

Sistema de navegación adaptativa para un vehículo autónomo en un entorno con establecimiento objetivo

1^{er} Over Alexander Mejia Rosado ✉

Inteligencia Artificial

Universidad Nacional de Colombia - De La Paz
San Diego

2^{do} Ronald Mateo Ceballos Lozano ✉

Inteligencia Artificial

Universidad Nacional de Colombia - De La Paz
Valledupar

3^{er} Rhonald José Torres Diaz ✉

Inteligencia Artificial

Universidad Nacional de Colombia - De La Paz
Valledupar

I. INTRODUCCIÓN

Uno de los mayores problemas del transporte moderno es el alto índice de accidentes en el tránsito terrestre, la gran cantidad de tráfico atribuida a factores como errores humanos, falla mecánicas y fenómenos naturales.

Según un estudio realizado por la Organización Mundial de la Salud (OMS) en 2023, entre 2010 y 2021 el número de vehículos a nivel mundial se duplicó, superando los mil millones. Este crecimiento ha contribuido a un incremento en los accidentes de tránsito, que provocaron aproximadamente 1,19 millones de muertes en todo el mundo, con una tasa de mortalidad de 15 personas por cada 100,000 habitantes, lo que representa una disminución del 5% en comparación con 2010. Además, cerca del 80% de las vías de tránsito a nivel mundial no cumplen con los estándares básicos de seguridad para peatones y ciclistas. Esta falta de infraestructura segura expone a ciclistas y peatones a un alto riesgo de accidentes, especialmente en entornos urbanos con alta densidad de tráfico, lo que también genera congestión vehicular, alargando los tiempos de trayecto, incrementando las emisiones de gases contaminantes y provocando un desperdicio de combustible [1].

Estos factores resaltan la necesidad de intervenciones más efectivas, como la automatización y los sistemas inteligentes de tráfico, que permitan reducir las cifras de mortalidad y mejorar la eficiencia del tránsito mediante sistemas coordinados.

I-A. Sistemas Autónomos

Con el creciente avance tecnológico, la implementación de vehículos autónomos pasó de ser un futuro cercano a un presente en constante mejora. Estos sistemas mejoran la seguridad y eficiencia tanto del vehículo como del transporte en general, enfocándose no solo en el control del automóvil, sino también en la capacidad del sistema para adaptarse a condiciones variables y mantener una trayectoria segura para los pasajeros y otros agentes viales [2]. El control de movimiento

y la precisión en el seguimiento de trayectoria son aspectos críticos para la evaluación del agente inteligente a bordo del vehículo, siempre con el objetivo de preservar la integridad de las personas dentro del automóvil. El control de movimiento en vehículos autónomos, en especial el control óptimo de seguimiento de trayectoria busca mantener al vehículo dentro de los límites de seguridad establecidos, optimizando los movimientos del automóvil y ajustando el comportamiento del vehículo a través de inteligencia artificial avanzada, como la ADP (Adaptive Dynamic Programming), o incluso Approximate Dynamic Programming (Programación dinámica aproximada) [3]. La ADP es una técnica de control avanzada utilizada en los vehículos autónomos para la toma de decisiones en tiempo real, considerando condiciones de incertidumbre, que con tecnología de control basada en el aprendizaje basada en refuerzo, el uso de programación dinámica adaptativa crítica, aumenta el rendimiento óptimo del sistema [4], [5]. El ADP genera una función de costo que evalúa el rendimiento del control del vehículo, con el objetivo de minimizar esta función gracias a procesos iterativos, ajustando continuamente las acciones para optimizar el rendimiento y reducir errores de seguimiento en la trayectoria. Este artículo [4] propone un método de control óptimo adaptativo con rendimiento prescrito para resolver el problema del control de seguimiento de trayectorias para vehículos autónomos con dinámica incierta, al introducir una función de rendimiento prescrito (PPF) en la programación dinámica adaptativa (ADP), el controlador puede restringir el error de seguimiento del sistema dentro de un límite de rendimiento especificado al tiempo que optimiza el costo de control.

El aprendizaje por refuerzo que ha demostrado ser eficaz en la toma de decisiones en vehículos autónomos es el Deep Q-Network (DQN). En este enfoque, el agente aprende a tomar decisiones a través de la interacción con su entorno, recibiendo recompensas o penalizaciones en función de sus acciones, lo que mejora su aprendizaje [6]. La técnica DQN utiliza

funciones que estiman el valor de las acciones en los diferentes estados en los que se encuentra el agente, representando así la expectativa de recompensa acumulada al realizar una acción específica desde un estado particular.

Además, DQN implementa un balance entre exploración y explotación. La exploración implica probar nuevas acciones, mientras que la explotación se refiere a elegir acciones que el agente considera "buenas". Este balance se evalúa mediante el mecanismo epsilon-greedy, donde el agente elige acciones aleatorias con una probabilidad ϵ que varía entre 0 y 1 [7]. Un valor alto de ϵ permite una mayor exploración, mientras que un valor bajo indica una mayor explotación de las acciones ya aprendidas. Sin embargo, debido a que el operador máximo en DQN selecciona el valor máximo para evaluar una acción producida por el mismo valor Q, existe el riesgo de obtener valores estimados de manera demasiado optimista. Para mitigar este problema, se propone disociar la selección de la evaluación, lo que da lugar al enfoque Double DQN [8]. En este método, se utilizan dos funciones de valor (redes Q) que se aprenden actualizando alternativamente una de ellas, resultando en dos conjuntos de pesos de red: w y w' . La función de valor con peso w se utiliza para determinar la acción a tomar, de acuerdo con una política voraz, mientras que la función con peso w' se utiliza para evaluar el valor Q.

DQN puede ser implementado en los casos donde se requiera alta precisión, como por ejemplo la construcción mapas. También aplicar el DQN en la conducción autónoma y el transporte inteligente, mejorando así la trazabilidad de las trayectorias en vehículos autónomos [9].

El Deep Deterministic Policy Gradient (DDPG), es otro algoritmo de aprendizaje por refuerzo diseñado para resolver problemas de control continuo, este dependiendo del entorno, puede aumentar la optimización del sistema, agregando un mayor índice de credibilidad al algoritmo [10].

Por otro lado se encuentra la PPO (Proximal Policy Optimization) una técnica de aprendizaje por refuerzo avanzada y popular utilizada en entornos complejos de sistemas de vehículos autónomos. La PPO pertenece a los métodos de optimización de políticas, lo que significa que se centra en las estrategias a seguir con respecto al entorno dinámico donde se encuentra el agente. Esta técnica introduce una optimización proximal para limitar cuánto se puede actualizar la política en cada paso de entrenamiento, limitando así los grandes cambios en las políticas y estabilizando el aprendizaje para que el agente sea consistente [11]. A diferencia de los métodos basados en funciones Q, PPO aprende directamente una política estocástica, lo que significa que genera probabilidades para cada acción en lugar de seleccionar siempre la misma acción en un estado determinado. Este implementa un algoritmo *actor – critic* donde el actor aprende la política (qué acción tomar en cada estado) y el critic estima el valor de cada estado (cuál es la recompensa esperada en un estado dado). Esta técnica se ha utilizado en el siguiente artículo [12] donde se presentó un enfoque basado en MAPPO (Multi-Agent Proximal Policy Optimization) para garantizar la maniobra segura y eficiente de vehículos autónomos en

presencia de vehículos de emergencias. El método propuesto genera políticas cooperativas que permiten al vehículo de emergencia circular a una velocidad media un 15 % superior manteniendo unas distancias de seguridad elevadas.

También tenemos el enfoque de Soft Actor-Critic (SAC) que combina dos características avanzadas en aprendizaje por refuerzo profundo: aprendizaje fuera de línea y ajuste automático de la función de valor. El SAC se aplica para maximizar tanto la recompensa acumulada como la entropía de las acciones, lo que ayuda a gestionar entornos complejos y requisitos de control variables. En concreto, el algoritmo incluye un componente de política estocástica, que beneficia la planificación de la trayectoria del agente autónomo al mejorar la adaptabilidad de las acciones de control en entornos inciertos y dinámicos. El enfoque basado en SAC también utiliza una estructura de recompensa compuesta para equilibrar múltiples objetivos de optimización, como la eficiencia del controlador, la solidez y el uso de energía, lo que mejora el rendimiento general y la precisión de la navegación [13]. Este cuenta con distintas variantes, como lo es el LGE-SAC, este introduce una variante en el manejo de experiencias, enfocándose en aprender de las experiencias positivas en la memoria de repetición de experiencias. También existe el RSAC, este incrementa la optimización a través de su adaptabilidad debido al manejo complejo de recompensas, permite a los agentes aprender de experiencias de mayor calidad de manera más rápida, optimizando el proceso de aprendizaje en entornos complejos [14].

La adaptabilidad del sistema es esencial para la precisión y la velocidad en la toma de decisiones, esto es aplicado en la búsqueda de rutas que se han convertido en factores clave para medir el rendimiento de un AGV (Vehículo de Guiado Automático) en un RMFS (Sistema de Manejo de Materiales en Tiempo Real). El algoritmo de Dijkstra es un método clásico para la búsqueda de rutas, ya que determina la trayectoria disponible desde el nodo de inicio hasta el nodo de destino explorando todos los nodos posibles. Por su parte, el algoritmo A* es una extensión del algoritmo de Dijkstra que mejora el rendimiento al utilizar heurísticas para guiar su búsqueda.

Sin embargo, la exploración y el registro de todos los nodos posibles pueden requerir mucho tiempo, especialmente en sistemas de gran tamaño. En este contexto, el aprendizaje por refuerzo profundo se presenta como una solución prometedora para resolver problemas de búsqueda de rutas. A través de una red neuronal entrenada, el AGV puede tomar decisiones informadas basadas en la situación del sistema y determinar la ruta más adecuada.

Para la planificación de rutas de un solo AGV, se puede emplear un algoritmo de aprendizaje Q, que es un enfoque basado en valores. Para escenarios que involucran múltiples AGV, se puede utilizar un algoritmo de aprendizaje por refuerzo conocido como Actor-Critic, que ayuda a encontrar rutas libres de conflictos entre varios vehículos [15].

Sin duda, la evolución de los sistemas de transporte (TS) y la aparición de vehículos autónomos (VA) marcan avances

fundamentales que configuran las ciudades modernas. Sin embargo, los VA enfrentan importantes desafíos en entornos complejos e impredecibles. El transporte urbano ha evolucionado significativamente, enfatizando la eficiencia, la seguridad y los avances en TS para abordar la rápida expansión urbana. A pesar de estos progresos, la complejidad de lograr sistemas completamente autónomos sigue siendo un reto.

