubre 10 grados, de -π a +π alrededor del coche
opponents	[0,200] (m)	Vector de 36 sensores de localización de oponentes: cada sensor cubre un rango de 10 grados dentro de un rango de 200 metros y devuelve la distancia del oponente más cercano en el área cubierta. Los 36 sensores cubren todo el espacio alrededor del coche, abarcando en sentido de las agujas del reloj desde -180 grados hasta +180 grados con respecto al eje del coche.
racePos	$\{1,2,\cdots,N\}$	Posición en la carrera con respecto a otros coches
rpm	$[0, +\infty)$ (rpm)	Número de rotaciones por minuto del motor del coche
speedX	$(-\infty, +\infty)$ (km/h)	Velocidad del coche a lo largo del eje longitudinal del coche
speedY	$(-\infty, +\infty)$ (km/h)	Velocidad del coche a lo largo del eje transversal del coche

speedZ	$(-\infty,+\infty)$ (km/h)	Velocidad del coche a lo largo del eje Z del coche
track	$[0,200]$ (m)	Vector de 19 sensores de telemetría: cada sensor representa la distancia entre el borde de la pista y el coche. Los sensores están orientados cada 10 grados desde -π/2 y +π/2 por delante del coche. Las distancias están en metros dentro de un rango de 200 metros. Cuando el coche está fuera de la pista (es decir, la posición en la pista es menor que -1 o mayor que 1), estos valores no son fiables.
trackPos	$[-1,1]$	Distancia entre el coche y el eje de la pista. El valor se normaliza con respecto al ancho de la pista: es 0 cuando el coche está en el eje, -1 cuando el coche está en el borde izquierdo de la pista y +1 cuando está en el borde derecho del coche. Los valores mayores que 1 o menores que -1 significan que el coche está fuera de la pista
wheelSpinVel	$[0,+\infty]$ (rad/s)	Vector de 4 sensores que representan la velocidad de rotación de las ruedas
z	$[-\infty,+\infty]$ (m)	Distancia del centro de masas del coche desde la superficie de la pista a lo largo del eje Z.

Tabla 5. Tipos de sensores disponibles en TORCS.

Fuente: (Loiacono, Cardamone, & Lanzi, 2013)

Acción (nombre)	Rango (unidades)	Descripción
Accel	$[0,1]$	Acelerador virtual (0 significa que no se está produciendo una aceleración, mientras que 1 es aceleración máxima)
Brake	$[0,1]$	Pedal de freno virtual (0 significa que no se está frenando, mientras que 1 es freno máximo)
clutch	$[0,1]$	Valor del embrague (0 significa no pisado, 1 significa pisado)
Gear	$-1,0,1,\cdots,6$	Valor de la marcha actual
Steering	$[-1,1]$	Valor de la dirección: -1 y +1 significan, respectivamente, izquierda y derecha, lo que corresponde a un ángulo de 0,785398 rad
focus	$[-90,90]$	La dirección de atención (ver tabla 5) en grados
Meta	$0,1$	Este es el comando de meta-control: 0 No hacer nada, 1 pedir al servidor de competición que reinicie la carrera

Tabla 6. Tipos de acciones disponibles en TORCS.

Fuente: (Loiacono, Cardamone, & Lanzi, 2013)

Los datos de los sensores y actuadores recogidos previamente y con lo que se va a tratar, son los proporcionados por Lex van Teeffelen en su repositorio, ya que presentan una gran cantidad de datos de coches ya conducidos en distintos circuitos.

Una vez visto los sensores proporcionados por el simulador, se determina cuáles van a componer los datos de entrada de la red neuronal. Las **variables de entrada** las componen: el ángulo, la posición de la pista y la velocidad al igual que en el controlador borroso y, adicionalmente, añadiremos los 19 sensores que componen la variable *track*. Cada uno de ellos indica la distancia existente desde el centro del coche hasta el extremo de la pista en

ángulos de 10 grados hasta completar los 180 grados de la visión frontal del vehículo (véase la figura 33).

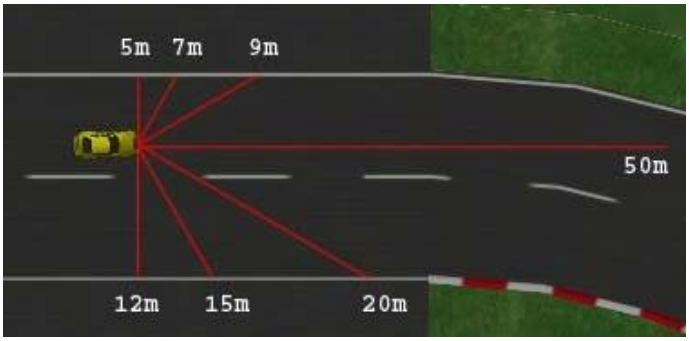

Figura 33. Sensores de distancia del límite de la pista.

Aunque se podría haber hecho uso de otras variables de entrada (marcha, gasolina etc.), estas aumentarían la complejidad de la red, por lo que se ha escogido las variables que pueden determinar en mayor medida los valores de salida. En cualquier caso, la propia red neuronal artificial es la encargada de ajustar los pesos en la fase de entrenamiento, de manera que dote a cada variable su importancia asociada.

Mientras que el controlador borroso requería un conocimiento experto para la definición de reglas a partir de unas variables concretas, las redes neuronales pueden funcionar como “cajas negras”, donde ellas mismas se encargan de asignar la importancia de las variables de entrada. De esta manera, se observa que en casos donde existan muchas variables de entrada y de salida a tener en cuenta, las redes neuronales ofrecen una ventaja notable frente a la lógica borrosa.

Las salidas de la red neuronal artificial, que a su vez serán **actuadores del robot**, son el pedal de freno, el de aceleración y el de volante. Tanto el embrague como el cambio de marchas no estarán influenciado por la red neuronal, dejando así su funcionamiento por defecto dependiendo del coche elegido.

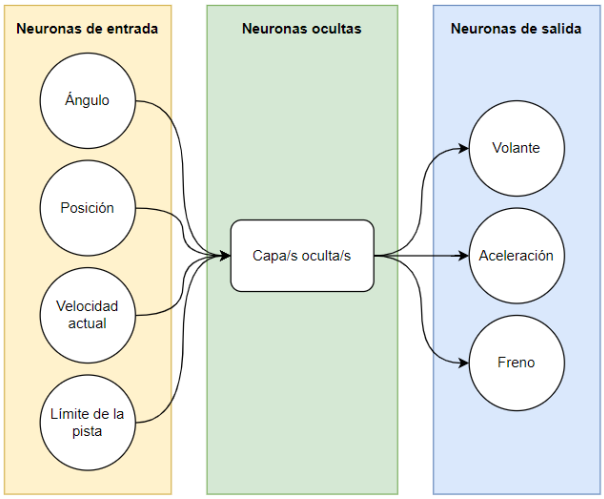

Figura 34. Arquitectura general de la red neuronal artificial a implementar[3]

Se puede comprobar que existe gran variabilidad entre los rangos definidos para cada una de las variables de entrada. Esto puede hacer que se produzca un ajuste incorrecto de pesos, confundiendo valores altos de entrada con la importancia que la red asigna a la variable. De esta manera, se tendrá que realizar una normalización de datos de entrada en la que la función de activación elegida puede tener un papel importante. La normalización aplicada de manera general es la siguiente:

$$y = \frac{(x - x_{min})(d2 - d1)}{x_{max} - x_{min}} + d1$$

Donde:

- x es el valor de entrada
- y es el valor de salida normalizado
- $[x_{min}, x_{max}]$ son los valores entre los que está comprendida la variable de entrada
- $[d1, d2]$ son los valores entre los que va a estar comprendida la variable de salida

Se realiza una normalización de las variables de entrada que va de -1 a 1. En el caso del sensor de velocidad, aunque el rango teórico de esta variable de cero a infinito, la normalización se realiza tomando como valor máximo 400 km/h, ya que es una velocidad límite e inalcanzable para los coches del simulador.

Un tratamiento de datos usual es la reducción en frecuencia de los valores más comunes para que el modelo sea capaz de aprender patrones que estén presentes en menor medida. Este concepto aplicado a la conducción autónoma, podría darse en la diferencia existente entre la

[3] Por motivos de visualización, el esquema no representa la totalidad de variables de entrada

cantidad de datos recogidos de rectas frente a los recogidos en curvas muy cerradas, ya que la mayor parte del tiempo del automóvil en la pista, comprende rectas o curvas leves, mientas que las curvas muy pronunciadas pueden estar poco representadas.

6.2. Entrenamiento de la red

Se define una arquitectura común para distintos entrenamientos. A partir de esa arquitectura, podremos modificar una serie de hiperparámetros con los que se extraen la mejor combinación de estos. Se realizará un entrenamiento para cada una de las arquitecturas y se elegirá la red que menos error proporciona para los datos de test.

Dada la gran cantidad de posibilidades existentes de modificación de hiperparámetros de la red, se definirán unos parámetros fijos que afectarán tanto a la arquitectura de la red como al proceso de entrenamiento.

- La capa de entrada tiene las 22 neuronas que corresponden variables de entrada.
- La capa de salida tiene 3 salidas que corresponden con los actuadores del automóvil elegidos.
- Optimizador: *Resilient Back Propagation* (Rprop)
- El entrenamiento se hará por 5000 *epochs* o ciclos de entrenamiento.
- Habrá una, dos o ninguna capa oculta con el mismo número de neuronas que la capa de entrada.
- La función de activación para todas las capas será lineal, tangente hiperbólica o sigmoidea.

Por lo tanto, se variará tanto la función de activación y el número de capas ocultas (la profundidad de la red) manteniendo el número de neuronas de estas (anchura de la red). De esta manera, se comprueba qué grado de complejidad se ajusta más a los requisitos del sistema, evitando tanto el *underffiting* como el *overfitting* o sobreentrenamiento.

Estos son los resultados obtenidos después del entrenamiento:

Error de entrenamiento (MSE)		**Número de capas ocultas**		
		0	**1**	**2**
Función de activación	**Linear**	0,2088	0,2087	0,2088
	Tanh	0,1909	0,1547	**0,1386**
	Sigmoid	0,1742	0,1843	0,2654

Tabla 7. Resultado del entrenamiento de las ANN

Después observar los valores de MSE (Mean Square Error) presentados en la tabla 7, se puede concluir que tanto el número de capas ocultas, como las funciones de activación, juegan un papel muy importante a la hora que buscar que la red neuronal artificial que pueda producir unos mejores controladores. Se observa cómo, independientemente del número de capas ocultas que presente la red, la función de activación **lineal** es incapaz de ajustarse bien al espacio de características. Esta función de activación tiene el mismo valor tanto de entrada como de salida por lo que, aunque tiene la ventaja de ser simétrica y que abarca un rango infinito de valores de salida, no aporta ningún valor añadido a la neurona.

La función **sigmoidea** no presenta tampoco los mejores resultados, incluso superando en error a la función lineal cuando la red presenta dos capas ocultas. Se observa un claro *overffiting* al aumentar la complejidad de la red, ya que esta complejidad hace que no sea capaz de aprender patrones de acción comunes, siendo muy reactivo a cualquier ruido que puedan presentar los datos de entrada. El error que presenta, también está dado por los rangos de salida que presenta la función de activación sigmoidea. Aunque, al igual que la función sigmoidea, tanto la salida de freno como de acelerador están comprendidas de 0 a 1, la salida del volante comprende desde el -1 al 1. Esto provoca que esta función de activación solo pueda tener la capacidad de girar en una dirección y provocando un mal funcionamiento del controlador independientemente del error de entrenamiento.

Por último, la función **tangente hiperbólica** es la función de activación que presenta unas mayores prestaciones, especialmente cuando aumentamos la profundidad de la red a 2 capas ocultas. Esta función está comprendida entre -1 y 1, por lo que está centrada en 0 y es capaz de representar la totalidad de las variables de salida. Aunque en determinadas ocasiones esta función de activación se satura con valores de entradas grandes, reduciendo el valor del gradiente al mínimo, en este caso ha demostrado ser la que menor error presenta en el conjunto de validación.

Por todas las conclusiones extraídas de los entrenamientos anteriores, escogemos la arquitectura que contienen 2 capas ocultas con la función de activación tangente hiperbólica para realizar la comparación de los distintos algoritmos de conducción autónoma.

7. Diseño del controlador implementando redes neuronales convolucionales

Se pretende diseñar un controlador de automóvil basado en un sistema de visión artificial, aplicando redes neuronales convolucionales para su implementación en TORCS.

Existen dos paradigmas para la conducción autónoma basados en visión artificial (Chen, Seff, Kornhauser, & Xiao, 2015). El primero, es la conocido como método de la percepción mediada (*mediated perception approach*), el cual reconoce y localiza distintas entidades que sean potenciales condicionantes de la conducción (semáforos, líneas, coches, etc.) y usa ese conocimiento para tomar decisiones que pueden o no estar implementadas con algoritmos de inteligencia artificial.

