Escuela
Politécnica
Superior

Algoritmos genéticos para conducción autónoma

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
José Carlos Zambrana Navajas
Tutor/es:
Miguel Ángel Cazorla Quevedo
Francisco Gómez Donoso

Mayo 2023

Universitat d'Alacant
Universidad de Alicante

Algoritmos genéticos para conducción autónoma

Autor
José Carlos Zambrana Navajas

Tutor/es
Miguel Ángel Cazorla Quevedo
Departamento de ciencia de la computación e inteligencia artificial
Francisco Gómez Donoso
Departamento de ciencia de la computación e inteligencia artificial

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Mayo 2023

Resumen

Este Trabajo Final de Grado (TFG) tiene como foco principal el estudio y desarrollo de sistemas autónomos de navegación, utilizando una combinación de redes neuronales artificiales y algoritmos genéticos. El documento detalla nuestra exploración en profundidad del concepto y la aplicación de un algoritmo genético clásico y de la Neuroevolución de Topologías Aumentadas (NEAT), una técnica específica que aplica algoritmos genéticos a la arquitectura y pesos de redes neuronales artificiales.

A lo largo de este trabajo, se ha hecho uso de diversas herramientas y plataformas de software que facilitan la simulación y el análisis de la navegación autónoma, como ROS, Gazebo y Unity. Además, se diseñó y desarrolló un controlador de robots que emplea el modelo Turtlebot3. Este proceso implicó generar entornos de simulación complejos, la creación y programación de controladores de robots, y la implementación de algoritmos genéticos con el fin de optimizar la eficacia de las redes neuronales.

Los resultados de la investigación se obtuvieron a través de pruebas en diversos entornos, incluyendo tanto un circuito como una ciudad con objetos estáticos. Los datos resultantes muestran un rendimiento notablemente satisfactorio en cada caso.

El estudio concluye que la combinación de algoritmos genéticos y redes neuronales, y en particular la aplicación de la técnica NEAT, tiene el potencial de ofrecer soluciones robustas y eficientes para los retos presentes en la navegación autónoma. Se espera que estos hallazgos sirvan como punto de partida para futuras investigaciones y desarrollos en el campo de los vehículos autónomos.

Agradecimientos

Este trabajo no habría sido posible sin el apoyo de todas las personas que han contribuido a mi formación académica y personal.

En primer lugar, quisiera agradecer a mis padres, por su incondicional amor y apoyo a lo largo de todos estos años. Su fe en mí y sus innumerables sacrificios han sido la fuente de mi perseverancia y determinación.

A mis tutores, su experiencia y guía me han sido invaluables en este viaje. Su pasión por el conocimiento y su dedicación a la enseñanza me han inspirado profundamente.

No quiero olvidarme de los compañeros de carrera con los que he compartido este viaje universitario. Juntos, hemos enfrentado desafíos, compartido triunfos y aprendido unos de otros. Gracias por hacer estos cuatro años de suplicio un poco más amenos.

A mi profesora de primaria, Milagros, su entusiasmo y dedicación en la enseñanza dejaron una impresión duradera en mí. Su influencia me ha acompañado a lo largo de mis años de estudio y ha contribuido a mi amor por el aprendizaje.

A mi profesor Eusebio, gracias por su apoyo constante y su inspiradora actitud hacia la enseñanza. Su influencia me ha impulsado a esforzarme por la excelencia.

A mis amigos, quiero agradecerles por estar siempre allí, por compartir risas y palabras de aliento, por ser una fuente constante de alegría y apoyo.

Por último, pero no menos importante, quiero expresar mi más profundo agradecimiento a mi pareja. Tu cariño, paciencia y apoyo han sido mi roca en los momentos más difíciles.

"La inteligencia es
la habilidad de adaptarse
a los cambios."

Stephen Hawking.

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

En la actualidad, la conducción autónoma representa una de las fronteras más desafiantes y prometedoras en el campo de la tecnología y la ingeniería. Su potencial para transformar la movilidad y mejorar la seguridad vial es indudable. No obstante, la completa autonomía, donde los vehículos pueden operar de manera segura y eficiente sin intervención humana en cualquier entorno y situación, sigue siendo un desafío formidable. Aquí radica la necesidad de explorar y desarrollar algoritmos robustos y versátiles que puedan manejar la complejidad y la incertidumbre inherentes a la conducción autónoma.

El desarrollo de robots autónomos ha experimentado un rápido progreso en los últimos años, abriendo un abanico de posibilidades para su aplicación en diferentes contextos. Uno de estos contextos es la navegación autónoma en entornos peatonales, que plantea desafíos significativos debido a su complejidad e imprevisibilidad. Por tanto, la necesidad de desarrollar y perfeccionar algoritmos que puedan manejar la complejidad y la incertidumbre de estas tareas es imprescindible.

Los algoritmos genéticos, concebidos por primera vez por John Holland en la década de 1970[3] e inspirados en los principios de la selección natural y la evolución, han demostrado ser eficaces en la resolución de problemas complejos y la optimización de soluciones en una amplia variedad de campos[17]. En el contexto de la conducción autónoma, pueden emplearse para optimizar las decisiones de control y adaptar el comportamiento del vehículo a diferentes condiciones y entornos.

Sin embargo, la aplicación de estos algoritmos a las redes neuronales artificiales, una estrategia menos explorada pero prometedora, puede aportar mejoras significativas en la capacidad de aprendizaje y adaptación de los robots a entornos desafiantes[12]. Estas redes neurales están inspiradas en el funcionamiento del cerebro humano y son capaces de aprender y adaptarse a partir de los datos recogidos por los sensores del vehículo y predecir el comportamiento adecuado en función de los datos de entrada.