Esta complejidad se origina en escenarios de tráfico intrincados, la necesidad de sistemas robustos de prevención de colisiones y la integración de tecnologías impulsadas por inteligencia artificial (IA) para una toma de decisiones óptima. En este contexto, la investigación destaca el papel del aprendizaje por refuerzo (RL) como una herramienta fundamental para permitir que los VA naveguen en escenarios de tráfico inciertos y complicados. El RL capacita a los VA para adaptarse y tomar decisiones óptimas a través de un proceso de aprendizaje basado en prueba y error, mejorando significativamente su resiliencia y confiabilidad en situaciones impredecibles.

Además, el aprendizaje automático, junto con la IA, gestiona la detección de objetos, rastrea múltiples entidades y pronostica posibles escenarios en el entorno del vehículo. La integración de algoritmos de aprendizaje por refuerzo profundo (DRL) optimiza el uso de los sensores de los vehículos autónomos, mejorando sus capacidades de conducción [16]. La combinación de todas estas técnicas, elaboradas dentro de una arquitectura clásica de control, permitirá una navegación autónoma eficiente que priorice el cumplimiento de la ruta, evite colisiones y se adapte en tiempo real a situaciones dinámicas.

Para complementar esta arquitectura, el método de Neuro-Evolución de Topologías Aumentadas (NEAT) ofrece una potente herramienta de aprendizaje evolutivo en la creación de redes neuronales artificiales (ANN). NEAT utiliza algoritmos genéticos (GA) para optimizar las redes neuronales, aumentando gradualmente su complejidad según la necesidad de la tarea. En este contexto, el estudio en [17] presenta un modelo de simulación de evacuación basado en agentes, en el cual se estudia la dinámica de diferentes vehículos y su proceso de aprendizaje mediante NEAT. Este método refuerza la analogía entre los GA y la evolución biológica, permitiendo que las redes neuronales evolucionen de manera adaptativa para afrontar entornos dinámicos y optimizar soluciones de forma simultánea.

El rendimiento de los algoritmos de Neuro-Evolución se compara favorablemente con el de los algoritmos de retropropagación basados en gradientes. Una característica clave de NEAT es que el proceso de evolución comienza a partir de redes neuronales muy simples, cuya topología aumenta gradualmente en complejidad a lo largo de la evolución[18]. Esta característica reduce la complejidad innecesaria de la red neuronal final, algo que no es posible lograr utilizando algoritmos basados en gradientes. Este trabajo examinó cómo el algoritmo NEAT (NeuroEvolution of Augmenting Topologies) permite la optimización evolutiva de redes neuronales mediante topologías dinámicas y mutaciones estructurales. La implementación de NEAT aborda problemas complejos como

la clasificación y el control en tiempo real, empleando especiación y un registro histórico de innovaciones para mantener la diversidad y mejorar la precisión sin ajuste manual. también implementando algoritmos evolutivos como Evolutionary Acquisition of Neural Topologies, (EANT) los cuales pueden superar NEAT en temas como el rendimiento [19].

I-B. Sistemas Multi-Agentes

Uno de los mayores desafíos para los vehículos autónomos es la capacidad de navegar de forma segura en entornos con agentes aleatorios, como intersecciones donde otros vehículos y peatones pueden cambiar de dirección de forma impredecible. El manejo seguro de intersecciones es esencial, ya que estos puntos concentran altos índices de accidentes y congestión vehicular, incrementando el tiempo de espera y el consumo de combustible en áreas urbanas densas.

Este proyecto se inspira en soluciones basadas en sistemas multi-agente, que emplean aprendizaje por refuerzo y redes neuronales profundas para coordinar el tráfico en intersecciones, permitiendo a los vehículos autónomos pasar de forma segura y eficiente sin intervención humana [20]. Esta técnica aporta mayor adaptabilidad y seguridad en comparación con sistemas estáticos, que requieren ajustes específicos y no consideran agentes móviles inesperados.

La implementación de estos sistemas autónomos incorpora herramientas de retroalimentación como el aprendizaje reforzado y redes neuronales, que permiten evaluar cada acción tomada por el agente inteligente y ajustar parámetros para mejorar las decisiones y alcanzar los objetivos propuestos [21].

II. METODOLOGÍA

Para la construcción del sistema de navegación adaptativa, se uso el entorno con la librería Pygame de python. Se asigna un mapa y un agente principal (Vehículo Autónomo). El entorno implementado es una fracción del mapa clásico del video juego GTAII, además la imagen del agente, y parámetros iniciales de configuración es proporcionada por el repositorio NeuralNine, tambien se hace uso de algunas imágenes que pertenecen al repositorio Mihir Gandhi. Se establecen los estados, un estado inicial donde se ubica el vehículo autónomo y un estado final. Se asignan pixeles verdes RBG(0, 0, 255) al mapa, que será el objetivo.

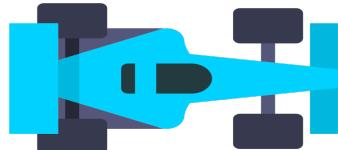


Figura 1: Agente previo a la aplicación de la red neuronal

II-A. NEAT-Python

El agente debe aprender a moverse en el entorno, NEAT permite que el agente recorra el espacio asignándole valores de salida dependiendo de los valores de entrada. Los valores de entrada son valores que se toman del entorno, para este

caso en particular, las entradas estarían dadas por el color de cada pixel, dependiendo del color del pixel, el agente cambia o no de dirección, a esto se le conoce como acción o salida.

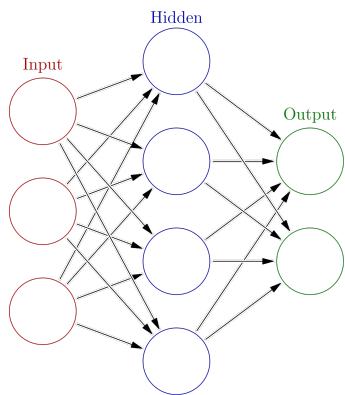


Figura 2: Ejemplo de una red neuronal por [22]

NEAT le otorga al agente sensores que le permiten el reconocimiento del entorno, estos sensores son asignados para determinar cuando este sale de las vías, y la distancias entre puntos [23]. En este artículo implementamos el uso de 5 sensores, asignados a una lista: -90°, -45°, 0°, 45° y 90°. Estas son las principales entradas para nuestro algoritmo NEAT aplicado al agente Figura 1, ver Figura 3.

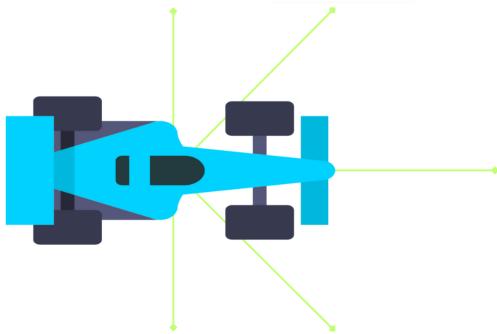


Figura 3: Asignación de sensores al agente

En la Figura 2, Hidden o capas ocultas, que son nodos que aplican un procesado a la entrada, con cada generación las capas aumentan dependiendo de la ganancia, esta ganancia está dada por las acciones previas del agente. Si el agente se desplaza en el mapa sin exceder las limitaciones, se mantiene en la generación. Las generaciones en NEAT es el umbral con el que se desea finalizar la simulación, por ejemplo, si se establecen 50 generaciones cada valor del umbral se le asocian a los agentes creados por la red neural aplicando distintos valores de configuración, si en la primera generación todos los coches chocan, se pasa a la siguiente generación, en este caso la segunda, así sucesivamente hasta llegar a la última que sería la generación 50. En el entorno se crean distintos agentes, cada agente recorre el mapa, si un agente choca o se sale de la vía, esta configuración tendrá menos probabilidad de replicarse, de esta manera se garantiza una generación final

donde el o los agentes aprenda a desplazarse por el entorno de forma autónoma.

En la configuración inicial, Hidden es igual a 0, puesto que se inicia con una configuración sencilla, con cada generación Hidden aumenta para que la salida dada la entrada sea la más óptima.

II-B. Fitness function

Para evaluar que tan bien son los genomas (próximas generaciones) Neat implementa una función que evalúa que tan bien resuelve el problema en cuestión, esta función es llamada *Fitness function*. Si un genoma A resuelve el problema de manera más exitosa que un genoma B, entonces el valor de aptitud de A debe ser mayor que el de B. La magnitud absoluta y los signos de estas aptitudes no son importantes, solo sus valores relativos, los pasos estarán dados por:

- Crear una red neuronal: Basada en el genoma.
- Proveer entradas y calcular salidas: Por ejemplo, para cada caso en la tabla de verdad del XOR, se proporcionan las entradas a la red y se calcula la salida.
- Calcular el error: El error se calcula entre las salidas esperadas y las salidas reales de la red.
- Asignar aptitud: Si la red produce exactamente la salida esperada, su aptitud es 1. De lo contrario, es un valor menor a 1, disminuyendo más cuanto más incorrectas sean las respuestas de la red [24].

La ecuación de actitud sería:

$$\text{Aptitud} = 1 - \sum_i (e_i - a_i)^2$$

Donde, e_i (salidas esperadas): Son los valores que se esperan obtener de la red neuronal o agente autónomo. Las salidas esperadas son las posiciones ideales o los movimientos óptimos que el agente debería realizar para alcanzar el objetivo de la manera más eficiente posible.

a_i (salidas reales): Son los valores que realmente se obtienen de la red neuronal o agente autónomo. Estas son las posiciones o movimientos que el agente realiza durante su desplazamiento en el mapa.

Función de Aptitud. En el contexto de nuestro proyecto, la función de aptitud es fundamental para evaluar y guiar el proceso de evolución de los agentes autónomos (vehículos) controlados por redes neuronales generadas mediante el algoritmo NEAT.

Los pasos para calcular la aptitud de cada genoma son los siguientes:

- **Crear una red neuronal:** Se genera una red neuronal basada en el genoma actual.
- **Simular el agente:** La red neuronal controla al agente autónomo en el entorno simulado, recibiendo entradas del entorno (como las distancias detectadas por los sensores del agente) y produciendo acciones (como cambios en la dirección y velocidad del agente).
- **Recopilar métricas:** Durante la simulación, se registran métricas relevantes, como la distancia mínima al objetivo, el tiempo de supervivencia y si el agente ha colisionado.

- **Calcular la aptitud:** Se utiliza una función de aptitud definida para calcular el valor de aptitud del genoma, basándose en las métricas recopiladas.

La función de aptitud implementada es:

$$\text{Aptitud} = R_d$$

Donde:

- R_d es la recompensa por distancia al objetivo.