El segundo paradigma es conocido como el método de reflejo del comportamiento (*behavior reflex approach*), que procesa directamente los datos de entrada (la imagen completa) para tomar una decisión de volante producida de principio a fin en solo un paso.

Como se puede deducir de lo dicho en los párrafos anteriores, el primer enfoque está relacionado con las tareas de detección, por la cual se toma una decisión con base en la categoría y posición de las entidades encontradas. Por otro lado, el segundo enfoque está relacionado con la tarea de clasificación, en tanto que utiliza toda la imagen como entrada para determinar una solución. Ambas tienen ventajas y desventajas: para el método de percepción mediada, el continuo análisis de la imagen para la localización de objetos, que muchas veces no aportará información relevante para el objetivo de la conducción, puede resultar computacionalmente costoso. En cambio, para el método del reflejo del comportamiento, aunque no tiene este problema, pueden aparecer mayores complicaciones a la hora de tomar decisiones en ambientes complicados (tráficos densos, movimiento de personas, gran cantidad de objetos en la imagen), ya que no hace una evaluación individual de cada una de las características que engloban la imagen de entrada.

En el artículo académico de Chen et al. (2015), creadores del modelo DeepDriving, se implementa un método que combina los dos anteriores para resolver el problema de la conducción autónoma. Este nuevo método conocido como percepción directa (*direct perception approach*), es un método híbrido donde toma tanto información de los objetos presentes en la imagen como de toda la imagen para tomar una decisión de la salida. En la figura 35, se puede observar una representación gráfica del funcionamiento de los diferentes métodos.

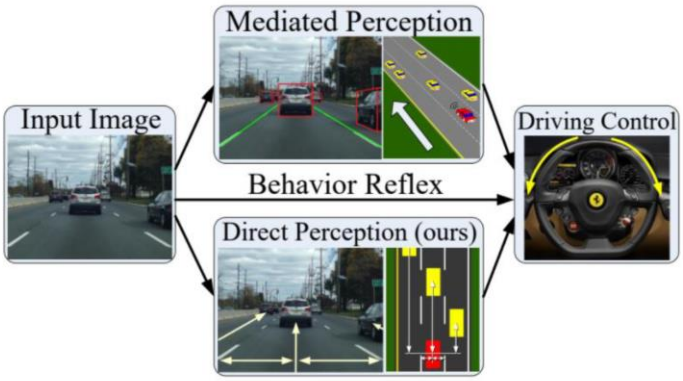

Figura 35. Métodos basados en visión artificial para la conducción autónoma.

Fuente: (Chen, Seff, Kornhauser, & Xiao, 2015)

En la solución aportada en este apartado, se empleará el método de reflejo del comportamiento para implementar el controlador del coche. Para aplicaciones donde solo existe interacción entre la pista sin obstáculos y el coche, este acercamiento es el más adecuado debido a su mayor eficiencia computacional y desempeño en ambientes poco concurridos.

Se hace uso otra vez del parche SCR_patch de competición para la conexión entre el simulador de coches y la implementación del algoritmo de inteligencia artificial. Se instala para este fin, la versión que implementa ROS (*Robot Operating System*), siguiendo los pasos e información aportada en el repositorio de fmirus (https://github.com/fmirus/torcs_ros). El *framework* ROS, además de implementar una capa de abstracción y enviar información de variables numéricas, facilita la extracción de las imágenes necesarias para el procesamiento de la imagen, tanto en el proceso de entrenamiento como de inferencia. Este proceso de captura de imágenes, es posible gracias a la integración con OpenCV, una librería libre de visión artificial que destaca por sus funcionalidades relacionadas con el tratamiento y procesado de imágenes.

Para la creación de la estructura de las CNN, así como todo el procesado de datos, se utilizará el *framework* Tensorflow de aprendizaje automático implementado en Python, ya que la librería Encog propuesta para la creación de las ANN, no tiene desarrollo para tareas de visión artificial. Tensorflow es un *framework* de código abierto desarrollado por investigadores de Google con la finalidad de ser utilizado para aplicaciones de aprendizaje automático, aprendizaje profundo y análisis estadísticos y predictivos. Utiliza grafos de computación en forma de tensores para proporcionar una computación más rápida y eficiente (Goldsborough,

2016). Dentro de Tensorflow, se hará uso del módulo llamado Keras. Esta librería de alto nivel, fue desarrollada con el objeto de facilitar un proceso de experimentación rápida.

7.1. Recolección y tratamiento de los datos

Se propone un modelo de recolección de datos (véase la figura 36) en el que se necesita por un lado la imagen, y por otro la salida del volante asociada a dicha imagen. Estos datos son los que posteriormente servirán como entrada para el entrenamiento de las CNN. La aplicación de los actuadores del acelerador y del freno como salida de la red convolucionales proporciona resultados muy pobres en su implementación en el simulador. Esto puede estar motivado por el hecho de no almacenar valores pasados de velocidad, por lo que, por una única imagen de entrada, la red no es capaz inferir la velocidad a la que entra el automóvil en una curva dada. Se podría plantear una estructura híbrida *(Almeida & de Castro, Paulo André Lima, 2019), en* que se combina las CNN con las redes neuronales recurrentes. Esto trabajaría como con una función de memoria orientada a almacenar datos de velocidades pasadas y a partir de esos valores, hacer predicciones futuras. Por lo tanto, el control de velocidad, se aplicará con el algoritmo por defecto que incluye el automóvil utilizado.

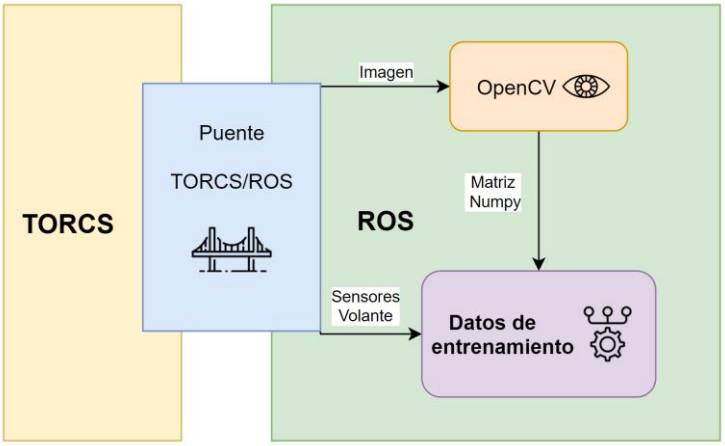

Figura 36. Proceso de recolección de datos.

Este proceso de recolección de datos se hará en 3 circuitos distintos de los propuestos en la sección de planteamiento de la comparativa, ya que eso supondría en estos últimos un desempeño superior debido a la memorización que pudieran tener de los datos de entrada. Se utiliza para este propósito de recolección de datos, la conducción del vehículo de manera manual durante diez vueltas en cada uno de los circuitos. Se recogerán tanto datos de las imágenes capturas en primera persona como la posición del volante, con una frecuencia de imagen de 5 Hz a la resolución nativa de del simulador (480x640). Se asume que una

frecuencia de muestreo mayor no aporta información adicional, ya que no representa una variación significativa sobre el fotograma anterior.

Con el fin de acelerar el proceso de entrenamiento, las imágenes obtenidas son primero recortadas y luego reescaladas, obteniendo una imagen final con resolución 66x200 a color. El recorte de la imagen también está orientado a reducir parte del entorno que no aporta información significativa para el desempeño de la conducción y que además pueden entorpecer la labor de entrenamiento de la red.

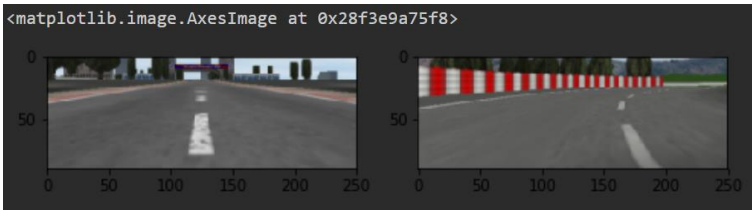

Figura 37. Muestra de las imágenes reescaladas y recortadas.

7.2. Fase de entrenamiento

Como una primera aproximación a la creación de distintas CNN, se utilizan las redes conocidas comúnmente como AlexNet (Krizhevsky, Sutskever, & Hinton, 2012) y NvidiaNet (Wang, Liu, Jeon, Chu, & Matson, 2019). Se ha hecho esta elección de arquitectura debida a que ambas redes han sido propuestas anteriormente para soluciones de conducción autónoma en el campo de la visión artificial. Mientras que AlexNet es la CNN utilizada en el proyecto de DeepDriving, NvidiaNet es el resultado de la modificación de la red creada una investigación académica hecha por Nvidia titulada *End to End Learning for Self-Driving Cars* (Bojarski et al., 2016).

A grandes rasgos y tal como se observa en la figura 38, AlexNet presenta dos bloques seguidos que concatenan una capa de convolución y otra de *maxpooling*. Presenta adicionalmente tres capas convolucionales, una capa de *pooling* y tres capas *fully connected* para dar la salida del volante.

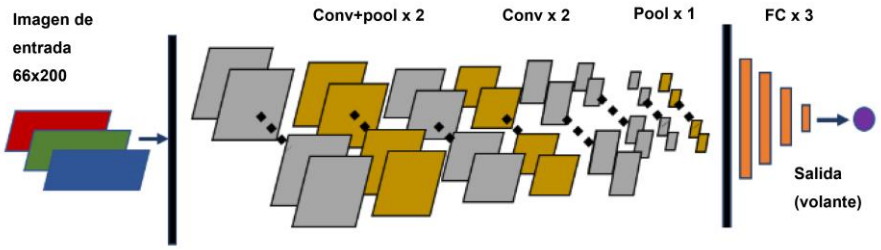

Figura 38. Arquitectura de la red AlexNet

Por otro lado, la red NvidiaNet representada en la figura 39, presenta una estructura más sencilla y ligera con relación al número de parámetros de la red. La imagen de entrada es primero sometida una capa de normalización, a cinco capas de convolución y, al igual que en la red AlexNet, cuenta para terminar con tres capas *fully connected* que proporcionan la salida de la red.

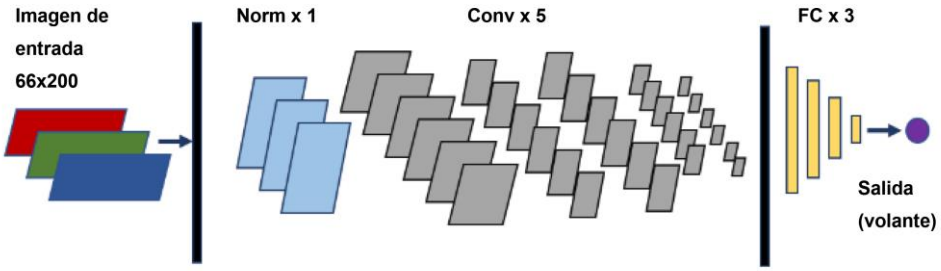

Figura 39. Arquitectura de la red NvidiaNet

Las arquitecturas propuestas, aunque se ha intentado mantener la estructura lo más parecida posible a la definición de sus creadores, han sufrido dos modificaciones principales. La primera es el cambio en el tamaño de la imagen de entrada, ya que en ambas arquitecturas, la imagen de entrada era cuadrada y de distinto tamaño. La otra modificación propuesta, es el cambio de función de activación de la neurona de salida; de Softmax a tangente hiperbólica. Esto es debido a que, mientras que la función de activación Softmax solo tiene de rango de salida de 0 a 1, la salida del volante en TORCS debe tomar valores de -1 a 1 (izquierda y derecha, respectivamente), por lo que necesitamos una función de activación que cubra todo ese rango para un correcto funcionamiento del controlador.

Una vez vista la arquitectura de las redes y con la finalidad de hacer un entrenamiento lo más equitativo posible, se establece un marco común de definición de hiperparámetros, los cuales serán los utilizados para la realización del proceso de entrenamiento.

- Tamaño de imagen de entrada: 66x200x3 píxeles
- Normalización L2: 0,001
- Learning Rate: 0,005
- Iteraciones (*epoch*s): 2000
- Tamaño del lote (*batch size*): 20 imágenes/lote

- 80% de datos reservados para entrenamiento y 20% para validación.
- Optimizador: Adam
- Métrica de error: Error medio cuadrático (MSE)

Dado el coste computacional y el tiempo requerido para el entrenamiento de cada una de las redes (aproximadamente 5 horas), no se hace una exploración más a fondo de los parámetros que proporcionarían potencialmente mejores resultados. En cualquier caso, el objetivo de este TFM se centra en hacer la comparativa, ventajas y desventajas de la aplicación y de los diferentes algoritmos de inteligencia artificial propuestos, y no tanto en el ajuste óptimo de cada uno de ellos. Una vez realizado el entrenamiento de las CNN, se obtienen los siguientes resultados:

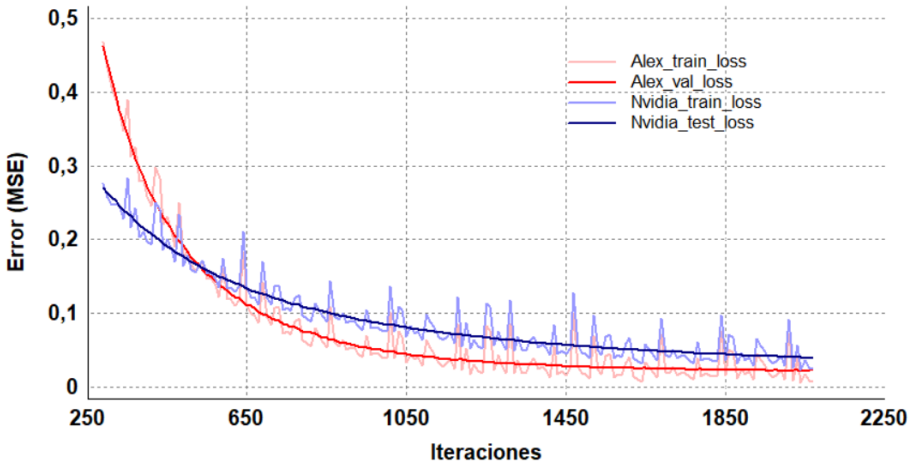

Figura 40. Resultados del entrenamiento de las redes

Se observa que para el entrenamiento en 2000 iteraciones y con el resto de hiperparámetros descritos anteriormente, la red conocida como AlexNet presenta mejores resultados de NvidiaNet, a pesar de ser una red más longeva y no estar diseñada de manera específica para el problema de la conducción autónoma. En ambas CNN, se puede ver cómo, a pesar de que la pérdida de validación sufre de una mayor inestabilidad que la pérdida de entrenamiento, amabas siguen la misma tendencia descendente. Esto hace suponer que ninguna de las dos redes está sufriendo de *overfitting* y que se podrían ampliar las iteraciones definidas en un principio con el fin de disminuir aún más el MSE.

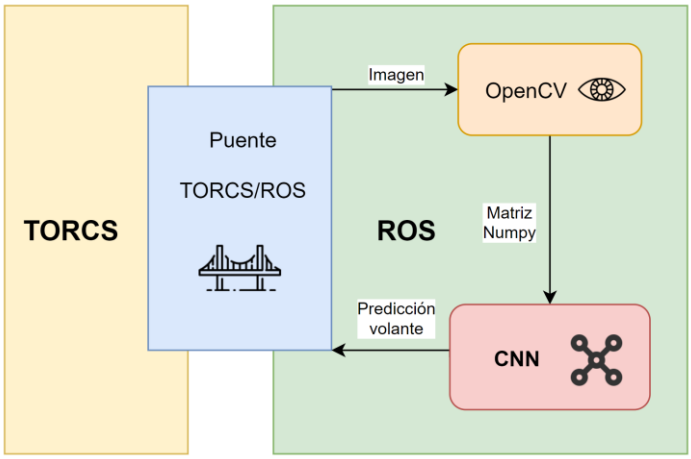

Figura 41. Proceso de predicción o inferencia

Una vez entrenada la red, está lista para aplicarse dentro del simulador. Utilizando una estructura similar a la propuesta en la recolección de datos, se recogen las imágenes proporcionadas por TORCS con la vista en una hipotética cámara frontal del coche. Para un funcionamiento óptimo, estas imágenes tendrán que ser procesadas de la misma manera que en la fase de entrenamiento, de tal manera que puedan ser la entrada de la CNN (imagen a *array* numpy, recorte y reescalado). Por último, la CNN implementada se encarga, a partir de las imágenes de entrada, dar un valor de la variable del volante y este será enviado de nuevo al simulador para modificar el movimiento del coche.

8. Discusión y análisis de resultados

Una vez que se ha realizado el desarrollo y la implantación de cada uno de los controladores en el simulador TORCS, se procede a comentar las ventajas y desventajas que tiene cada uno de los algoritmos a nivel general. Mas tarde, se analizan los resultados obtenidos mediante la prueba de los diferentes controladores en los circuitos propuestos en el apartado de planteamiento de la comparativa.

8.1 Análisis del proceso de implantación

Una vez realizada la implantación de cada uno de los algoritmos de inteligencia artificial en el simulador, se extraen una serie de concusiones descritas a continuación:

- Los algoritmos que implementan lógica borrosa necesitan una persona “experta” que entienda el funcionamiento del sistema para poder definir unas reglas que se obtienen de la experiencia. Sin embargo, las aplicaciones con CNN y ANN pueden funcionar como “cajas negras” en donde a partir de unas entradas, se producen unas salidas que modelan una función matemática que es desconocida.

- Tanto los algoritmos de ANN como de CNN precisan de gran cantidad de datos para funcionar, ya que a través de estos son capaces de aprender patrones comunes de comportamiento para su propia red. Si los datos están corruptos, son insuficientes, no están bien normalizados o alguna categoría está más presente que otra, se puede producir un incorrecto funcionamiento de este tipo de algoritmos.

- El hecho de que se tenga que definir una regla para cubrir todos los casos en lógica borrosa, hace que sea inviable plantear este problema cuando el número de variables tanto de entrada como de salida es elevado, ya que por cada variable, la complejidad de la definición de reglas crece de manera exponencial. Mientras tanto para las ANN y CNN, aunque aumente el coste computacional de estas, el tener mayor número de entradas o de salidas en la red aumenta la complejidad de su estructura, pero no de su definición o creación.

- Las CNN presentan un coste computacional mucho mayor que la lógica borrosa y aunque en menor medida, mayor que las ANN. El hecho de trabajar con imágenes (que incluyen gran cantidad de parámetros) hace que tanto el proceso de entrenamiento como el de inferencia sea mucho mayor, por lo que se necesita un

hardware que sea capaz de procesar tanta cantidad de información en el tiempo requerido.

- Mientras que la lógica borrosa ha sido utilizada de manera muy prolija en décadas anteriores, las CNN y ANN están ganando clara popularidad ligada al aumento del rendimiento de procesadores y GPU, además de por la capacidad que tienen estas redes de computar grandes cantidades de datos con una precisión muy alta. Esto hace que tanto las CNN como las ANN sigan en continuo progreso actualmente, surgiendo nuevas arquitecturas y formas de optimización de estas de manera recursiva.

8.2 Análisis de resultados

Se hace un análisis exhaustivo del desempeño de los diferentes algoritmos (implementados en RobotFL, RobotANN y RobotCNN) para cada uno de los circuitos probados (C1, C2 y C3). Las variables a considerar en la comparativa son el tiempo medio por vuelta (de la vuelta 2 a la vuelta 5), las colisiones producidas por el robot, la velocidad máxima alcanzada y el tipo de conducción. Para esta última prueba, se evaluarán distintos parámetros como puede ser la trazada, las oscilaciones o las salidas de pista para ordenar los diferentes robots de 3 a 1 (siendo 3 el mayor puntaje que se puede obtener en esta categoría).

	Tiempo medio por vuelta (min:seg:cen)			**Colisiones**	**Velocidad máxima (Km/h)**			**Conducción**
	C1	**C2**	**C3**	**C1, C2, C3**	**C1**	**C2**	**C3**	**C1, C2, C3**
RobotFL	58:93	1:08:11	44:68	0	249	234	207	3
RobotANN	3:12:23	1:35:87	1:59:73	0	39	100	81	2
RobotCNN	55:32	1:01:43	41:23	5	240	221	201	1

Tabla 8. Comparación de los controladores

Haciendo un primer acercamiento a los resultados obtenidos, se puede observar que, aunque el tiempo medio por vuelta de la RobotCNN es superior al del RobotFL, ambos tienen un desempeño parecido en esta categoría. Sin embargo, la actuación del robot implementado

con ANN, es bastante inferior en cuando a tiempo por vuelta. Esto es debido a que, aunque presenta una buena conducción sin colisiones, no es capaz de alcanzar más de 100 Km/h en ninguno de los circuitos propuestos, perjudicando gravemente el tiempo por vuelta.

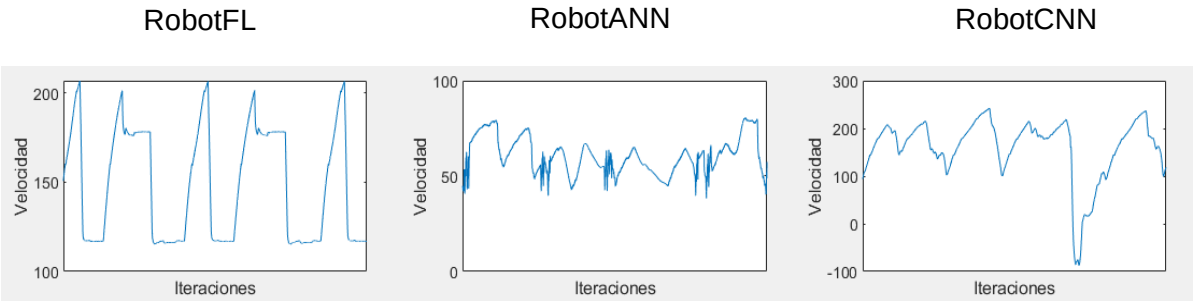

Figura 42. Representación de la variable speed para cada uno de los controladores.

En figura 42, donde se hace una comparativa de las velocidades alcanzadas en el transcurso de 3 vueltas en el circuito 3, se observa que en efecto, la velocidad del robot que implementa ANN se mantiene alrededor de 50 Km/hora con cambios muy bruscos de velocidad, mientras que el de sus competidoras presentan un mayor rango, alcanzando velocidades que superan los 200 Km/h. Cabe recordar que RobotCNN no implementa un control de velocidad basado en CNN, lo que no se debe de tener en cuenta para este análisis.

Se puede observar de igual manera, el efecto que tienen los diferentes algoritmos aplicados en la salida de velocidad del coche. Mientras que tanto ANN como CNN presentan una inconsistencia mayor en las velocidades, RobotFL muestra patrones mucho más ordenados, en donde se pueden ver los diferentes niveles de velocidad que se han definido mediante reglas (120 km/h velocidad baja, 180km/h velocidad y 260 km/h velocidad alta). Esta diferencia surge de las distintas formas de implantación de los modelos. Mientras que en RobotFL se fija una velocidad específica a la que ir según las variables de entrada, en RobotCNN y RobotANN se ataca directamente al acelerador y a los frenos sin establecer una velocidad en concreto. En RobotCNN, también se puede apreciar una bajada de velocidad drástica que llega incluso a obtener una velocidad negativa. Esta velocidad negativa está relacionada con una colisión, salida de pista y su correspondiente reincorporación a la misma.

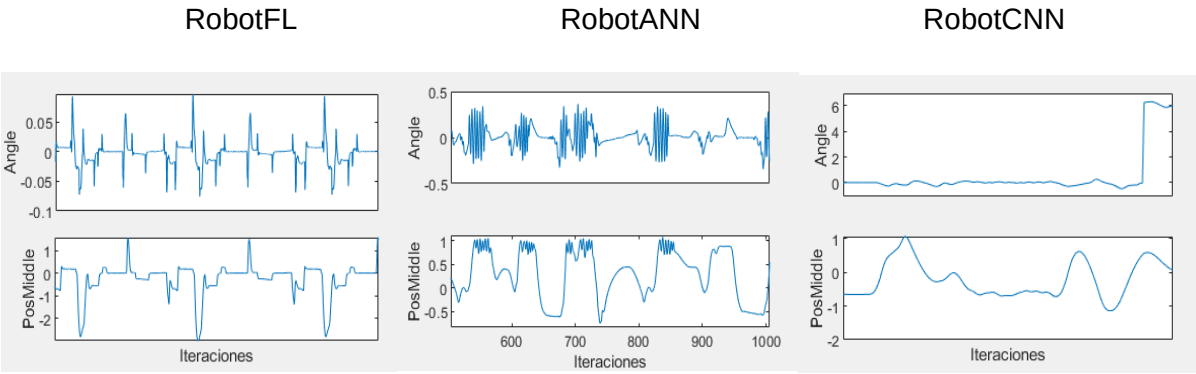

Figura 43. Representación de las variables angle y trackPos para cada uno de los controladores.

Con el fin de determinar cuál es el algoritmo que implementa la mejor trayectoria o conducción, se han realizado una serie de análisis, tanto gráficos como visuales. Primero, se ha visto el desempeño de todos los robots, teniendo en cuenta si hacían movimientos erráticos, oscilatorios, o si simplemente no se ajustaban a la trazada propuesta por la pista. También se han analizado y representado tanto las variables *trackPos* (distancia al centro de la carretera) y *angle* (desviación con respecto al ángulo central de la carretera) durante tres vueltas con el fin de obtener un análisis más en detalle (véase la figura 43).