Cada componente se calcula de la siguiente manera:

- **Recompensa por distancia al objetivo (R_d):**

La recompensa por distancia es inversamente proporcional a la distancia d_{agent} entre el agente y el objetivo:

$$R_d = \begin{cases} 10000, & \text{si } d_{agent} = 0 \\ 10000 - d_{agent}, & \text{si } d_{agent} > 0 \end{cases}$$

Donde para la distancia Euclíadiana d_{agent} se calcula como:

$$d_{agent} = \sqrt{(x_{agent} - x_{goal})^2 + (y_{agent} - y_{goal})^2}$$

Para la distancia de Manhattan d_{agent} se calcula como:

$$d_{agent} = |x_{agent} - x_{goal}| + |y_{agent} - y_{goal}|$$

El tercer y último método es la distancia de Chebyshev, esta es calculada de la siguiente manera

$$d_{agent} = \max(|x_{agent} - x_{goal}|, |y_{agent} - y_{goal}|)$$

Aquí, (x_{agent}, y_{agent}) son las coordenadas actuales del agente, y (x_{goal}, y_{goal}) son las coordenadas de cada punto objetivo en la lista de objetivos. Esta recompensa incentiva al agente a acercarse lo más posible al objetivo.

Combinando los términos anteriores, la función de aptitud se expresa como:

$$\text{Aptitud} = 10000 - d_{agent}$$

Además, para asegurar que el valor de aptitud no sea negativo, se aplica:

$$\text{Aptitud} = \max(0, \text{Aptitud})$$

Entre mayor sea la aptitud, los genomas con sus características aumentan las probabilidades de ser seleccionados para la próxima generación.



Figura 4: Mapa del entorno donde se realizaran las simulaciones

II-C. Implementación de asignación entrada salida

Dadas las entradas de los sensores, se le asigna un angulo de rotación y una velocidad al agente autónomo, es un esquema básico, pero funcional ver Figura 5. Sin embargo, que pasa si las variaciones de los ángulos son muy altas, o tal vez muy bajas, o quizás la velocidad es muy lenta?

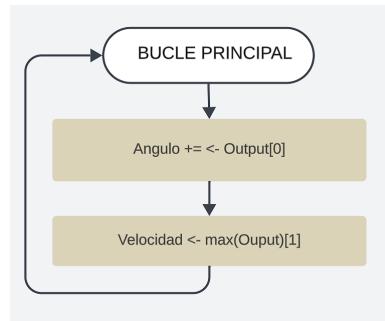


Figura 5: Bucle básico de como el agente autónomo es retroalimentado con las entradas

Las altas variaciones de los ángulos generan inconvenientes, uno de ellos es causar que el agente gire sin parar. Si el agente recibe una entrada constante de ángulos en una misma dirección, ocasiona que la suma constante de esos ángulos lo haga girar, más si dichas entradas se presentan en las primeras generaciones. Por contraparte, su desplazamiento se ve disminuido, lo que implica un estancamiento del agente. En el caso de que la entrada sea demasiada alterada pero iterando la dirección de los ángulos, su velocidad también es reducida, ya que si no lo hace, el carro será eliminado debido al inminente hecho de chocar con los bordes fuera de la zona establecida como carretera.

Para evitar alguna de las singularidades (tómese singularidad como estancamiento, rotación sin parar) se asigna un **Set_Value** a la velocidad si la velocidad previa del agente es menor a 0.1, esto evita que el agente avance demasiado lento, optimizando en medida el tiempo de cada simulación, en este artículo, **Set_Value** es igual a 5. A esta asignación la denominamos *Refuerzo forzado* o *Método de aceleración*.

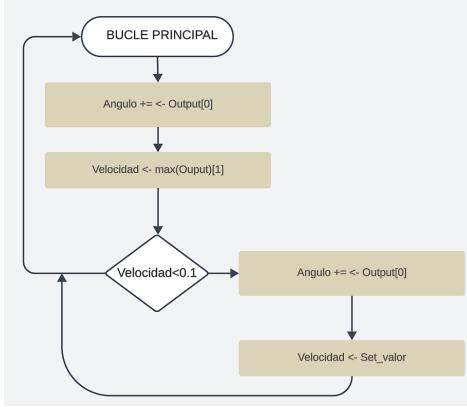


Figura 6: Sistema implementado para optimizar y reducir el estancamiento de algunos agentes en primeras generaciones, **Refuerzo forzado**

Una vez establecidos los parámetros iniciales, se realizan distintas simulaciones con el fin de recopilar datos para analizar el comportamiento de los agentes. Se realizaron un total de 48 simulaciones para el primer mapa ver Figura 4, distribuidas en grupos de tres, 20 simulaciones para la distancia Manhattan, 20 para la Euclídea y 20 para la distancia Chebyshev. Se distribuyen de la siguiente manera

Cuadro I: Orden de simulaciones

Simulaciones	Generaciones
Euclídea (5)	20, 30, 50
Manhattan (5)	30, 30, 50
Chebyshev (5)	20, 30, 50

a cada número de generación le corresponden 5 simulaciones, es decir en la distancia Euclídea según la tabla fue un total de $3 * 5 = 15$ simulaciones realizadas, donde 5 fueron las simulaciones realizadas con una generación de 20, otras 5 para la generación de 30 y por último otras 5 para la generación de 50; así para la distancia Manhattan y Chebyshev. Estas fueron realizadas con la aplicación del método **Refuerzo forzado** ver Figura 6, sin embargo se realizaron tres simulaciones adicionales sin este método. Corresponden a una simulación de 50 generaciones para las tres distancias aplicadas, por ende el total de simulaciones son las 45 contadas anteriormente, más 3 simulaciones sin la aplicación de este método.

Cuadro II: Orden de simulaciones

Simulaciones	Generaciones	Refuerzo forzado
Euclídea (5)	20, 30, 50	SI
Manhattan (5)	20, 30, 50	SI
Chebyshev (5)	20, 30, 50	SI
Euclídea (1)	50	NO
Manhattan (1)	50	NO
Chebyshev (1)	50	NO

También se realizaron 15 simulaciones adicionales para un segundo mapa ver Figura 7, en estas simulaciones solo se

realizaron para una generación de 50, la tabla sería de la siguiente manera:

Cuadro III: Orden de simulaciones

Simulaciones	Generaciones	Refuerzo forzado
Euclídea (5)	50	SI
Manhattan (5)	50	SI
Chebyshev (5)	50	SI

donde (5) representan las simulaciones realizadas, siguiendo el orden planteado anteriormente, 5 simulaciones realizadas para la distancia Euclídea con una generación de 50, y así para la distancia Manhattan y Chebyshev, para un total de $5 * 3 = 15$ simulaciones.



Figura 7: Mapa con bloqueo de vía inferior

III. RESULTADOS Y ANÁLISIS DE RESULTADOS

Durante la ejecución de las simulaciones se recopilaron datos para el análisis. Los registros obtenidos de las simulaciones contienen el fitness promedio de los genomas, el mejor fitness de la simulación, por últimos las máximas y mínimas recompensas. Con estos datos, como observara a continuación se gráfica el fitness promedio con las desviaciones.

III-A. Mapa implementado en la simulación

Durante la preparación del mapa para la simulación, a los bordes principales de la carretera se les asignó un color RGB(20, 23.5, 21.6). Sin embargo, en la configuración para

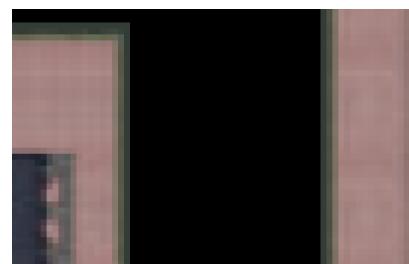


Figura 8: Bordes laterales de la vía usada en el mapa

determinar los límites de la vía se opta por establecer que los límites son todos los colores distintos del negro.

III-B. Configuraciones principales del algoritmo NEAT

El archivo de configuración para NEAT consta de distintas secciones, cada una de ellas orientada a como será la creación de generaciones en cada simulación. En la sección [NEAT] IV, se pueden encontrar los parámetros fitness y el tamaño de la población en cada generación; en la sección [DefaultReproduction] V, podemos encontrar la configuración de elitism y survival threshold, que permiten determinar el numero de especies mas aptas de cada generación que se desea conservar, y la proporción de cada especie permitida para reproducirse en cada generación; en la sección [DefaultGenome] VI, se define los parámetros específicos para la estructura de los genomas (redes neuronales) que Neat evolucionará, como el modo de activación o activation default y activation mutate rate.

III-C. Resultados de las simulaciones aplicando la distancia Euclidiana

III-D. 50 Generaciones

Durante las simulaciones realizadas se almacenaban los datos del fitness correspondiente a esta distancia, las primeras 5 simulaciones realizadas correspondieron a la generación número 50, como resultado se obtiene la Figura 9.

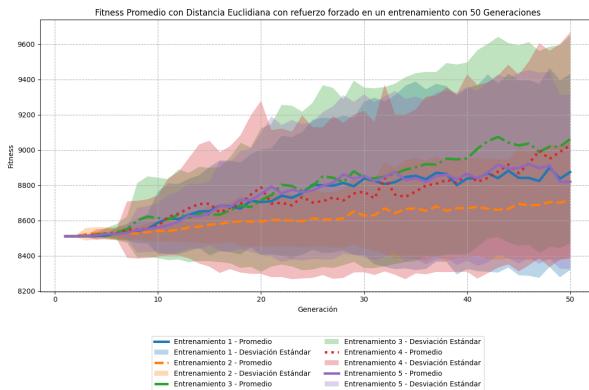


Figura 9: Gráfica de promedios y desviaciones fitness correspondiente a 5 simulaciones realizadas con una generación de 50

La zona sombreada corresponde a las desviaciones de cada simulación, estas desviaciones indican agentes alejados del promedio, basándose en ello se puede observar que simulaciones obtuvieron la mayor cantidad de puntos. De manera individual, el entrenamiento (simulación) 4 tiene el siguiente comportamiento:

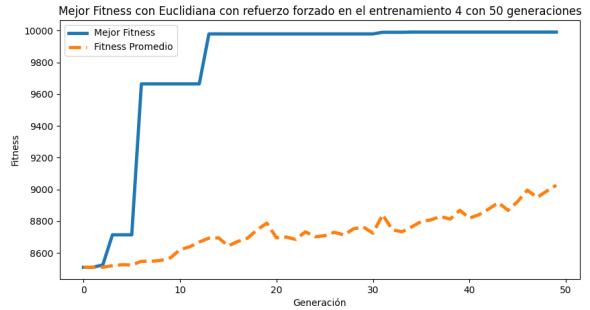


Figura 10: Fitness individual para el entrenamiento 4 de 50 generaciones aplicando la distancia Euclidiana

De todas las simulaciones realizadas para esta sección, el entrenamiento 4 obtuvo el fitness máximo, con un valor exacto de 9990. Ademas, este entrenamiento presentó la mayor desviación, como se observa en la Figura 11.