RobotANN presenta una conducción aceptable, ya que no produce colisiones y se ajusta al interior de las curvas. Por el contrario, tiene un movimiento oscilatorio y constante en curvas que le hace moverse de derecha a izquierda para intentar encontrar la posición deseada (véase la figura 43). Debido a los datos a los que se ha sometido, RobotCNN muestra a simple vista una conducción parecida a la humana, sin oscilaciones ni rectificaciones muy buscas, pero presenta una mayor impredecibilidad que causa que en la mayoría de curvas rápidas tenga una alta probabilidad de colisión. La gráfica obtenida para la variable ángulo para este robot no es muy representativa, ya que los valores que produce cuando el robot está fuera de la pista no son fiables. En cuanto a la gráfica de la posición, vemos cambios leves que intentan ajustarse a la trazada de una manera más suave que la implementada por RobotANN. El alto gasto computacional ha podido jugar un papel importante en las colisiones, ya que como se explicó anteriormente, el software de competición recibe información cada 30 ms, y si esa información no está disponible, el robot toma la acción anterior. De esta manera, si el tiempo de procesar la imagen de entrada y convertirlo en una acción para el volante es muy elevado, es posible que en curvas donde sea necesario reaccionar rápido, se produzca una colisión.

Por último, RobotFL presenta la conducción más correcta de todas. Aunque, debido a cómo está definida intenta trazar las curvas por el centro de la pista, en curvas rápidas es despedido

hacia el exterior de la curva hasta que vuelve a retomar el centro. Tal y como se muestra en la figura 43, no presenta grandes valores de las variables *angle* y *trackPos* y como ya sucedió en la gráfica de la velocidad, es el único algoritmo que actúa de igual manera en todas las vueltas, realizando de forma continua el mismo tiempo por vuelta. Los valores reducidos de las variables angle y trackPos, hacen que la conducción no presente ni grandes oscilaciones ni cambios bruscos, evitando que se produzcan colisiones.

Tanto Robot ANN como RobotCNN, han sido entrenados con datos. Estos datos producen que la red sea capaz de aprender los patrones comunes de cada conducción. La principal diferencia de ambas redes es que, mientras que los datos de con los que ha sido entrenada la ANN han sido extraídos de otro Robot que implementaba otro algoritmo de conducción automático, los datos de la red de CNN han sido generados por una conducción humana. Esto provoca que los patrones de esta última no sean tan reconocibles (ya que una conducción humana puede ser más errática y no presenta siempre las mismas reacciones a los mismos estímulos) pudiendo afectar al entrenamiento de la red propiciando mayores colisiones.

Por todo lo anterior, se le atribuye la mejor conducción al robot que implementa Lógica Borrosa, seguido del que implementa ANN y por último, el de CNN (con 3, 2 y 1 punto respectivamente, tal que como se muestra en la tabla 8).

Se puede decir que tal y como está diseñado el experimento, y teniendo en cuenta que esta valoración está condicionada al objetivo por conseguir de cada uno, el robot que implementa lógica borrosa ha demostrado tener unas mejores prestaciones en conjunto ya que, aunque no tenga el mejor paso por vuelta, a diferencia de sus competidores, no presenta ningún punto de gran debilidad.

9. Conclusiones y trabajo futuro

Se presenta en este apartado las conclusiones asociadas a todo el trabajo previo realizado, así como se enumeran una serie de propuestas para continuar con el proyecto.

9.1. Conclusiones

En este trabajo de fin de máster, se ha realizado por primera vez una comparativa de distinto algoritmos de inteligencia artificial implementados en el simulador de coches TORCS con la finalidad de resolver problema de la conducción autónoma. Estos algoritmos han sido las CNN, las ANN y la Lógica borrosa.

Se ha conseguido el objetivo de realizar todo el proceso de desarrollo, optimización e implementación de los algoritmos para comparar no solo los resultados obtenidos, sino también las ventajas y desventajas de la implantación de cada uno de ellos en este caso concreto. A través de la experimentación y optimización de los algoritmos, se ha concluido que uno de los factores que produce las mayores diferencias en cuanto a conducción, además del cambio de los diferentes algoritmos, es el de su propia optimización. Algunos de estos parámetros más determinantes son la definición de reglas y funciones de pertenencia en la lógica borrosa o la definición de la arquitectura, entrenamiento, recolección y tratamiento de datos tanto para las ANN como las CNN.

Por último, se ha realizado un análisis exhaustivo de las variables más representativas relacionadas con la conducción, para así obtener conclusiones sobre cada uno de los algoritmos presentados.

Aunque los experimentos propuestos han demostrado que, en un entorno cerrado y controlado, la lógica borrosa es el algoritmo que ha presentado los mejores resultados en conjunto, no se ha podido dar una respuesta rotunda a la pregunta de cuál es el mejor algoritmo de inteligencia artificial para la resolución del problema de conducción autónoma. Esto se debe a que la elección de la mejor técnica depende de cuál sea el objetivo que se quiera maximizar dentro del problema (rapidez, conducción, fiabilidad etc.).

9.2. Líneas de trabajo futuro

Una futura investigación puede incluir en una primera instancia, una optimización exhaustiva de cada uno de los algoritmos propuestos con el fin de llegar al límite de sus capacidades.

También se pueden aplicar otros algoritmos de inteligencia artificial, como pueden ser el aprendizaje por refuerzo, o se pueden añadir a los algoritmos ya implementados, otros que hacen que destaquen sus características. Este podría ser el caso de combinar CNN con redes neuronales recurrentes para crear funciones que almacenan variables a lo largo del tiempo, o ANN y lógica borrosa con algoritmos genéticos para la optimización de hiperparámetros.

Otra propuesta adicional sería el incremento de la variabilidad y por ende, de dificultad del entorno. De esta manera, en vez de probar esta herramienta en un circuito cerrado, se podría plantear la inclusión de señales de tráfico, peatones y demás parámetros que puedan producir una modificación en la conducción autónoma.

Una vez incluidas estas propuestas, el siguiente paso podría ser implementar los algoritmos fuera de un simulador, utilizando datos de entornos reales. La aproximación a este acercamiento para el algoritmo de CNN, podría ser el utilizar un conjunto de datos que reproduce de manera realista el entorno. Utilizando el *dataset* de KITTI (Geiger, Lenz, Stiller, & Urtasun, 2013), orientado a ayudar con las tareas de conducción autónoma, conseguimos la visión de una cámara frontal de un automóvil que puede utilizarse como entrada para este tipo de algoritmos, además de numerosos sensores adicionales y detecciones que nos proporciona el propio *dataset*.

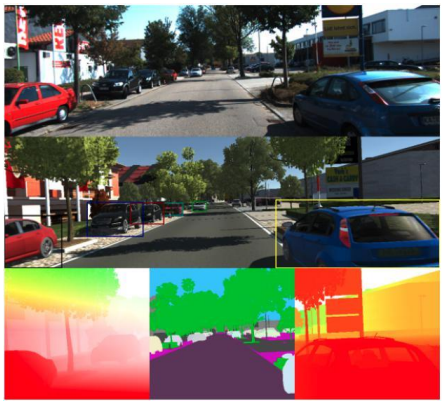

Figura 44. KITTI dataset. Fuente: (Geiger, Lenz, Stiller, & Urtasun, 2013)

Para el resto de los algoritmos, no valdría solo con un sensor como en el caso de las redes CNN con las cámaras. Para estimar variables de entrada la posición central, ángulo respecto a la carretera o distancia a la siguiente curva, se necesitaría otro número más elevado de sensores que complican su implantación en entornos reales.

Bibliografía

Abraham, A. (2005). Artificial neural networks. *Handbook of Measuring System Design.*

Albelihi, K., & Vrajitoru, D. (2015). An application of neural networks to an autonomous car driver. Paper presented at the *Proceedings on the International Conference on Artificial Intelligence (ICAI),* 716.

Almeida, B. C., & de Castro, Paulo André Lima. (2019). Autonomous driving on a direct perception system with deep recurrent layers. Paper presented at the *Proceedings of the 2nd International Conference on Applications of Intelligent Systems,* 1-6.

Ande Gaetan, Beelitz Wolf-Dieter, Xavier Bertaux, Eckhard M. Jaeger, Kristof Kaly-Kullai, Gabor Kmetyko, Enrico Mattea, Haruna Say, Joe Thompson, and Simon Wood. (2013). Speed dreams v2.0. Retrieved from http://www.speed-dreams.org

Armagan, E., & Kumbasar, T. (2018). A fuzzy logic based intelligent autonomous vehicle control system design in the TORCS game environment. *Mühendislik Bilimleri Dergisi, 24*(8), 1435-1442. doi:10.5505/pajes.2018.77910

Athanasiadis, C., Galanopoulos, D., & Tefas, A. (2012). Progressive neural network training for the open racing car simulator. Paper presented at the *2012 IEEE Conference on Computational Intelligence and Games (CIG),* 116-123.

Benoit, A., Bonnaud, L., Caplier, A., Ngo, P., Lawson, L., Trevisan, D. G., . . . Chanel, G. (2009). Multimodal focus attention and stress detection and feedback in an augmented driver simulator. *Personal and Ubiquitous Computing, 13*(1), 33-41.

Bernhard Wymann. (2013). The torcs endurance world championship. Retrieved from http://www.berniw.org/trb/.

Bogoni, T. N., & Pinho, M. S. (2012). Use of a simulator to assess the application of economic driving techniques by truck drivers. Paper presented at the *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC),* 3020-3026.

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

Cardamone, L., Loiacono, D., & Lanzi, P. L. (2011). Interactive evolution for the procedural generation of tracks in a high-end racing game. Paper presented at the *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation,* 395-402.

Chen, C., Seff, A., Kornhauser, A., & Xiao, J. (2015). Deepdriving: Learning affordance for direct perception in autonomous driving. Paper presented at the *Proceedings of the IEEE International Conference on Computer Vision,* 2722-2730.

Drolia, U., Wang, Z., Pant, Y., & Mangharam, R. (2011). AutoPlug: An automotive test-bed for electronic controller unit testing and verification. Paper presented at the *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC),* 1187-1192.

Forsyth, D., & Ponce, J. (2011). *Computer vision: A modern approach.* Prentice hall.

Fujii, S., Nakashima, T., & Ishibuchi, H. (2008). A study on constructing fuzzy systems for high-level decision making in a car racing game. Paper presented at the *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence),* 2299-2306.

Gamal, O., Imran, M., Roth, H., & Wahrburg, J. (2020). Assistive parking systems knowledge transfer to end-to-end deep learning for autonomous parking. Paper presented at the *2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE),* 216-221.

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research, 32*(11), 1231-1237.

Göhring, D., Wang, M., Schnürmacher, M., & Ganjineh, T. (2011). Radar/lidar sensor fusion for car-following on highways. Paper presented at the *The 5th International Conference on Automation, Robotics and Applications,* 407-412.

Goldsborough, P. (2016). A tour of tensorflow. *arXiv Preprint arXiv:1610.01178.*

Hassaballah, M., & Awad, A. I. (2020). *Deep learning in computer vision: Principles and applications* CRC Press.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Paper presented at the *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 770-778.

Heaton, J. (2015). Encog: Library of interchangeable machine learning models for java and C#. *J.Mach.Learn.Res., 16*, 1243-1247.

Hecht, J. (2018). Lidar for self-driving cars. *Optics and Photonics News, 29*(1), 26-33.

Ho, D. T., & Garibaldi, J. M. (2008). A novel fuzzy inferencing methodology for simulated car racing. Paper presented at the *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence),* 1907-1914.

Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. Paper presented at the *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 7132-7141.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., . . . Wu, Y. (2019). Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in Neural Information Processing Systems, 32*, 103-112.

Jin, Y., Lee, S., Sung, Y., & Cho, K. (2017). A learning and testing system for self-driving using CNN on TORCS. Paper presented at the *Proceedings of the Korea Information Processing Society Conference,* 839-841.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237-285.

Karavolos, D. (2013). Q-learning with heuristic exploration in simulated car racing. *Published Thesis, University of Amsterdam.*

Keith Curtis. (2013). Pytorcs. Retrieved from https://github.com/KeithCu/PyTorcs.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*, 1097-1105.

Loiacono, D., Cardamone, L., & Lanzi, P. L. (2013). Simulated car racing championship: Competition software manual. *arXiv Preprint arXiv:1304.1672.*

Manrique Gamo, D., & Suárez de Figueroa Baonza, María del Carmen. (2015). *Razonamiento con imprecisión:Lógica borrosa.*

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics, 5*(4), 115-133.

McNeill, D. K. (2018). Training RNN simulated vehicle controllers using the SVD and evolutionary algorithms. Paper presented at the *2018 IEEE Intelligent Vehicles Symposium (IV),* 1949-1953.

Moliner, R., & Tanda, R. (2016). Herramienta para la sintonía robusta de controladores PI/PID de dos grados de libertad. *Revista Iberoamericana De Automática E Informática Industrial, 13*(1), 22-31.

Munoz, J., Gutierrez, G., & Sanchis, A. (2009). Controller for torcs created by imitation. Paper presented at the *2009 IEEE Symposium on Computational Intelligence and Games,* 271-278.

Onieva, E., Pelta, D. A., Alonso, J., Milanés, V., & Pérez, J. (2009). A modular parametric architecture for the torcs racing engine. Paper presented at the *2009 IEEE Symposium on Computational Intelligence and Games,* 256-262.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv Preprint arXiv:1511.08458,*

Perez, D., Recio, G., Saez, Y., & Isasi, P. (2009). Evolving a fuzzy controller for a car racing competition. Paper presented at the *2009 IEEE Symposium on Computational Intelligence and Games,* 263-270.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*(6), 386.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533-536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., . . . Bernstein, M. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision, 115*(3), 211-252.

Salem, M., Mora, A. M., Merelo, J. J., & García-Sánchez, P. (2017). Driving in TORCS using modular fuzzy controllers. Paper presented at the *European Conference on the Applications of Evolutionary Computation,* 361-376.

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena, 404*, 132306.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv Preprint arXiv:1409.1556,*

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation, 10*(2), 99-127.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. Paper presented at the *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 1-9.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. Paper presented at the *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2818-2826.

Szeliski, R. (2010). *Computer vision: Algorithms and applications* Springer Science & Business Media.

Talpaert, V., Sobh, I., Kiran, B. R., Mannion, P., Yogamani, S., El-Sallab, A., & Perez, P. (2019). Exploring applications of deep reinforcement learning for real-world autonomous driving systems. *arXiv Preprint arXiv:1901.01536,*

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. Paper presented at the *International Conference on Machine Learning,* 6105-6114.

Tran, D., Do, H. M., Lu, J., & Sheng, W. (2020). Real-time detection of distracted driving using dual cameras. Paper presented at the *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* 2014-2019.

Vashishtha, S., & Verma, S. (2020). Restoring chaos using deep reinforcement learning. *Chaos: An Interdisciplinary Journal of Nonlinear Science, 30*(3), 031102.

Wang, Y., Liu, D., Jeon, H., Chu, Z., & Matson, E. T. (2019). End-to-end learning approach for autonomous driving: A convolutional neural network model. Paper presented at the *Icaart (2),* 833-839.

Wymann, B., Espié, E., Guionneau, C., Dimitrakakis, C., Coulom, R., & Sumner, A. (2000). Torcs, the open racing car simulator. *Software Available at Http://Torcs.Sourceforge.Net, 4*(6), 2.

Anexos

Artículo de investigación

Título

Mario Hernández García

Universidad Internacional de la Rioja, Logroño (España)

15/09/2021

Resumen

En este trabajo de fin de máster, se propone una comparativa entre diferentes algoritmos de inteligencia artificial destinados a resolver el problema de la conducción autónoma, implantados en el simulador de carreras TORCS. Los algoritmos involucrados en esta comparativa son: la lógica borrosa (FL), las redes neuronales artificiales (ANN) y las redes neuronales convolucionales (CNN).

Se ha seguido todo proceso de desarrollo y optimización de los algoritmos propuestos, además de la implementación en el simulador con la finalidad de los resultados obtenidos, ventajas y desventajas de la implantación de cada uno de ellos.

Por último, se ha realizado un análisis exhaustivo de los distintos controladores atendiendo a su desempeño en entorno cerrado y controlado. Aunque los experimentos propuestos han demostrado que la lógica borrosa es el algoritmo que presenta los mejores resultados en el experimento realizado, no se ha podido determinar el mejor algoritmo de inteligencia artificial para la resolución del problema de conducción autónoma, ya que influye en gran medida tanto el entorno de pruebas como la optimización de los algoritmos en la fase de desarrollo.

Palabras Clave

Inteligencia Artificial, conducción autónoma, lógica borrosa, red neuronal, TORCS.

I. Introducción

La condición autónoma tiene un gran potencial para cambiar la manera en la que vivimos. Este paradigma tiene la capacidad de reducir problemas relacionados con la conducción manual, como puede ser el peligro que supone el uso de un automóvil por parte de un conductor ebrio o con somnolencia.

El auge de la inteligencia artificial, propiciado por sus grandes resultados en diversos ámbitos, hace que el desarrollo de este tipo de técnicas sea cada vez más común para resolver gran variedad de problemas, siendo el de la conducción autónoma uno de ellos.

En términos generales, el objetivo de este trabajo es implantar distintos algoritmos de inteligencia artificial en un simulador de coches y determinar qué ventajas y desventajas tiene cada uno y cuál es capaz de realizar una mejor conducción, teniendo en cuenta aspectos como el trazado (que no produzca una conducción errática), el tiempo por vuelta o si se produce alguna salida de pista.

La estructura del trabajo consta de nueve bloques diferenciados:

- **Contexto y estado del arte:** Se hará una introducción teórica para aclarar conceptos de cada uno de los algoritmos de inteligencia artificial aplicados en la comparativa. Además, se realizará una breve contextualización de la situación actual de cada uno de los algoritmos de inteligencia artificial involucrados tanto de manera general (últimos avances y aplicaciones) como las aportaciones recientes en el problema de la conducción autónoma.
- **Objetivos concretos y metodología de trabajo:** Se describen una serie de objetivos a cumplir tanto a nivel general como específico, además de un plan de desarrollo para completar dichos objetivos.
- **Planteamiento de la comparativa:** Se propone un marco común de pruebas, de tal manera que todos los algoritmos partan de las mismas condiciones (mismos coches, mimos circuitos de pruebas) para así poder hacer una comparación justa.
- **Preparación del trabajo:** En este capítulo, se sigue todo el proceso de instalación y creación de los Robots que van a suponer una estructura donde implementar los controladores.
- Desarrollo del controlador de **lógica borrosa.**
- Desarrollo del controlador de **ANN.**
- Desarrollo del controlador de **CNN.**
- **Discusión y análisis de resultados:** A raíz del desarrollo propuesto en el apartado anterior, se realizará una comparativa exhaustiva de cada uno de los algoritmos propuestos. La información se propondrá en forma de tablas para que se presente la información útil de forma clara y concisa.
- **Conclusiones y trabajos futuros:** En este apartado se responderá a si se cumplieron los objetivos tanto generales como específicos propuestos. Además, se extraerán conclusiones de la información aportada en los resultados de la comparativa, intentando determinar si existe un mejor algoritmo para este contexto de aplicación o si, por el contrario, existen tanto ventajas como desventajas para cada uno de los modelos de conducción autónoma propuestos. Por último, se desarrollará de manera teórica una forma de realizar un desarrollo posterior al trabajo de fin de máster y se planteará de manera teórica la forma de implementar los desarrollos realizados en un coche real

.II. Estado del Arte

Se describe el contexto de aplicación del software necesario para realizar el trabajo, centrándolo en el simulador de carreras TORCS. También se estudiará más en profundidad el marco teórico y la situación del desarrollo existente de los algoritmos propuestos en la comparativa. Estos son el de lógica borrosa, las ANN y por último, las CNN.

TORCS

La implantación de los diferentes algoritmos de inteligencia artificial se hará en el simulador de carreas 3D de código abierto (TORCS, por sus siglas en inglés *The Open Racing Car Simulator*) [21]. Se trata de simulador modular, multiagente y multijugador fundado en 2007 por Guionneau Christophe y Eric Espie.

Gracias a su modularidad, a su simplicidad en el código base y a su estabilidad entre otras características, es ideal para investigaciones industriales y científicas, como lo abala el hecho de la cantidad de investigaciones de distinta índole utilizando este software.

En TORCS, los conductores tanto reales como NPCs (*Non Playable Characters*) son denominados robots. Estos robots son cargados de manera externa en el juego, de forma que se pueda introducir nuevos módulos de robots para en este caso, implementar soluciones de inteligencia artificial de una manera estructurada.

Existen más de 300 artículos académicos que hayan empleado TORCS como base, utilizando algoritmos de inteligencia artificial. En cualquier caso, TORCS ha encontrado otros usos más allá de la implementación en el simulador. Por ejemplo, ha sido usado como banco de pruebas para las unidades electrónicas de los automóviles [5]. También se han realizado estudios de control de la atención y del estrés de un conductor al volante [2] y para desarrollar un controlador económico de conducción autónoma para tractores [3].

Lógica borrosa

El primer algoritmo de inteligencia artificial aplicado es el llamado lógica borrosa, que se define como una disciplina que permite representar el conocimiento acerca de un dominio y realizar procesos de inferencia o razonamiento, proporcionando un lenguaje formal de representación del conocimiento basado en la lógica y las matemáticas [12].

El elemento más importante de este tipo de algoritmos es la base de conocimiento o *rule base*, que contiene la información de cómo controlar el sistema, en forma de un conjunto de reglas. Asocia unas entradas con unas salidas con asociaciones lingüísticas no cuantitativas.

También presenta unos mecanismos de inferencia o *inference mechanism*, que son los métodos con los que se evalúan las reglas de control que son relevantes en un momento especifico, y establece la salida.

Por último, existen dos elementos o procesos contrarios entre sí, uno la interfaz de borrosificación y otra la de desborrosificación. El proceso de borrosificación hace que las variables de entrada sean entendibles según nuestra base de conocimiento asociando etiquetas lingüísticas a valores numéricos. Una vez realizada la inferencia y de manera opuesta a la borrosificación, la interfaz de desborrosificación se coloca a la salida del sistema para dar una salida numérica a las etiquetas lingüísticas asociadas, de tal manera que puedan ser entendibles por la planta.

La lógica borrosa ha sido utilizada de manera prolífica para abordar el problema de la conducción autónoma. Esta técnica ha sido implementada en un principio con mayor frecuencia que otros algoritmos de inteligencia artificial debido a que no es necesario tener una capacidad computacional elevada si se compara con el coste de la implementación de ANN y CNN y sus respectivos procesos de entrenamiento.

Por ejemplo, se ha examinado el desempeño de un sistema borroso basado en reglas de decisiones de alto nivel en un controlador, transformando la información de entrada proporcionada por los sensores [7]. También se ha propuesto el diseño de un controlador borroso para calcular la posición deseada del coche, aunque no se usó para provocar el movimiento del volante [14]. También se ha desarrollado la implantación de dos controladores borrosos que funcionan de manera simultánea pero independiente, unos para el control de la velocidad, y otro para controlar la dirección del volante [17]. Cogiendo de referencia esta última solución, se realizará el desarrollo del control borroso del coche. Con la información aportada por investigaciones anteriores, se hará un ajuste más preciso y acorde a los sensores de entrada de los que se dispone. Recientemente, se han propuesto otros sistemas híbridos que utilizan técnicas de machine learning con lógica borrosa [10], o lógica borrosa con algoritmos genéticos [15] para la optimización de hiperparámetros.

Redes neuronales artificiales

El siguiente algoritmo de inteligencia artificial a implantar son las ANN. Estas redes tienen la finalidad de aproximar cualquier función matemática. Son especialmente útiles cuando el proceso que relaciona las entradas con las salidas es complejo o presenta ruido (perturbaciones de alta frecuencia que interfieren sobre una tendencia estable), ya que es capaz de computar grandes cantidades de variables de entrada para modelar una salida.

Existen numerosos casos de implantación de este tipo de algoritmo en el simulador de coches TORCS. Para realizar la fase de entrenamiento del automóvil con esta técnica, se necesitan datos que permitirán a la red neuronal ajustar sus pesos para tener un mejor desempeño en este problema. Este algoritmo recibe como entrada datos proporcionados por el propio simulador, tanto de datos referentes a los parámetros internos del vehículo (velocidad, inercia, aceleración) como en relación con la pista (ángulo, posición respecto al centro de la pista). Gracias a esos datos de entrada y a la información de salida (giro del volante, acción de acelerar, acción de frenar), son capaces de aprender esos conocimientos y ponerlos en práctica en circuitos donde no han sido entrenados. Para la recolección de estos datos, se ha hecho uso de los robots que están previamente implementados en el simulador TORCS por defecto [1], mientras que [13], los datos se extraen de un automóvil conducido por una persona, ya que su propósito de esta investigación es el de diseñar un robot que sea capaz de imitar en cualquier pista la conducción humana.

Un caso de combinación de distintos algoritmos de inteligencia artificial se encuentra en las llamadas redes NEAT (*NeuroEvolution of Augmenting Topologies*). Este tipo de redes puede optimizar y evolucionar la estructura entera de una red neuronal [18] para producir unos resultados que minimicen la función de coste. En el campo de la conducción autónoma, han sido implementados junto a las ANN en [13].