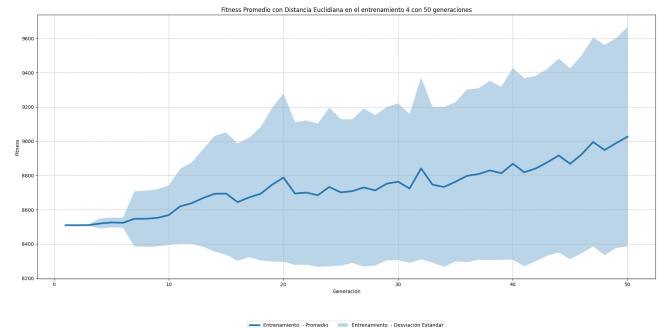


Figura 11: Fitness promedio y desviación individual para el entrenamiento 4 de 50 generaciones aplicando la distancia Euclidiana

El segundo fitness más alto le corresponde al entrenamiento 1, con un valor de 9988, además otra característica para el primer entrenamiento corresponde a la desviación, esta supera los 9400, sin embargo respecto al cuarto entrenamiento su desviación son 200 puntos por debajo (Fig. S1A-E y S2A-E). Seguido de este entrenamiento, el tercer entrenamiento obtuvo un fitness de 9985, con una desviación de 9600, misma desviación que el cuarto entrenamiento (Fig. S5A-E y S6A-E). El quinto entrenamiento obtuvo un fitness de 9982, con una desviación de 9400, en el mismo rango de desviación que el primer entrenamiento (Fig. S7A-E y S8A-E). Para finalizar, el entrenamiento 2 obtuvo el fitness más bajo con un puntaje de 9626, con la menor desviación de las 5 simulaciones (Fig. S3A-E y S4A-E).

III-D1. 30 Generaciones: Con los datos recopilados en estas 5 simulaciones para esta generación, la gráfica obtenida es:

Cuadro IV: [NEAT]

Variable	Valor
fitness_criterion	max
fitness_threshold	10000
pop_size	50
reset_on_extinction	True

Cuadro V: [DefaultReproduction]

Variable	Valor
excess_coeff	1.0
disjoint_coeff	1.0
weight_diff_coeff	0.5
compatibility_threshold	3.0
elitism	5
survival_threshold	0.2

Cuadro VI: [DefaultGenome]

Variable	Valor
Opciones de Activación de Nodos	
activation_default	tanh
activation_mutate_rate	0.01
activation_options	tanh, relu, sigmoid
Opciones de Agregación de Nodos	
aggregation_default	sum
aggregation_mutate_rate	0.01
aggregation_options	sum
Opciones de Sesgo de Nodos	
bias_init_mean	0.0
bias_init_stdev	1.0
bias_max_value	30.0
bias_min_value	-30.0
bias_mutate_power	0.5
bias_mutate_rate	0.7
bias_replace_rate	0.1
Opciones de Compatibilidad del Genoma	
compatibility_disjoint_coefficient	1.0
compatibility_weight_coefficient	0.5
Tasas de Adición/Eliminación de Conexiones	
conn_add_prob	0.5
conn_delete_prob	0.5
Opciones de Habilitación de Conexiones	
enabled_default	True
enabled_mutate_rate	0.1
Configuraciones de Topología	
feed_forward	False
initial_connection	full
Tasas de Adición/Eliminación de Nodos	
node_add_prob	0.2
node_delete_prob	0.2

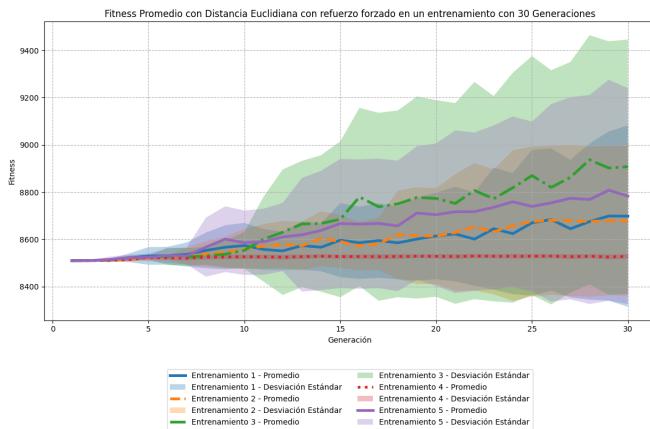


Figura 12: Gráfica de promedios y desviaciones fitness correspondiente a 5 simulaciones realizadas con una generación de 30

En el caso de la Figura 12, los entrenamientos que presentan las mayores desviaciones son el primer, tercero y quinto entrenamiento. El fitness máximo corresponde al tercer entrenamiento con 9940.

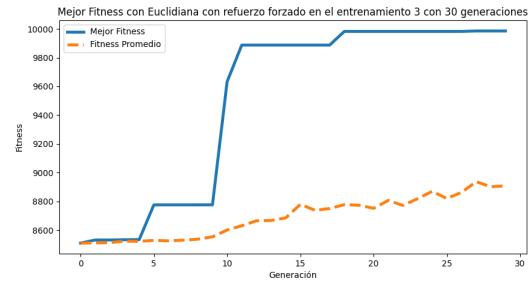


Figura 13: Fitness individual para el entrenamiento 3 de 30 generaciones aplicando la distancia Euclidiana

También el tercer entrenamiento presentó la mayor desviación de todas para esta sección, con un valor de 9400:

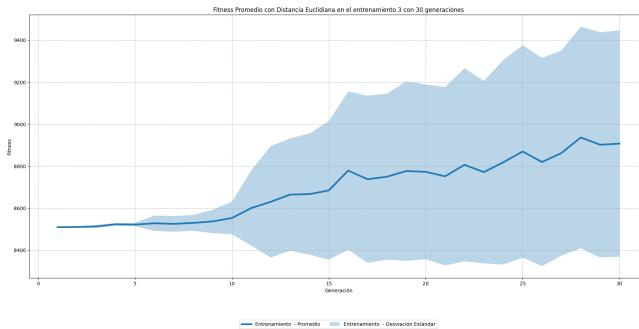


Figura 14: Fitness promedio y desviación individual para el entrenamiento 3 de 30 generaciones aplicando la distancia Euclidiana

El segundo fitness mas alto le corresponde al entrenamiento 5, con un valor de 9984, seguido de este entrenamiento, el primer entrenamiento obtuvo un fitness de 9981, con una desviación que se aproxima a los 9100, 100 puntos menos que la desviación correspondiente al quinto entrenamiento (Fig. S7B-E, S8B-E, S1B-E y S2B-E). Para el segundo entrenamiento la diferencia con respecto al primero es de 98 puntos por debajo, con un fitness máximo de 9883, y una desviación de 9000 (Fig. S3B-E y S4B-E). Por último, el cuarto entrenamiento obtuvo el fitness mas bajo con 8544, con una desviación que no supera los 8600 (Fig. S5B-E y S6B-E).

III-D2. 20 Generaciones: La gráfica general de las desviaciones y promedios obtenidos en estas simulaciones es:

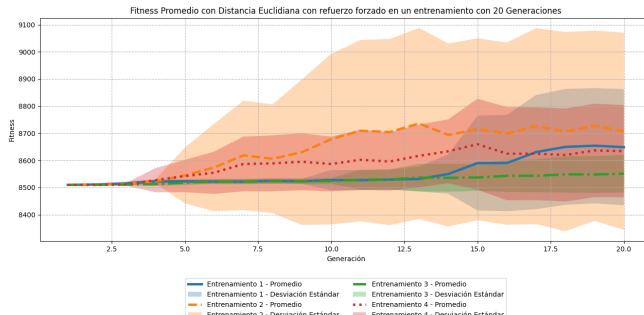


Figura 15: Gráfica de promedios y desviaciones fitness correspondiente a 5 simulaciones realizadas con una generación de 20

Como se puede observar, la desviación con mayor valor se obtuvo en el segundo entrenamiento, además su puntaje fitness fue el mas alto de todos los demás con un valor de 9881:

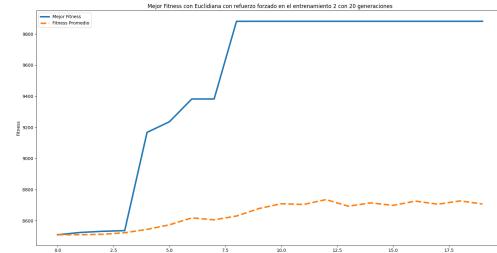


Figura 16: Fitness individual para el entrenamiento 2 de 20 generaciones aplicando la distancia Euclidiana

La gráfica de desviación es:

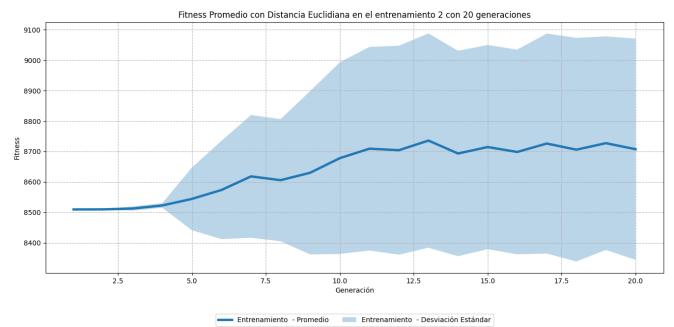


Figura 17: Fitness promedio y desviación individual para el entrenamiento 2 de 20 generaciones aplicando la distancia Euclidiana

Con un fitness maximo de 9632, el primer entrenamiento obtuvo el segundo mejor puntaje, con una desviación por encima de los 8800; para el cuarto entrenamiento el puntaje máximo fue de 9237,también sobre pasando los 8800, sin embargo el fitness promedio durante la generación 17 y 20 fue en un rango de 8600 y 8650 puntos (Fig. S1C-E, S2C-E, S5C-E y S6C-E). Por último, el quinto y tercer entrenamiento obtuvieron los fitness mas bajos con 8782 y 8773, con una desviación que no supera los 8750 (Fig. S7C-E, S8C-E, S3C-E y S4C-E).

III-E. Resultados de las simulaciones aplicando la distancia Manhattan

III-E1. 50 Generaciones: Principalmente se iniciaron las simulaciones para las 50 generaciones, de estas simulaciones se graficaron los mejores promedios y las desviaciones de cada entrenamiento, de esta manera la grafica que reúne cada uno de los entrenamientos es la Figura 18.