Redes neuronales convolucionales

El último algoritmo a desarrollar e implantar en este proyecto son las CNN. Este algoritmo constituye el estado del arte para muchas de las tareas de visión artificial (VA) por su capacidad de extraer y aprender patrones espaciales (relaciones

entre los píxeles que definen una imagen).

Gracias a sus grandes prestaciones dentro de este campo, las CNN se han utilizado para abordar numerosos problemas, como pueden ser la clasificación, la localización, la detección, la identificación, la segmentación y el seguimiento de objetos. La clasificación categoriza una imagen en función de un conjunto de etiquetas predefinidas. La localización ubica un objeto que se sabe dentro de la imagen. La detección ubica un objeto en una imagen si este está presente. La segmentación particiona la imagen en segmentos (o áreas) que delimitan objetos. Por último, el seguimiento, se encarga de ubicar y mantener un único identificador por cada objeto mostrado [6].

Referente al estado del arte de las CNN aplicado a resolver el problema de la conducción autónoma, se han propuesto numerosos casos de cámaras, tanto frontales y traseras de asistencia al conductor. Algunos ejemplos de esta aplicación puede ser la ayuda al aparcamiento [8] o la observación del conductor para detectar patrones de distracción de la carretera [19]. Adicionalmente, se han desarrollado algunos sistemas que implementan esta tecnología para la propia conducción del automóvil para el simulador de carreras TORCS.

El mayor avance en el uso de CNN para la conducción autónoma implementado en el simulador TORCS viene dado por la universidad de Princeton con el proyecto nombrado DeepDriving [4]. En este proyecto, se utilizan pistas modificadas con líneas divisorias que ayudan al automóvil a predecir tanto la dirección de la pista con mayor exactitud. Se propone una solución tanto para la conducción autónoma, como para la detección de otros automóviles para evitar posibles colisiones por una técnica que nombran como método directo.

La menor cantidad de recursos encontrados para esta aplicación frente a otros algoritmos como pueden ser las ANN o la lógica borrosa, puede estar dada por la gran capacidad de cómputo necesaria para llevar a cabo la tarea de entrenar una red de este tipo, desde la recolección de datos (imágenes) y su almacenamiento, hasta el proceso de entrenamiento con esos datos. En cualquier caso, el avance que se está dando dentro del mundo tecnológico y en concreto en las tarjetas gráficas o GPU (*Graphics Processing Unit*), necesarias para todo el proceso con redes neuronales, está permitiendo una mayor accesibilidad de estos sistemas y, por lo tanto, se espera un aumento del número de investigaciones sobre este tema.

III. Objetivos y Metodología

El objetivo principal del trabajo de fin de máster es mostrar una comparativa de manera extensa y detallada de cuál es la técnica o algoritmo de inteligencia artificial que más se adecúa al problema de la conducción autónoma. Los algoritmos desarrollados se probarán en el simulador de coches de código abierto TORCS por su estructura modular y recursos de investigación disponibles. Para concluir, también plantearemos una manera en la que se puedan trasladar los conocimientos adquiridos y los resultados obtenidos en el simulador, a la conducción de un automóvil real.

Para conseguir el objetivo principal de la comparación y determinación de las ventajas y desventajas de cada una de las técnicas de inteligencia artificial aplicadas, definimos una serie de objetivos parciales o específicos:

- Desarrollar en profundidad cada una de las técnicas o algoritmos de inteligencia artificial a implementar elegidos, proporcionando la configuración óptima de ese algoritmo ajustándose al problema propuesto.
- Evaluar los algoritmos aplicados en 3 circuitos diferentes, manteniendo las mismas condiciones para cada una de las pruebas hechas en los circuitos.

De cara a alcanzar los objetivos específicos y con ellos, el objetivo general, definimos una metodología de trabajo, en la que se describen paso a paso las acciones a seguir, y el orden en el tiempo de estas.

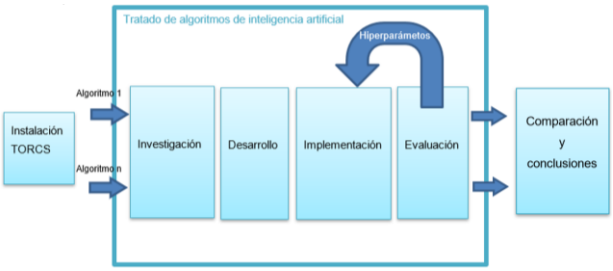

Fig. 1. Metodología de trabajo

En la figura 1, se muestra un diagrama que pretende ser una guía visual de los pasos a seguir. Se empezará con la instalación del simulador de coches TORCS en el sistema operativo Linux dentro de una máquina virtual. Esto es debido a que la mayoría de las investigaciones sobre este software están en este sistema operativo, además de que también existe la mayor comunidad de resolución de problemas derivados de su instalación y modificación. Una vez instalado el software y todas sus dependencias necesarias, se escogerá una técnica o algoritmo de inteligencia artificial para ver si es apta su implementación. Si determinamos que, en efecto, ese algoritmo puede ser útil para el problema de la conducción autónoma, se plantean una serie de pasos a alto nivel que van a compartir todos los algoritmos que se vayan a probar en el simulador.

El primer paso describe la investigación sobre el algoritmo, donde determinaremos el funcionamiento de este y los recursos necesarios que se pueden buscar en la web (por ejemplo, un dataset público con imágenes para la implantación de redes neuronales convolucionales). Una vez completada la fase de investigación, estableceremos una fase de desarrollo de ese algoritmo. Para esta fase de desarrollo se buscará la manera de hacer uso de los recursos disponibles (uso de distintos lenguajes de programación, de distintos frameworks de aprendizaje automático o de software específico como Matlab). Hecho un modelo primigenio tras una fase de implementación dentro del simulador, se evaluará su comportamiento e intentaremos ajustar los hiperparámetros de la red dependiendo del algoritmo utilizado (modificación del dataset en la redes neuronales o cambio de las funciones de pertenencia en el controlador borroso) de manera que busquemos la mejor solución teniendo en cuenta la persecución de los objetivos del trabajo. Tras haber repetido el proceso de ajuste de hiperparámetros hasta que se haya encontrado una solución óptima, se hará una comparación de todos los resultados de todos los algoritmos en una tabla, de manera que se pueda extraer conclusiones, si hay algún algoritmo que sea claramente mejor que otro, teniendo en cuenta los criterios de evaluación o si, por el contrario, cada algoritmo tiene ventajas y desventajas respecto al resto.

IV. Contribución

Diseño del controlador borroso

Se decide diseñar dos controladores para este apartado de lógica borrosa, un controlador de volante y otro de velocidad. Ambos tendrán que trabajar de manera conjunta para el correcto funcionamiento del automóvil, pero la información de entrada que tienen en cuenta para sus respectivas salidas puede ser diferente. Se podría haber definido solo una salida que controlase tanto el volante como el freno del coche, pero esto habría añadido mucha complejidad a la definición de las reglas y una de las mayores ventajas que tiene esta técnica es que la definición de

reglas permite crearlas de manera intuitiva.

Para llevar a cabo un diseño adecuado del controlador de volante que se implementará en el vehículo elegido, primero se seleccionan de las variables de entrada que son proporcionadas por simulador, cuáles se van a tener en cuenta en la entrada para producir una salida.

Se define que las variables de entrada sean:

Error angular del vehículo (*angle*): representa el ángulo entre la dirección del eje mayor del coche y el eje central del segmento en el que se encuentra el coche.

Error de posición lateral con respecto al centro de la pista (*trackPos*). representa la distancia entre el centro del coche y el eje central del segmento en el que se encuentra el coche.

En el caso de la variable de salida, se define el giro del volante que deberá llevar a cabo nuestro vehículo con el fin de no salirse de la trazada. Esta salida, podrá variar entre -1 y 1, siendo cada uno de estos valores equivalentes a un giro del volante de 180 º en sentido horario o antihorario.

Se establecen unas funciones de pertenencia y una base del conocimiento que van a determinar el funcionamiento del controlador del volante:

Reglas del controlador del volante		Error angular		
		Derecho	Centro	Izquierdo
Error Lateral	Derecho	Izda.	Dcha. Peq.	Dcha. Peq.
	Centro	Izda.	Centro	Dcha.
	Izquierdo	Izda. Peq.	Izda. Peq.	Dcha.

Tabla 1. Reglas del controlador del volante

Se puede apreciar como la variable del error lateral queda en un segundo plano respecto a la del error angular, que es la que permite al vehículo mantenerse recto en la pista. Si se tiene la posición angular del coche desplazada a la derecha, el coche girará siempre a la izquierda (en distintos grados) con independencia del error actual en el que se encuentre el coche.

Para llevar a cabo un diseño adecuado del controlador de velocidad que implementaremos en el coche seleccionado, primero hemos de seleccionar cuáles serán las variables de entrada, y cuál será la variable de salida.

Se decide seleccionar las siguientes variables como entrada del controlador:

Tiempo al siguiente tramo (*TimetonextRadius*). Calculada haciendo una división entre los metros al siguiente tramo y velocidad actual.

Curvatura del tramo actual (*CurrentRadius*) y Curvatura del siguiente tramo (*NextRadius*). Permiten saber si en una curva el vehículo está a la entrada o a la salida de la misma, viendo la relación entre la curvatura actual y la del siguiente tramo.

Con estas tres variables se es capaz de definir un conjunto de reglas que abarque todos los casos posibles encontrados en los circuitos a probar del simulador TORCS.

Para la salida se define la variable Velocidad con un rango de entre 0 y 300 Km\h, sin embargo, debido a las limitaciones físicas del vehículo, la velocidad máxima que alcanza será de aproximadamente 250 Km/h.

Las tablas 2 y 3 muestra la base del conocimiento o conjunto de reglas que determinan el funcionamiento de la velocidad.

Tiempo = Pequeño		Radio Siguiente			
		Recta	Pequeño	Mediano	Grande
Radio Actual	Recta	Alta	Baja	Alta	Alta
	Pequeño	Baja	Baja	Baja	Media
	Mediano	Media	Baja	Media	Alta
	Grande	Alta	Baja	Alta	Alta

Tabla 2. Reglas de control de la velocidad (parte 1)

Tiempo = Grande		Radio Siguiente			
		Recta	Pequeño	Mediano	Grande
Radio Actual	Recta	Alta	Alta	Alta	Alta
	Pequeño	Baja	Baja	Baja	Baja
	Mediano	Media	Media	Media	Media
	Grande	Alta	Alta	Alta	Alta

Tabla 3. Reglas de control de la velocidad (parte 2)

Con esta base de reglas definidas para el control de velocidad, se pueden apreciar dos paradigmas completamente diferenciados y que coinciden con cada una de las tablas propuestas. Cuando el tiempo a la siguiente curva es pequeño, se le da prioridad al radio siguiente, mientras que si el tiempo a la siguiente curva es grande, la prioridad pasa a ser el radio actual hasta acercarse a la próxima curva. Si por ejemplo el tiempo es grande y el radio actual es pequeño, la velocidad será siempre baja con independencia del valor que esté tomando el radio siguiente.

Diseño del controlador por redes neuronales artificiales

Los datos de los sensores y actuadores con lo que se va a tratar, son los proporcionados por Lex van Teeffelen en su repositorio, ya que presentan una gran cantidad de información de coches ya conducidos en distintos circuitos.

Se determina para este controlador cuáles van a componer los datos de entrada de la red neuronal. Las variables de entrada las componen: el ángulo, la posición de la pista y la velocidad al igual que en el controlador borroso y, adicionalmente, añadiremos los 19 sensores que componen la variable track. Cada uno de ellos indica la distancia existente desde el centro del coche hasta el extremo de la pista en ángulos de 10 grados hasta completar los 180 grados de la visión frontal del vehículo.

Tal y como se observa en la figura 2, las salidas de la red neuronal artificial, que a su vez serán actuadores del robot, son el pedal de freno, el de aceleración y el de volante. Tanto el embrague como el cambio de marchas no estarán influenciado por la red neuronal, dejando así su funcionamiento por defecto dependiendo del coche elegido.

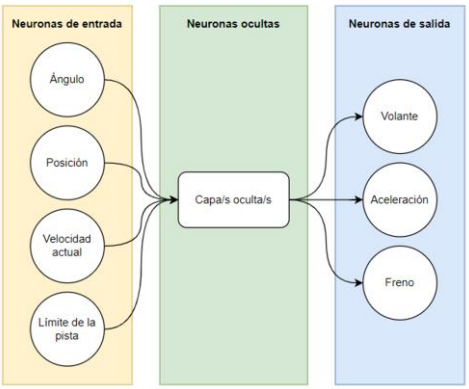

Fig. 2. Estructura de la red neuronal artificial.

Existe una gran variabilidad entre los rangos definidos para cada una de las variables de entrada. Esto puede hacer que se produzca un ajuste incorrecto de pesos, confundiendo valores altos de entrada con la importancia que la red asigna a la variable.