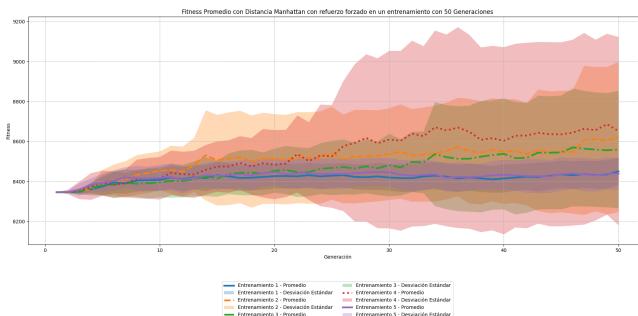


Figura 18: Gráfica de promedios y desviaciones fitness correspondiente a 5 simulaciones realizadas con una generación de 50 para la distancia Manhattan

En el entrenamiento número 4 se presentó el fitness mas alto con 9978 junto con las desviaciones mas altas de todos los entrenamientos, de al rededor de 9100 puntos:

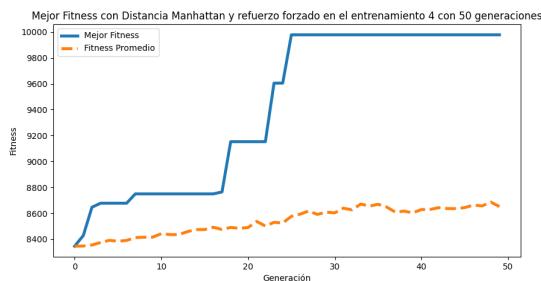


Figura 19: Fitness individual para el entrenamiento 4 de 50 generaciones aplicando la distancia Manhattan

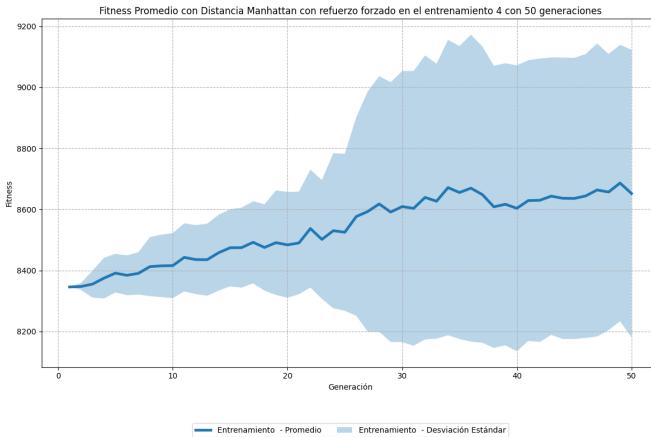


Figura 20: Fitness promedio y desviación individual para el entrenamiento 4 de 50 generaciones aplicando la distancia Manhattan

Los siguientes dos entrenamientos con los puntajes mas altos fueron el 2 y 3, con 9688 y 9621 respectivamente, con una desviación que no supera los 8900 para el entrenamiento numero 3, y 9000 para el entrenamiento numero 2 (Fig. S3A-M, S4A-M, S5A-M y S6A-M). Por último, el primer y quinto

entrenamiento obtuvieron los puntajes mas bajos, con 8767 y 8771, con una desviación que no supera los 8600 (Fig. S1A-M, S2A-M, S7A-M, S8A-M).

III-E2. 30 Generaciones: En la gráfica general para las simulaciones con 30 generaciones se observan los fitness promedio de cada entrenamiento y las desviaciones:

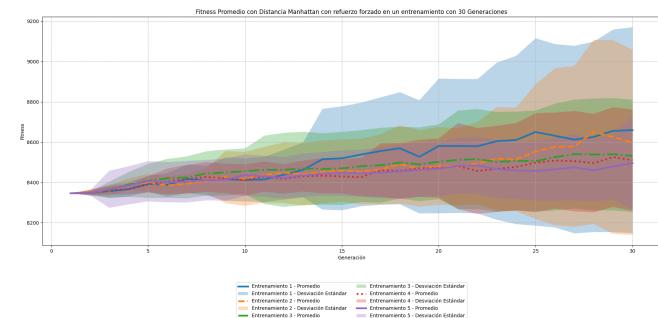


Figura 21: Gráfica de promedios y desviaciones para correspondiente a las generaciones de 30 para la distancia Manhattan

Los entrenamientos que más fitness obtuvieron durante estas simulaciones, fue el primer y segundo entrenamiento. El puntaje máximo para el primer entrenamiento es de 9990

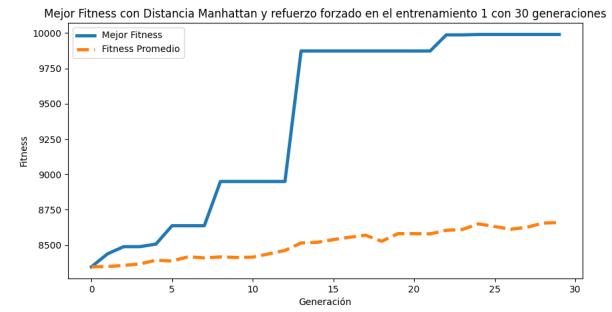


Figura 22: Fitness promedio y más alto para el entrenamiento 1 de 30 generaciones con la distancia Manhattan

La desviación para el primer entrenamiento ronda los 9100

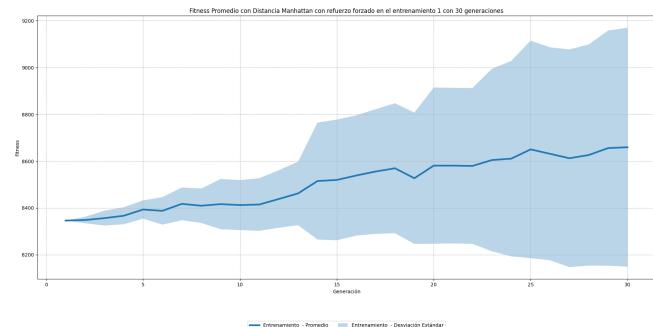


Figura 23: Fitness promedio y desviación individual para el entrenamiento 1 de 30 generaciones aplicando la distancia Manhattan

Para el segundo entrenamiento el puntaje máximo es de 9987, superando tambien los 9000 puntos en la desviación, sin embargo en la generación 28 hubo un decremento en el promedio de fitness obtenidos, todo lo contrario al primer entrenamiento (Fig. S1B-M y S2B-M). El siguiente mayor puntaje le corresponde al entrenamiento numero 5, este obtuvo un puntaje de 9630, con una desviación que supera los 8700 puntos a partir de la generación 29. Por último, el cuarto y tercer entrenamiento se obtuvieron los puntajes mas bajos, con 9623 y 9618, con una desviación que no supera los 8900 puntos (Fig. S3B-M, S4B-M, S5B-M y S6B-M).

III-E3. 20 Generaciones: En la gráfica general para las simulaciones con 20 generaciones se observan los fitness promedio de cada entrenamiento y las desviaciones:

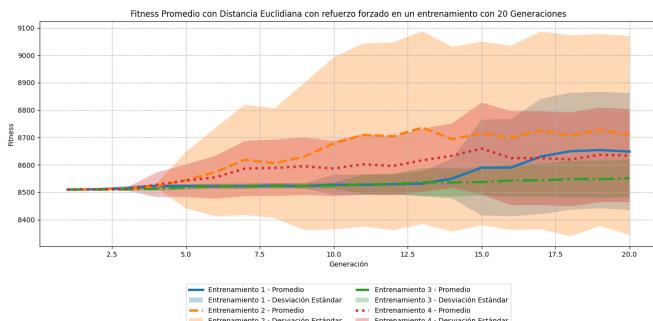


Figura 24: Fitness promedio y desviaciones para las simulaciones realizadas con 20 generaciones aplicando la distancia Manhattan

El entrenamiento número dos obtuvo el mejor fitness, con un puntaje de 9980, su gráfica es:

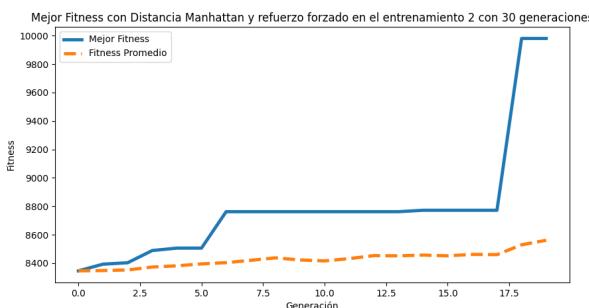


Figura 25: Fitness máximo y promedio para los entrenamientos realizados de 20 generaciones en la distancia Manhattan

Este entrenamiento también posee la mayor desviación de todas

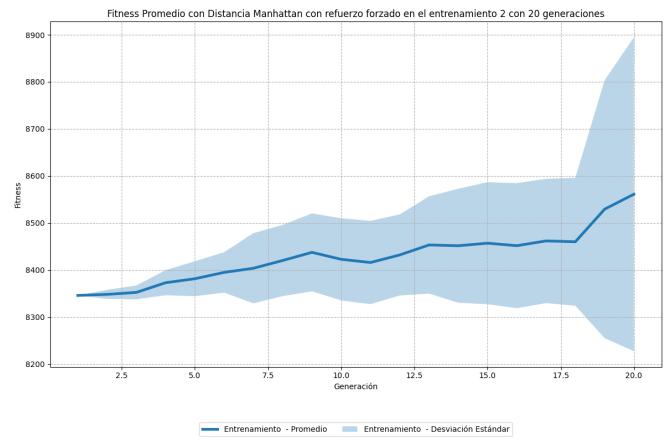


Figura 26: Fitness promedio y desviaciones para el segundo entrenamiento de 20 generaciones en la distancia Manhattan

El entrenamiento número 5 obtuvo el segundo mejor fitness, con un puntaje de 9360, con el pasar de cada generación el promedio aumenta, junto con la desviación. En la generación 29 a 30 el puntaje sobrepasó los 8800 (Fig. S7C-M y S8C-M). El cuarto entrenamiento obtuvo un puntaje de 8799, con una desviación máxima de 8600, en la generación 19 y 20 el promedio de fitness fue de 8600 puntos (Fig. S5C-M y S6C-M). Por último, el primer y tercer entrenamiento obtuvieron los puntajes mas bajos, con 8782 y 8771, con una desviación que sobrepasa los 8600 puntos en la generación numero 15. Por último los peores puntajes fueron obtenidos por el tercero y primer entrenamiento, con un puntaje de 8771, y 8766 respectivamente, ambas superan una desviación de 8600 puntos, de igualmanera comparten una disminución del fitness promedio en la generación 19 y 20 (Fig. S3C-M, S4C-M, S1C-M y S2C-M).

III-F. Resultados de las simulaciones aplicando la distancia de Chebyshev

III-F1. 50 Generaciones: De las simulaciones realizadas aplicando la distancia Chebyshev, se obtienen los mejores fitness y promedios, además de todas las desviaciones realizadas en estas

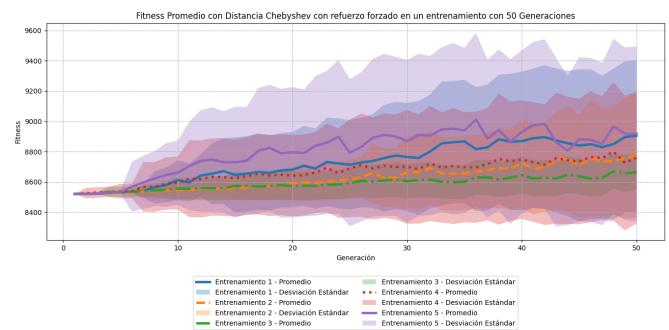


Figura 27: Fitness promedio y desviaciones para las simulaciones realizadas con 50 generaciones aplicando la distancia de Chebyshev

Analizando la gráfica, los valores fitness más altos corresponden al entrenamiento 1 y 5. Por un lado el entrenamiento número 1 presenta el mayor puntaje con 9988:

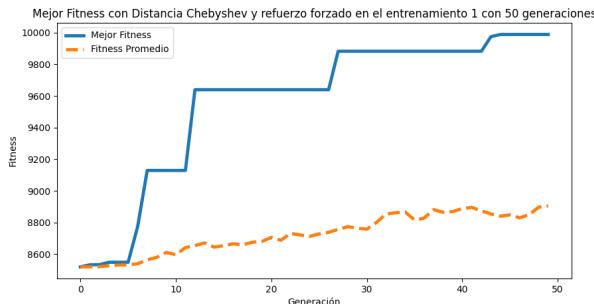


Figura 28: Mejor fitness para una generación de 50 aplicando la distancia de Chebyshev

Mientras que el puntaje para el entrenamiento 5 es de 9986

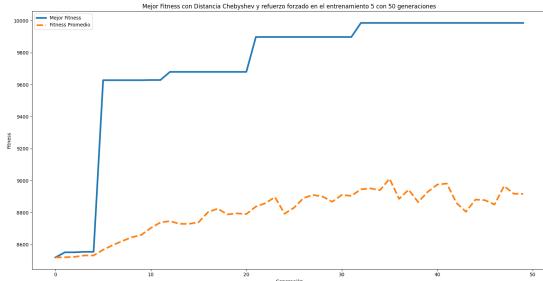


Figura 29: Mejor fitness para el entrenamiento 5 de una generación de 50 aplicando la distancia de Chebyshev

Sin embargo en este caso particular la mayor desviación la posee el entrenamiento número 5

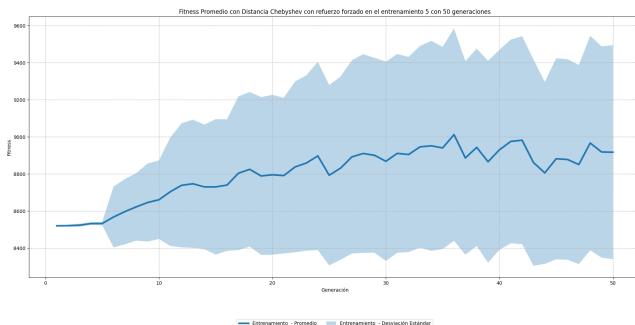


Figura 30: Mayor desviación en el entrenamiento numero 5, para una generación de 50 con la distancia de Chebyshev

Mientras que el entrenamiento número 1 obtuvo una desviación de 9400 puntos en la generación 50, el entrenamiento número 5 sobrepasa esa desviación a partir de la generación 30. Por otro lado, el entrenamiento número 4, obtuvo el tercer mejor fitness con 9984, acercándose a 9200 puntos en la

desviación máxima para la generación 48. Los menores fitness obtenidos fueron para el segundo y tercer entrenamiento, con 9973 y 8777 respectivamente, ambos con una desviación que no supera los 9200 puntos (Fig. S2A-C, S7A-C, S8A-C, S3A-C, S4A-C, S5A-C y S6A-C).

III-F2. 30 Generaciones: La gráfica general que representa a todos los entrenamientos realizados para la generación de 30 es:

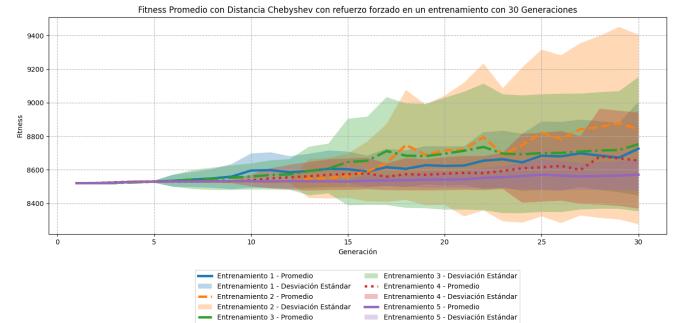


Figura 31: Fitness promedios y desviaciones de cada entrenamiento realizado para las generaciones de 30 con la distancia de Chebyshev

Tanto el entrenamiento numero 2 y 3 presenta el fitness más alto, el segundo entrenamiento posee un valor fitness máximo de 9994, mientras que el tercero entrena uno de 9974

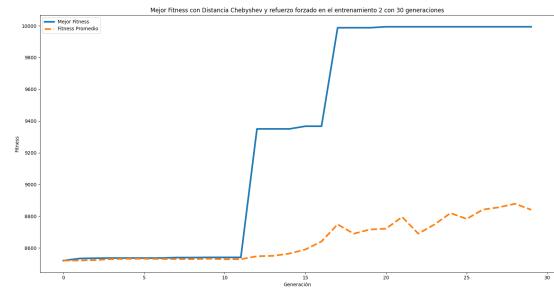


Figura 32: Fitness máximo y promedio del segundo entrenamiento para 30 generaciones de la distancia Shebyshev

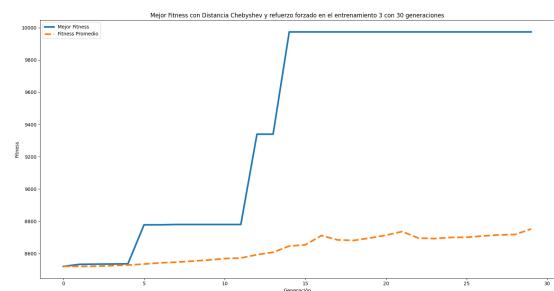


Figura 33: Fitness máximo y promedio del tercer entrenamiento para 30 generaciones de la distancia Shebyshev

La mayor desviación le corresponde al entrenamiento número dos, su gráfica es:

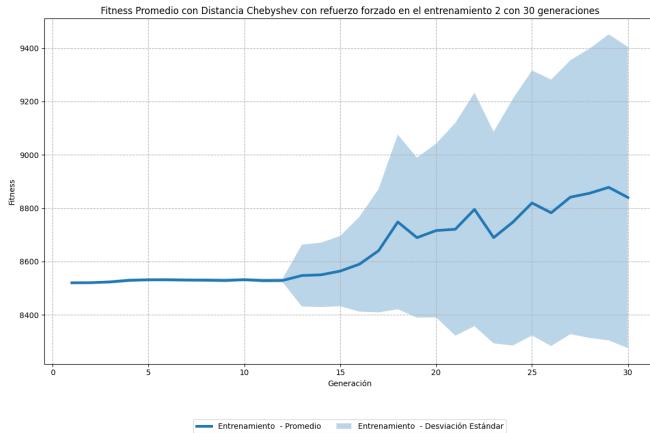


Figura 34: Mayor desviación para las simulaciones realizadas con las distancia de Chebyshev y 30 generaciones

El entrenamiento numero 3 apenas supera el puntaje de desviación 9100 en su última generación, pero tambien en esta última aumenta ligeramente su promedion em comparacion con el segundo entrenamiento. Por otro lado, el cuarto entrenamiento obtuvo el tercer mejor fitness con 9884, con una desviación que no supera los 9000 puntos en la generación 30. Los fitness mas bajos se los llevan el primer y quinto entrenamiento, con 9665 puntos y 8787 respectivamente, ambbos con aumento en el fitnes promedios en la ultima generación (Fig. S3B-C, S4B-C, S5B-C, S6B-C, S7B-C, S8B-C y S9B-C).

III-F3. 20 Generaciones: En la gráfica general para las simulaciones con 20 generaciones se observan los fitness promedio de cada entrenamiento y las desviaciones:

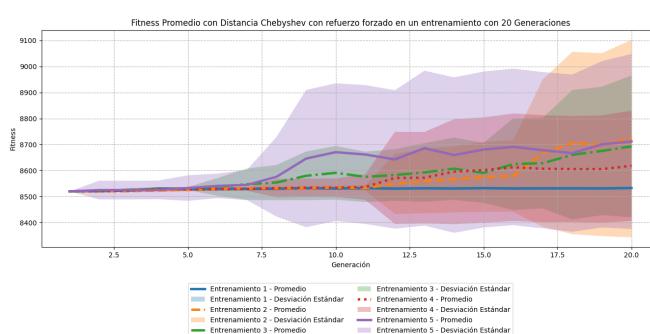


Figura 35: Fitness y desviaciones generales de los entrenamientos realizados para una generación de 20 con la distancia Chebyshev

El entrenamiento con mayor fitness en este caso también posee la mayor desviación, el entrenamiento número dos obtuvo un fitness máximo de 9990

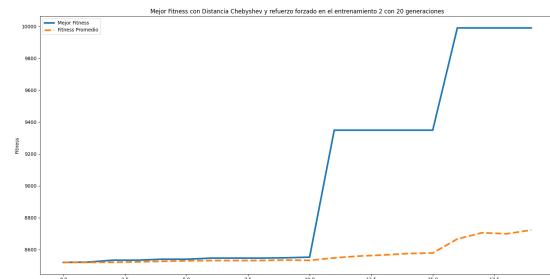


Figura 36: Fitnes máximo y promedio para una generación de 20 con la distancia de Chebyshev