De esta manera, se tendrá que realizar una normalización de datos de entrada en la que la función de activación elegida puede tener un papel importante.

Para realizar el entrenamiento de la red propuesta, Se define una arquitectura común. A partir de esa arquitectura, podremos modificar una serie de hiperparámetros con los que se extraen la mejor combinación de estos. Se realizará un entrenamiento para cada una de las arquitecturas y se elegirá la red que menos error proporciona para los datos de test.

Dada la gran cantidad de posibilidades existentes de modificación de hiperparámetros de la red, se definirán unos parámetros fijos que afectarán tanto a la arquitectura de la red como al proceso de entrenamiento.

- La capa de entrada tiene las 22 neuronas que corresponden variables de entrada.
- La capa de salida tiene 3 salidas que corresponden con los actuadores del automóvil elegidos.
- Optimizador: *Resilient Back Propagation* (Rprop)
- El entrenamiento se hará por 5000 epochs o ciclos de entrenamiento.
- Habrá una, dos o ninguna capa oculta con el mismo número de neuronas que la capa de entrada.
- La función de activación para todas las capas será lineal, tangente hiperbólica o sigmoidea.

Por lo tanto, se variará tanto la función de activación y el número de capas ocultas (la profundidad de la red) manteniendo el número de neuronas de estas (anchura de la red). De esta manera, se comprueba qué grado de complejidad se ajusta más a los requisitos del sistema, evitando tanto el *underffiting* como el *overfitting* o sobreentrenamiento.

Estos son los resultados obtenidos después del entrenamiento:

Error de entrenamiento (MSE)		Número de capas ocultas		
		0	1	2
Función de activación	Linear	0,2088	0,2087	0,2088
	Tanh	0,1909	0,1547	**0,1386**
	Sigmoid	0,1742	0,1843	0,2654

Tabla 4. Resultado del entrenamiento con ANN

Después observar los valores de MSE (Mean Square Error) presentados en la tabla 4, se puede concluir que tanto el número de capas ocultas, como las funciones de activación, juegan un papel muy importante a la hora que buscar que la red neuronal artificial que pueda producir unos mejores controladores.

Se observa cómo, independientemente del número de capas ocultas que presente la red, la función de activación lineal es incapaz de ajustarse bien al espacio de características. La función sigmoidea no presenta tampoco los mejores resultados, incluso superando en error a la función lineal cuando la red presenta dos capas ocultas. Se observa un claro *overffiting* al aumentar la complejidad de la red, ya que esta complejidad hace que no sea capaz de aprender patrones de acción comunes, siendo muy reactivo a cualquier ruido que puedan presentar los datos de entrada. Por último, la función tangente hiperbólica es la función de activación que presenta unas mayores prestaciones, especialmente cuando aumentamos la profundidad de la red a 2 capas ocultas.

Por todas las conclusiones extraídas de los entrenamientos anteriores, escogemos la arquitectura que contienen 2 capas ocultas con la función de activación tangente hiperbólica para realizar la comparación de los distintos algoritmos de conducción autónoma.

Diseño del controlador por redes neuronales convolucionales

Para la implantación del algoritmo de CNN en el simulador, se empleará el método de reflejo del comportamiento para implementar el controlador del coche. Este método procesa directamente los datos de entrada (la imagen completa) para tomar una decisión de volante producida de principio a fin en solo un paso. Para aplicaciones donde solo existe interacción entre la pista sin obstáculos y el coche, este acercamiento es el más adecuado debido a su mayor eficiencia computacional y desempeño en ambientes poco concurridos.

Tal y como ocurría en las ANN, se precisan de datos para poder hacer un entrenamiento de la red. Se propone un modelo de recolección de datos (véase la figura 3) en el que se necesita por un lado la imagen, y por otro la salida del volante asociada a dicha imagen. Estos datos son los que posteriormente servirán como entrada para el entrenamiento de las CNN. La aplicación de los actuadores del acelerador y del freno como salida de la red convolucionales proporciona resultados muy pobres en su implementación en el simulador. Esto puede estar motivado por el hecho de no almacenar valores pasados de velocidad, por lo que, por una única imagen de entrada, la red no es capaz inferir la velocidad a la que entra el automóvil en una curva dada. Se podría plantear una estructura híbrida, en que se combina las CNN con las redes neuronales recurrentes (RNN). Esto trabajaría como con una función de memoria orientada a almacenar datos de velocidades pasadas y a partir de esos valores, hacer predicciones futuras. Por lo tanto, el control de velocidad, se aplicará con el algoritmo por defecto que incluye el automóvil utilizado.

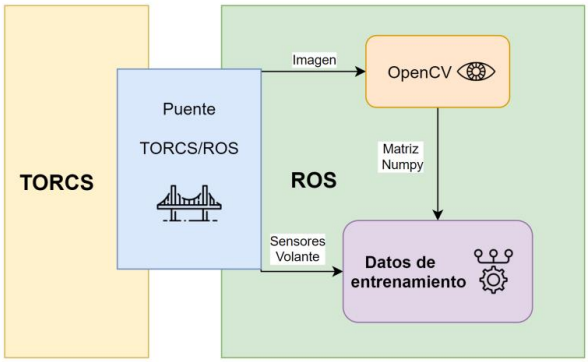

Fig. 3. Proceso de recolección de datos.

Este proceso de recolección de datos se hará en 3 circuitos distintos de los que posteriormente van a ser probados en la comparativa, ya que eso supondría en estos últimos un desempeño superior debido a la memorización que pudieran tener de los datos de entrada. Se utiliza para este propósito de recolección de datos, la conducción del vehículo de manera manual durante diez vueltas en cada uno de los circuitos. Se recogerán tanto datos de las imágenes capturas en primera persona como la posición del volante, con una frecuencia de imagen de 5 Hz a la resolución nativa de del simulador (480x640). Se asume que una frecuencia de muestreo mayor no aporta información adicional, ya que no representa una variación significativa sobre el fotograma anterior.

Con el fin de acelerar el proceso de entrenamiento, las imágenes obtenidas son primero recortadas y luego reescaladas, obteniendo una imagen final con resolución 66x200 a color. El recorte de la imagen también está orientado a reducir parte del

entorno que no aporta información significativa para el desempeño de la conducción y que además pueden entorpecer la labor de entrenamiento de la red.

Como una primera aproximación a la creación de distintas CNN, se utilizan las redes conocidas comúnmente como AlexNet [11] y NvidiaNet [20]. La elección de estas arquitecturas es debida a que ambas redes han sido propuestas anteriormente para soluciones de conducción autónoma en el campo de la visión artificial.

Se establece un marco común de definición de hiperparámetros, los cuales serán los utilizados para la realización del proceso de entrenamiento.

- Tamaño de imagen de entrada: 66x200x3 píxeles
- Normalización L2: 0,001
- Learning Rate: 0,005
- Iteraciones (epochs): 2000
- Tamaño del lote (batch size): 20 imágenes/lote
- 80% de datos reservados para entrenamiento y 20% para validación.
- Optimizador: Adam
- Métrica de error: Error medio cuadrático (MSE)

Se obtienen los siguientes resultados:

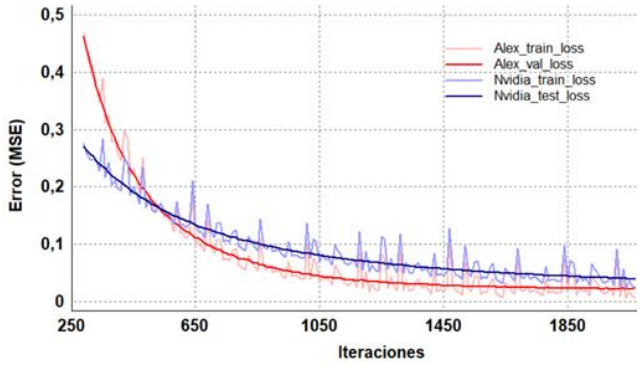

Fig. 4. Proceso de entrenamiento con CNN.

Se observa en la figura 4 que para el entrenamiento en 2000 iteraciones y con el resto de hiperparámetros descritos anteriormente, la red conocida como AlexNet presenta mejores resultados de NvidiaNet, a pesar de ser una red más longeva y no estar diseñada de manera específica para el problema de la conducción autónoma, es por eso que se utilizará AlexNet para la comparativa posterior de los distintos algoritmos de inteligencia artificial.

V. Resultados

Una vez realizada la implantación de cada uno de los algoritmos de inteligencia artificial en el simulador, se extraen una serie de concusiones descritas a continuación:

- Los algoritmos que implementan lógica borrosa necesitan una persona "experta" que entienda el funcionamiento del sistema para poder definir unas reglas que se obtienen de la experiencia. Sin embargo, las aplicaciones con CNN y ANN pueden funcionar como "cajas negras" en donde a partir de unas entradas, se producen unas salidas que modelan una función matemática que puede ser desconocida.
- Tanto los algoritmos de ANN como de CNN precisan de gran cantidad de datos para funcionar, ya que a través de estos son capaces de aprender patrones comunes de comportamiento para su propia red. Si los datos están corruptos, son insuficientes, no están bien normalizados o alguna categoría o valor está sub-representado o sobre-representado, se puede producir un incorrecto funcionamiento de este tipo de algoritmos.
- El hecho de que se tenga que definir una regla para cubrir todos los casos en lógica borrosa, hace que sea inviable plantear este problema cuando el número de variables tanto de entrada como de salida es elevado, ya que por cada variable, la complejidad de la definición de reglas crece de manera exponencial. Mientras tanto para las ANN y CNN, aunque aumente el coste computacional de estas, el tener mayor número de entradas o de salidas en la red aumenta la complejidad de su estructura, pero no de su definición o creación.
- Las CNN presentan un coste computacional mucho mayor que la lógica borrosa y, aunque en menor medida, mayor que las ANN. El hecho de trabajar con imágenes (que incluyen gran cantidad de parámetros) hace que tanto el proceso de entrenamiento como el de inferencia sea mucho mayor, por lo que se necesita un hardware que sea capaz de procesar tanta cantidad de información en el tiempo requerido.
- Mientras que la lógica borrosa ha sido utilizada de manera muy prolija en décadas anteriores, las CNN y ANN están ganando clara popularidad ligada al aumento del rendimiento de procesadores y GPU, además de por la capacidad que tienen estas redes de computar grandes cantidades de datos con resultados muy buenos. Esto hace que tanto las CNN como las ANN sigan en continuo progreso actualmente, surgiendo nuevas arquitecturas y formas de optimización de estas de manera recursiva.

Centrando el foco de la comparativa en desempeño de cada una de las técnicas desarrolladas, se hace un análisis exhaustivo del desempeño de los diferentes algoritmos (implementados en diferentes robots RobotFL, RobotANN y RobotCNN) para cada uno de los circuitos probados (C1, C2 y C3). Las variables a considerar en la comparativa son: el tiempo medio por vuelta (de la vuelta 2 a la vuelta 5), las colisiones producidas por el robot, la velocidad máxima alcanzada y el tipo de conducción. Para esta última prueba, se evaluarán distintos parámetros como puede ser la trazada, las oscilaciones o las salidas de pista para ordenar los diferentes robots de 3 a 1 (siendo 3 el mayor puntaje que se puede obtener en esta categoría).

	Tiempo medio por vuelta (min:seg:dec)			Colisiones	Velocidad máxima (Km/h)			Conducción
	C1	C2	C3	C123	C1	C2	C3	C123
FL	58:93	1:08:11	44:68	0	249	234	207	3
ANN	3:12:23	1:35:87	1:59:73	0	39	100	81	2
CNN	55:32	1:01:43	41:23	5	240	221	201	1

Tabla 5. Resultado de la comparativa

Haciendo un primer acercamiento a los resultados obtenidos, se puede observar que, aunque el tiempo medio por vuelta de la RobotCNN es superior al del RobotFL, ambos tienen un desempeño parecido en esta categoría. Sin embargo, la actuación del robot implementado con ANN, es bastante inferior en cuando a tiempo por vuelta. Esto es debido a que, aunque presenta una buena conducción sin colisiones, no es capaz de alcanzar más de 100 Km/h en ninguno de los circuitos propuestos, perjudicando gravemente el tiempo por vuelta.

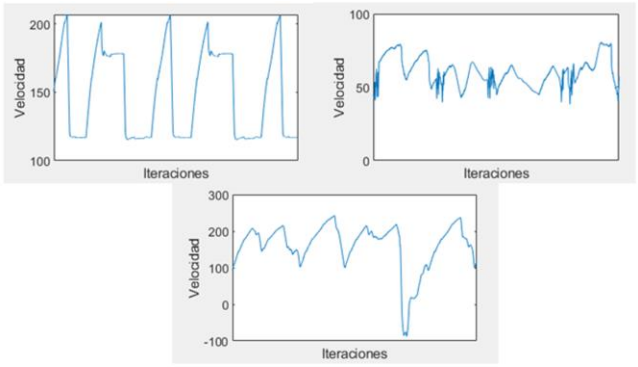

Fig.5. Representación de la variable speed en RobotFL, RobotANN y RobotCNN respectivamente

En figura 5, donde se hace una comparativa de las velocidades alcanzadas en el transcurso de 3 vueltas en el circuito 3, se observa que en efecto, la velocidad del robot que implementa ANN se mantiene alrededor de 50 Km/hora con cambios muy bruscos de velocidad, mientras que el de sus competidoras presentan un mayor rango, alcanzando velocidades que superan los 200 Km/h. Cabe recordar que RobotCNN no implementa un control de velocidad basado en CNN, lo que no se debe de tener en cuenta para este análisis.

Se puede observar de igual manera, el efecto que tienen los diferentes algoritmos aplicados en la salida de velocidad del coche. Mientras que tanto ANN como CNN presentan una inconsistencia mayor en las velocidades, RobotFL muestra patrones mucho más ordenados, en donde se pueden ver los diferentes niveles de velocidad que se han definido mediante reglas (120 km/h velocidad baja, 180km/h velocidad y 260 km/h velocidad alta). Esta diferencia surge de las distintas formas de implantación de los modelos. Mientas que en RobotFL se fija una velocidad específica a la que ir según las variables de entrada, en RobotCNN y RobotANN se ataca directamente al acelerador y a los frenos sin establecer una velocidad concreta. En RobotCNN, también se puede apreciar una bajada de velocidad drástica que llega incluso a obtener una velocidad negativa. Esta velocidad negativa está relacionada con una colisión, salida de pista y su correspondiente reincorporación a la misma.

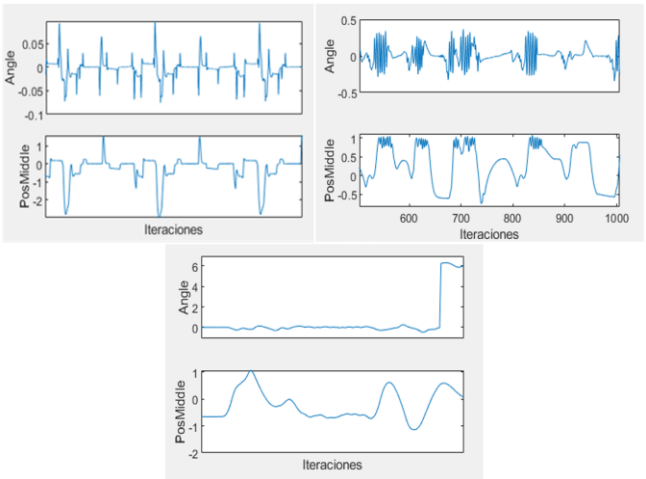

Fig.6. Representación de las variables Angle y PosMiddle en RobotFL, RobotANN y RobotCNN respectivamente

Con el fin de determinar cuál es el algoritmo que implementa la mejor trayectoria o conducción, se ha visto el desempeño de todos los robots, teniendo en cuenta si hacían movimientos erráticos, oscilatorios, o si simplemente no se ajustaban a la trazada propuesta por la pista. También se han analizado y representado tanto las variables *trackPos* (distancia al centro de la carretera) y *angle* (desviación con respecto al ángulo central de la carretera) durante tres vueltas con el fin de obtener un análisis más en detalle (véase la figura 6).

RobotANN presenta una conducción aceptable, ya que no produce colisiones y se ajusta al interior de las curvas. Por el contrario, tiene un movimiento oscilatorio y constante en curvas que le hace moverse de derecha a izquierda para intentar encontrar la posición deseada. Debido a los datos a los que se ha sometido, RobotCNN muestra a simple vista una conducción parecida a la humana, sin oscilaciones ni rectificaciones muy buscas, pero presenta una mayor impredecibilidad que causa que en la mayoría de curvas rápidas tenga una alta probabilidad de colisión. La gráfica obtenida para la variable ángulo para este robot no es muy representativa, ya que los valores que produce cuando el robot está fuera de la pista no son fiables. En cuanto a la gráfica de la posición, vemos cambios leves que intentan ajustarse a la trazada de una manera más suave que la implementada por RobotANN. El alto gasto computacional ha podido jugar un papel importante en las colisiones, ya que como se explicó anteriormente, el software de competición recibe información cada 30 ms, y si esa información no está disponible, el robot toma la acción anterior. De esta manera, si el tiempo de procesar la imagen de entrada y convertirlo en una acción para el volante es muy elevado, es posible que en curvas donde sea necesario reaccionar rápido, se produzca una colisión.

Por último, RobotFL presenta la conducción más ordenada de todas. Aunque, debido a cómo está definido su algoritmo, intenta trazar las curvas por el centro de la pista, en curvas rápidas es despedido hacia el exterior de la curva hasta que vuelve a retomar el centro. Tal y como se muestra en la figura 6, no presenta grandes valores de las variables Angle y trackPos y como ya sucedió en la gráfica de la velocidad, es el único algoritmo que actúa de igual manera en todas las vueltas, realizando de forma continua el mismo tiempo por vuelta. Los valores reducidos de las variables Angle y trackPos, hacen que la conducción no presente ni grandes oscilaciones ni cambios bruscos, evitando que se produzcan colisiones.

Tanto Robot ANN como RobotCNN, han sido entrenados con datos. Estos datos producen que la red sea capaz de aprender los patrones comunes de cada conducción. La principal diferencia de ambas redes es que, mientras que los datos de con los que ha sido entrenada la ANN han sido extraídos de otro Robot que implementaba otro algoritmo de conducción automático, los datos de la red de CNN han sido generados por una conducción humana. Esto provoca que los patrones de esta última no sean tan reconocibles (ya que una conducción humana puede ser más errática y no presenta siempre las mismas reacciones a los mismos estímulos) pudiendo afectar al entrenamiento de la red propiciando mayores colisiones.

Por todo lo anterior, se le atribuye la mejor conducción al robot que implementa Lógica Borrosa, seguido del que implementa ANN y por último, el de CNN (con 3, 2 y 1 punto respectivamente, tal que como se muestra en la tabla 5).

VI. Discusión

Podemos decir que tal y como está diseñado el experimento, y teniendo en cuenta que esta valoración está condicionada al objetivo por conseguir de cada, el robot que implementa lógica borrosa ha demostrado tener unas mejores prestaciones en conjunto ya que, aunque no tenga el mejor paso por vuelta, a diferencia de sus competidores, no presenta ningún punto de gran debilidad.

VII. Conclusiones

En este trabajo, se ha realizado por primera vez, una comparativa de distinto algoritmos de inteligencia artificial implementados en el simulador de coches TORCS con la finalidad de resolver problema de la conducción autónoma. Estos algoritmos han sido las CNN, las ANN y la Lógica borrosa.

Se ha conseguido el objetivo de realizar todo el proceso de desarrollo, optimización e implementación de los algoritmos para comparar no solo los resultados obtenidos, sino también las ventajas y desventajas de la implantación de cada uno de ellos en este caso concreto. A través de la experimentación y optimización de los algoritmos, se ha concluido que, uno de los factores que produce unas mayores diferencias en cuanto a conducción, además del cambio de los diferentes algoritmos, es el de su propia optimización. Algunos de estos parámetros más determinantes son la definición de reglas y funciones de pertenencia en la lógica borrosa o la definición de la arquitectura, entrenamiento, recolección y tratamiento de datos tanto para las ANN como las CNN.

Por último, se ha realizado un análisis exhaustivo de las variables más representativas relacionadas con la conducción, para así obtener conclusiones sobre cada uno de los algoritmos presentados.

Aunque los experimentos propuestos han demostrado que, en un entorno cerrado y controlado, la lógica borrosa es el algoritmo que ha presentado los mejores resultados en conjunto, no se ha podido dar una respuesta rotunda a la pregunta de cuál es el mejor algoritmo de inteligencia artificial para la resolución del problema de conducción autónoma. Esto se debe a que la elección de la mejor técnica depende de cuál sea el objetivo que se quiera maximizar dentro del problema (rapidez, conducción, fiabilidad etc.).

Trabajos futuros pueden incluir en una primera instancia, una optimización exhaustiva de cada uno de los algoritmos propuestos con el fin de llegar al límite de sus capacidades. También se pueden aplicar otros algoritmos de inteligencia artificial, como pueden ser el aprendizaje por refuerzo, o se pueden añadir a los algoritmos ya implementados, otros que hacen que destaquen sus características. Este podría ser el caso de combinar CNN con redes neuronales recurrentes para crear funciones que almacenan variables a lo largo del tiempo, o ANN y lógica borrosa con algoritmos genéticos para la optimización de hiperparámetros.

Otra propuesta adicional sería el incremento de la variabilidad y o la implementación de los algoritmos utilizando datos de entornos reales. Una aproximación a este acercamiento, puede ser el uso de dataset KITTI [9] para el entrenamiento y la prueba del algoritmo de CNN. Para el resto de los algoritmos, Para estimar variables de entrada la posición central, ángulo respecto a la carretera o distancia a la siguiente curva, se necesitarían otro número más elevado de sensores que complican su implantación en entornos reales.

Referencias

[1] Athanasiadis, C., Galanopoulos, D., & Tefas, A. (2012). Progressive neural network training for the open racing car simulator. Paper presented at the 2012 IEEE Conference on Computational Intelligence and Games (CIG), 116-123.

[2] Benoit, A., Bonnaud, L., Caplier, A., Ngo, P., Lawson, L., Trevisan, D. G., . . . Chanel, G. (2009). Multimodal focus attention and stress detection and feedback in an augmented driver simulator. Personal

[3] Bogoni, T. N., & Pinho, M. S. (2012). Use of a simulator to assess the application of economic driving techniques by truck drivers. Paper presented at the 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 3020-3026.

[4] Chen, C., Seff, A., Kornhauser, A., & Xiao, J. (2015). Deepdriving: Learning affordance for direct perception in autonomous driving. Paper presented at the Proceedings of the IEEE International Conference on Computer Vision, 2722-2730.

[5] Drolia, U., Wang, Z., Pant, Y., & Mangharam, R. (2011). AutoPlug: An automotive test-bed for electronic controller unit testing and verification. Paper presented at the 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), 1187-1192.

[6] Forsyth, D., & Ponce, J. (2011). Computer vision: A modern approach. Prentice hall.

[7] Fujii, S., Nakashima, T., & Ishibuchi, H. (2008). A study on constructing fuzzy systems for high-level decision making in a car racing game. Paper presented at the 2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress *on* Computational Intelligence), 2299-2306.

[8] Gamal, O., Imran, M., Roth, H., & Wahrburg, J. (2020). Assistive parking systems knowledge transfer to end-to-end deep learning for autonomous parking. Paper presented at the 2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE), 216-221.

[9] Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The kitti dataset. The International Journal of Robotics Research, 32(11), 1231-1237.

[10] Ho, D. T., & Garibaldi, J. M. (2008). A novel fuzzy inferencing methodology for simulated car racing. Paper presented at the 2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence), 1907-1914.

[11] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012a). Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097-1105

[12] Manrique Gamo, D., & Suárez de Figueroa Baonza, María del Carmen. (2015). Razonamiento con imprecisión: Lógica borrosa.

[13] Munoz, J., Gutierrez, G., & Sanchis, A. (2009a). Controller for torcs created by imitation. Paper presented at the 2009 IEEE Symposium on Computational Intelligence and Games, 271-278.

[14] Onieva, E., Pelta, D. A., Alonso, J., Milanés, V., & Pérez, J. (2009). A modular parametric architecture for the torcs racing engine. Paper presented at the 2009 IEEE Symposium on Computational Intelligence and Games, 256-262.

[15] Perez, D., Recio, G., Saez, Y., & Isasi, P. (2009). Evolving a fuzzy controller for a car racing competition. Paper presented at the 2009 IEEE Symposium on Computational Intelligence and Games, 263-270.

[16] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386.

[17] Salem, M., Mora, A. M., Merelo, J. J., & García-Sánchez, P. (2017). Driving in TORCS using modular fuzzy controllers. Paper presented at the European Conference on the Applications of Evolutionary Computation, 361-376.

[18] Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. Evolutionary Computation, 10(2), 99-127.

[19] Tran, D., Do, H. M., Lu, J., & Sheng, W. (2020). Real-time detection of distracted driving using dual cameras. Paper presented at the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2014-2019.

[20] Wang, Y., Liu, D., Jeon, H., Chu, Z., & Matson, E. T. (2019). End-to-end learning approach for autonomous driving: A convolutional neural network model. Paper presented at the Icaart (2), 833-839.

[21] Wymann, B., Espié, E., Guionneau, C., Dimitrakakis, C., Coulom, R., & Sumner, A. (2000). Torcs, the open racing car simulator. Software Available at Http://Torcs.Sourceforge.Net, 4(6), 2.