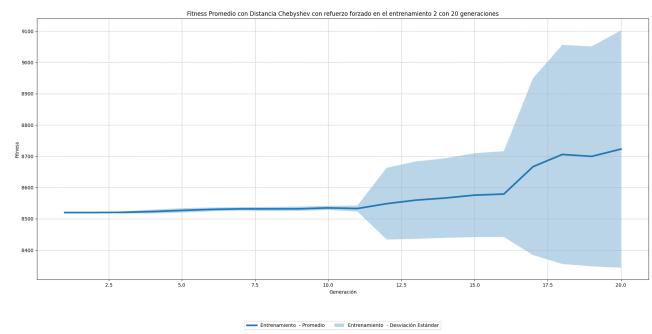


Figura 37: Fitnes promedio y desviaciones del entrenamiento 2 correspondiente a la distancia Chebyshev para una generación de 20

El entrenamiento número 3 obtuvo el segundo mejor resultado con un puntaje fitness máximo de 9900

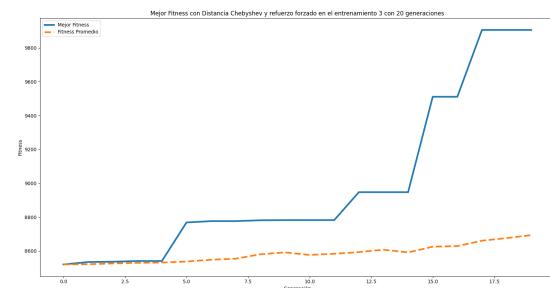


Figura 38: Fitnes máximo y promedio para una generación de 20 con la distancia de Chebyshev en el tercer entrenamiento

Además el quinto entrenamiento obtuvo 9882 como puntaje máximo fitness, los fitness mas bajos se los llevan el primer y cuarto entrenamiento, con 8543 y 9680 respectivamente. Sin embargo estos últimos tres entrenamientos en la ultima gneración incrementaron el fitness promedio (Fig. S1C-C, S2C-C, S5C-C, S4C-C, S6C-C, S7C-C, S8C-C).

III-G. 50 generaciones sin el Refuerzo forzado o Método de aceleración

Para esta sección solo se realizó un entrenamiento con el fin de ver que tan optimo era realizar las simulaciones sin la aplicación del método descrito en la Figura 6. En la aplicación de la distancia Euclidiana el mayor fitness obtenido es de 8536

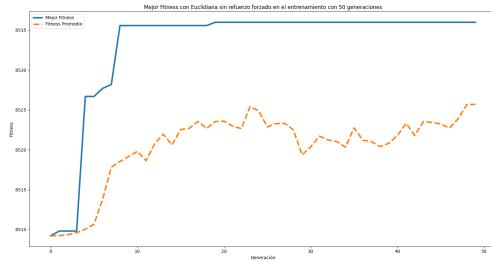


Figura 39: Mejor promedio fitness para la distancia Euclidiana sin la aplicación del Refuerzo forzado

por otra parte la desviación obtenida se puede observar en:

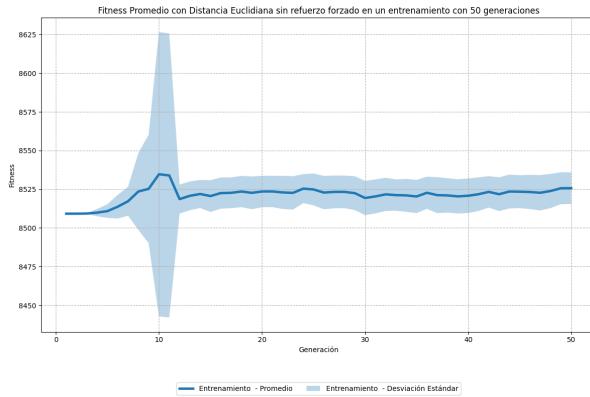


Figura 40: Desviación y promedio fitness para la distancia Euclidiana sin la aplicación del Refuerzo forzado

Donde la mayor desviación ocurrió en la generación número 10.

Implementando la distancia Manhattan el puntaje solo se diferencia en 1 de la distancia Euclidiana, con un puntaje de 8535

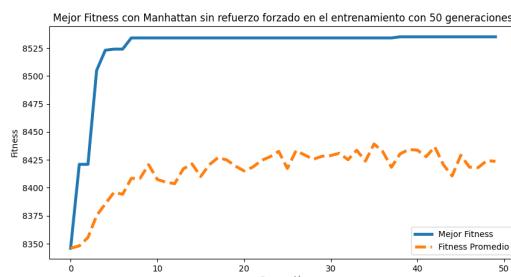


Figura 41: Mejor promedio fitness para la distancia Manhattan sin la aplicación del Refuerzo forzado

De igual manera, Manhattan también presenta la mayor desviación en la décima generación

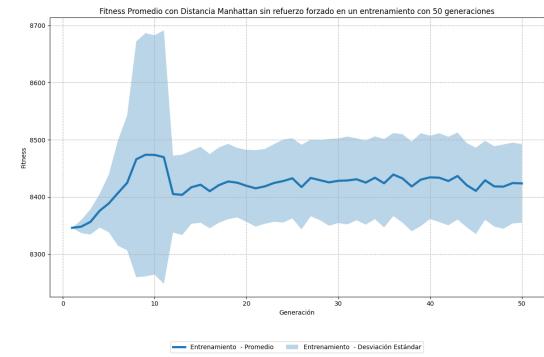


Figura 42: Desviación y promedio fitness para la distancia Manhattan sin la aplicación del Refuerzo forzado

Por ultimo, en esta sección la distancia de Chebyshev presentó los mejores resultados, con un puntaje fitness máximo obtenido de 9885, y una desviación que ronda los 8800 puntos.



Figura 43: Mejor promedio fitness para la distancia Chebyshev sin la aplicación del Refuerzo forzado

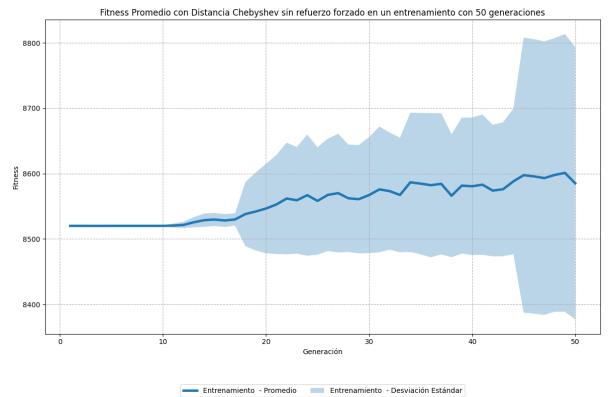


Figura 44: Desviación y promedio fitness para la distancia Chebyshev sin la aplicación del Refuerzo forzado

III-H. Simulaciones de 50 generaciones realizadas en el segundo mapa

Se realizaron 5 simulaciones de 50 generaciones para cada sistema de recompensas en el segundo mapa ver figura 7, Euclidian, Manhattan y Chebyshev. Con los datos recopilados

de estas simulaciones, se analizan las gráficas generales, y basándose en los resultados previos se hace una descripción

III-H1. Distancia Euclíadiana: La gráfica general obtenida después de cada entrenamiento para la distancia Euclíadiana es:

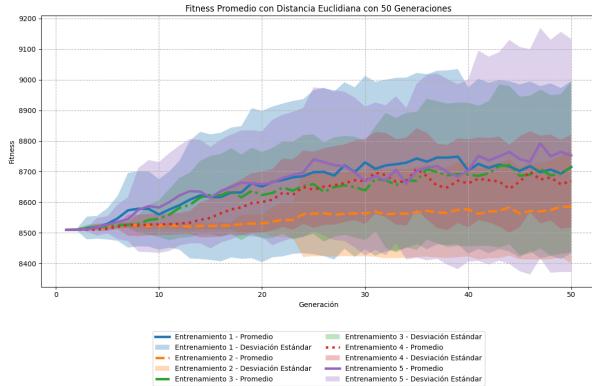


Figura 45: Gráfica de promedios fitness generales y desviaciones en cada entrenamiento para la distancia Euclíadiana correspondiente a una generación de 50 y el segundo mapa

III-H2. Distancia Manhattan: La gráfica general obtenida después de cada entrenamiento para la distancia Manhattan es:

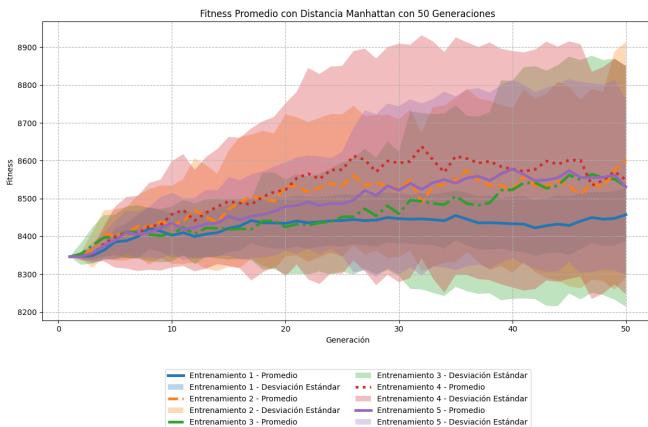


Figura 46: Gráfica de promedios fitness generales y desviaciones en cada entrenamiento para la distancia Manhattan correspondiente a una generación de 50 y el segundo mapa

III-H3. Distancia Chebyshev: La gráfica general obtenida después de cada entrenamiento para la distancia Chebyshev es:

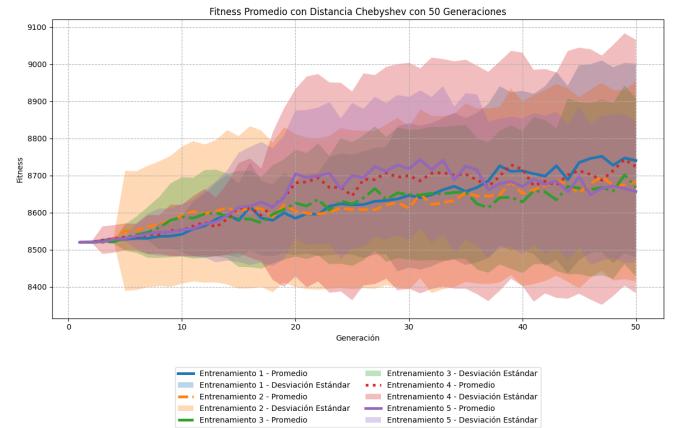


Figura 47: Gráfica de promedios fitness generales y desviaciones en cada entrenamiento para la distancia Chebyshev correspondiente a una generación de 50 y el segundo mapa

IV. CONCLUSIONES

La implementación de la librería NEAT para un vehículo autónomo permitió analizar distintos aspectos del aprendizaje y desempeño del agente en diversos entornos. En este caso se realizaron dos pruebas, el entorno podría encontraba libre de obstáculos, mientras que en la otra se añadió un bloqueo de ruta. Se observó que el agente mostró un mejor rendimiento en el entorno libre de obstáculos, en comparación del entorno con la ruta bloqueada. Al agente le tomaba más generaciones aprender en comparación cuándo algunas de las vías era obstruida. En cuanto a las métricas de distancia, la distancia Chebyshev demostró ser más óptima que la Manhattan y Euclíadiana en entornos libres de obstáculos. Sin embargo, en escenarios con obstáculos, la distancia Euclíadiana ofreció un mejor desempeño que las demás obteniendo un alto valor fitness. La distancia Euclíadiana aumentaba al igual que la de Chebyshev con el pasar de las generaciones, sin embargo la distancia Manhattan obtuvo su mayor fitness con una generación de 30. Con los datos obtenidos en la sección de resultados, se puede que puede concluir que para la distancia Manhattan y Chebyshev con 30 generaciones ya se obtienen unos excelentes puntajes, ya que para Chebyshev las simulaciones realizadas con 50 generaciones solo le aumentaron en dos.

Por otro lado, la implementación del refuerzo forzado generó mejoras importantes, ya que disminuyó el tiempo requerido para que el agente alcanzara un fitness significativo. Además, permitiendo que el agente aprendiera a girar en pocas generaciones.

REFERENCIAS

- [1] Organización Mundial de la Salud, *Informe sobre la situación mundial de la seguridad vial 2023*. Ginebra, Suiza: Organización Mundial de la Salud, 2023. dirección: <https://www.who.int/publications/item/9789241565684>.

- [2] S. Arshad, M. Sualeh, D. Kim, D. V. Nam y G.-W. Kim, «Clothoid: An Integrated Hierarchical Framework for Autonomous Driving in a Dynamic Urban Environment,» *English, Sensors*, vol. 20, n.º 18, pág. 5053, 2020, ISSN: 1424-8220. DOI: [10.3390/s20185053](https://doi.org/10.3390/s20185053). dirección: <https://research.ebsco.com/linkprocessor/plink?id=0d501679-ad5e-3650-956f-f38dba4076eb>.
- [3] Z. Lin et al., «Policy Iteration Based Approximate Dynamic Programming Toward Autonomous Driving in Constrained Dynamic Environment,» *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, n.º 5, págs. 5003-5013, 2023. DOI: [10.1109/TITS.2023.3237568](https://doi.org/10.1109/TITS.2023.3237568).
- [4] C. Hu, Z. Wang, X. Bu, J. Zhao, J. Na y H. Gao, «Optimal Tracking Control for Autonomous Vehicle With Prescribed Performance via Adaptive Dynamic Programming,» *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, n.º 9, págs. 12 437-12 449, 2024. DOI: [10.1109/TITS.2024.3384113](https://doi.org/10.1109/TITS.2024.3384113).
- [5] X. Han, X. Zhao, X. Xu, C. Mei, W. Xing y X. Wang, «Trajectory tracking control for underactuated autonomous vehicles via adaptive dynamic programming,» *Journal of the Franklin Institute*, vol. 361, n.º 1, págs. 474-488, 2024, ISSN: 0016-0032. DOI: <https://doi.org/10.1016/j.jfranklin.2023.12.003>. dirección: <https://www.sciencedirect.com/science/article/pii/S0016003223007676>.
- [6] N. Xu, Z. Shi, S. Yin y Z. Xiang, «A hyper-heuristic with deep Q-network for the multi-objective unmanned surface vehicles scheduling problem,» *Neurocomputing*, vol. 596, pág. 127 943, 2024, ISSN: 0925-2312. DOI: <https://doi.org/10.1016/j.neucom.2024.127943>. dirección: <https://www.sciencedirect.com/science/article/pii/S0925231224007148>.
- [7] V. Mnih et al., «Playing Atari with Deep Reinforcement Learning,» *CoRR*, vol. abs/1312.5602, 2013. arXiv: [1312.5602](https://arxiv.org/abs/1312.5602). dirección: <http://arxiv.org/abs/1312.5602>.
- [8] W. Zhang, J. Gai, Z. Zhang, L. Tang, Q. Liao e Y. Ding, «Double-DQN based path smoothing and tracking control method for robotic vehicle navigation,» *Computers and Electronics in Agriculture*, vol. 166, pág. 104 985, 2019, ISSN: 0168-1699. DOI: <https://doi.org/10.1016/j.compag.2019.104985>. dirección: <https://www.sciencedirect.com/science/article/pii/S0168169919302066>.
- [9] X. Sun, H. Dou y Z. Zhou, «Highly accurate map construction and deep Q-network for autonomous driving and smart transportation,» *Computers and Electrical Engineering*, vol. 110, pág. 108 899, 2023, ISSN: 0045-7906. DOI: <https://doi.org/10.1016/j.compeleceng.2023.108899>. dirección: <https://www.sciencedirect.com/science/article/pii/S0045790623003233>.
- [10] J. Nie, G. Zhang, X. Lu, H. Wang, C. Sheng y L. Sun, «Obstacle avoidance method based on reinforcement learning dual-layer decision model for AGV with visual perception,» *Control Engineering Practice*, vol. 153, pág. 106 121, 2024, ISSN: 0967-0661. DOI: <https://doi.org/10.1016/j.conengprac.2024.106121>. dirección: <https://www.sciencedirect.com/science/article/pii/S0967066124002806>.
- [11] D. Xue, D. Wu, A. S. Yamashita y Z. Li, «Proximal policy optimization with reciprocal velocity obstacle based collision avoidance path planning for multi-unmanned surface vehicles,» *Ocean Engineering*, vol. 273, pág. 114 005, 2023, ISSN: 0029-8018. DOI: <https://doi.org/10.1016/j.oceaneng.2023.114005>. dirección: <https://www.sciencedirect.com/science/article/pii/S002980182300389X>.
- [12] L. M. Leandro Parada Eduardo Candela y P. Angeloudis, «Safe and efficient manoeuvring for emergency vehicles in autonomous traffic using multi-agent proximal policy optimisation,» *Transportmetrica A: Transport Science*, vol. 0, n.º 0, págs. 1-29, 2023. DOI: [10.1080/23249935.2023.2246586](https://doi.org/10.1080/23249935.2023.2246586).
- [13] H. Guo, Z. Ren, J. Lai, Z. Wu y S. Xie, «Optimal navigation for AGVs: A soft actor–critic-based reinforcement learning approach with composite auxiliary rewards,» *Engineering Applications of Artificial Intelligence*, vol. 124, pág. 106 613, 2023, ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2023.106613>. dirección: <https://www.sciencedirect.com/science/article/pii/S0952197623007972>.
- [14] S. Wen, Y. Shu, A. Rad, Z. Wen, Z. Guo y S. Gong, «A deep residual reinforcement learning algorithm based on Soft Actor-Critic for autonomous navigation,» *Expert Systems with Applications*, vol. 259, pág. 125 238, 2025, ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2024.125238>. dirección: <https://www.sciencedirect.com/science/article/pii/S0957417424021055>.
- [15] L. Luo, N. Zhao, Y. Zhu e Y. Sun, «A* guiding DQN algorithm for automated guided vehicle pathfinding problem of robotic mobile fulfillment systems,» *Computers & Industrial Engineering*, vol. 178, pág. 109 112, 2023, ISSN: 0360-8352. DOI: <https://doi.org/10.1016/j.cie.2023.109112>. dirección: <https://www.sciencedirect.com/science/article/pii/S0360835223001365>.
- [16] K. Sivayazi y G. Mannayee, «Modeling and simulation of a double DQN algorithm for dynamic obstacle avoidance in autonomous vehicle navigation,» *e-Prime - Advances in Electrical Engineering, Electronics and Energy*, vol. 8, pág. 100 581, 2024, ISSN: 2772-6711. DOI: <https://doi.org/10.1016/j.prime.2024.100581>. dirección: <https://www.sciencedirect.com/science/article/pii/S277267112400161X>.
- [17] M. E. Yuksel, «Agent-based evacuation modeling with multiple exits using NeuroEvolution of Augmenting Topologies,» *Advanced Engineering Informatics*, vol. 35, págs. 30-55, 2018, ISSN: 1474-0346. DOI: <https://doi.org/10.1016/j.aei.2017.11.003>. dirección: <https://www.sciencedirect.com/science/article/pii/S1474034616304281>.

- [18] T. J. Ikonen e I. Harjunkoski, «Decision-making of online rescheduling procedures using neuroevolution of augmenting topologies,» en *29th European Symposium on Computer Aided Process Engineering*, ép. Computer Aided Chemical Engineering, A. A. Kiss, E. Zondervan, R. Lakerveld y L. Özkan, eds., vol. 46, Elsevier, 2019, págs. 1177-1182. DOI: <https://doi.org/10.1016/B978-0-12-818634-3.50197-1>. dirección: <https://www.sciencedirect.com/science/article/pii/B9780128186343501971>.
- [19] F. C. Guardo, «Evolución de redes neuronales mediante topologías aumentadas,» Trabajo Fin de Grado, Universidad Autónoma de Madrid, Madrid, España, jun. de 2019.
- [20] I. Lamouik, A. Yahyaouy y M. A. Sabri, «Smart multi-agent traffic coordinator for autonomous vehicles at intersections,» en *2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, 2017, págs. 1-6. DOI: [10.1109/ATSIP.2017.8075564](https://doi.org/10.1109/ATSIP.2017.8075564).
- [21] P. R. M. Araujo, E. Mounier, E. Dawson y A. Noureddin, «Smart Mobility: Leveraging Perception Sensors for Map-Based Navigation in Autonomous Vehicles,» en *2024 IEEE International Conference on Smart Mobility (SM)*, 2024, págs. 281-286. DOI: [10.1109/SM63044.2024.10733404](https://doi.org/10.1109/SM63044.2024.10733404).
- [22] J. Portilla, «A Beginner’s Guide to Neural Networks in Python,» *Springboard Blog*, 2017. dirección: <https://www.springboard.com/blog/data-science/beginners-guide-neural-network-in-python-scikit-learn-0-18/>.
- [23] R. Lauckner y H. Kolivand, «NEAT Algorithm in Autonomous Vehicles,» 2023, Preprint, not peer-reviewed. dirección: <https://ssrn.com/abstract=4644203>.
- [24] L. CodeReclaimers, *Overview of the basic XOR example (evolve-feedforward.py)*, Accessed: 2024-11-18, 2019. dirección: https://neat-python.readthedocs.io/en/latest/xor_example.html#fitness-function.