Este Trabajo Final de Grado (TFG) se centra en la combinación de algoritmos genéticos y redes neuronales para el desarrollo de robots autónomos capaces de navegar de manera efectiva en entornos peatonales. Se utiliza el algoritmo de Neuroevolución de Topologías Aumentadas (NEAT) [18], que combina estos dos enfoques para optimizar tanto la estructura como los pesos de la red neuronal.

Para implementar y evaluar el sistema propuesto, se utilizan herramientas de software como ROS[9], Gazebo[8] y Unity[13], que permiten la comunicación entre los diferentes componentes del sistema, la simulación de entornos realistas y el control de los vehículos autónomos.

Este trabajo busca contribuir a la evolución de la conducción autónoma, abriendo nuevas vías de investigación y desarrollo en la intersección de algoritmos genéticos y redes neuronales, y proporcionando una solución innovadora y efectiva para optimizar la capacidad de navegación y adaptabilidad de los robots en entornos peatonales.

Este documento se divide en siete secciones clave que abordan desde los fundamentos teóricos como las redes neuronales artificiales, los algoritmos genéticos y la navegación autónoma en la sección 2, pasando por la definición de objetivos en la sección 3 y la descripción de la metodología y herramientas utilizadas en la sección 4. El núcleo del trabajo se desarrolla en la sección 5, donde se presenta la implementación y desarrollo del proyecto, incluyendo la generación de entornos, la estrategia de control de robots y la aplicación de algoritmos genéticos. Posteriormente, en la sección 6 se exponen los resultados obtenidos de las pruebas realizadas, y finalmente, en la sección 7 se presentan las conclusiones y reflexiones finales sobre el trabajo realizado y las posibles direcciones futuras de la investigación.

2 Marco Teórico

Para entender completamente el alcance de la investigación presentada en este Trabajo de Fin de Grado, resulta esencial adentrarse en los conceptos clave y las teorías que forman la base de la metodología y enfoque adoptados.

Este marco teórico establece una base sólida para la comprensión del desarrollo y los resultados de la investigación, facilitando así una valoración más profunda de la contribución de este trabajo al campo de la navegación autónoma.

2.1 Redes neuronales artificiales

Las redes neuronales artificiales (RNA)[4] son modelos computacionales inspirados en la estructura y funcionamiento del cerebro humano, diseñadas para aprender y adaptarse a partir de datos de entrada. Las RNA están compuestas por unidades de procesamiento denominadas neuronas, las cuales están organizadas en capas. Dichas neuronas se conectan entre sí a través de elementos llamados sinapsis, que simulan las conexiones axonales en un cerebro biológico. Cada sinapsis tiene dos propiedades esenciales: un peso, que determina la fortaleza de la conexión, y un umbral, que controla la activación de la neurona destino.

2.1.1 Arquitecturas y tipos de redes neuronales

Existen diversas arquitecturas y tipos de RNA, entre las más comunes se encuentran:

- Redes feedforward: La información fluye en una única dirección desde la capa de entrada hasta la capa de salida, sin bucles. Este es el tipo de redes que se va a utilizar para el desarrollo de este proyecto.
- Redes recurrentes: Permiten conexiones entre neuronas en la misma capa o en capas anteriores, lo que permite modelar relaciones temporales y secuenciales en los datos.
- Redes convolucionales: Especialmente diseñadas para trabajar con datos estructurados en forma de rejilla, como imágenes, mediante la aplicación de filtros convolucionales.

2.1.2 Aprendizaje y entrenamiento

El aprendizaje en las RNA implica ajustar los pesos de las conexiones entre las neuronas para minimizar una función de error o pérdida que mide la diferencia entre las predicciones de la red y los valores reales. El algoritmo de *backpropagation* es uno de los métodos más utilizados para entrenar redes feedforward.

2.2 Algoritmos genéticos

Los Algoritmos genéticos (AAGG)[3] son técnicas de optimización y búsqueda inspiradas en la teoría de la evolución y genética natural. Trabajan con una población de soluciones candidatas codificadas como cromosomas, que evolucionan a lo largo de varias generaciones mediante operadores genéticos como selección, cruce y mutación.

2.2.1 Conceptos básicos

Los conceptos básicos de los AAGG incluyen:

- Población: Conjunto de soluciones candidatas, cada una de las cuales representa un cromosoma.
- Cromosoma: Representación codificada de una solución al problema.
- Gen: Elemento básico de información en un cromosoma.
- Fitness: Medida de la calidad o adaptación de una solución al problema.

2.2.2 Operadores genéticos y tipos de reproducción

Los operadores genéticos son procesos aplicados a la población para generar nuevas soluciones. Al mismo tiempo, es esencial mencionar que existen diferentes formas en las que la reproducción puede ocurrir en un algoritmo genético. Estas incluyen la reproducción sexual y asexual.

La reproducción sexual implica la combinación de información genética de dos progenitores para formar un descendiente. Esto a menudo implica operadores genéticos como el cruce o "crossover". Por otro lado, la reproducción asexual genera descendientes que son copias genéticas exactas del progenitor, lo que esencialmente implica operadores como la clonación y la mutación.

Los operadores genéticos comprenden:

- Selección: Escoge individuos de la población actual según su fitness para formar la siguiente generación.
- Cruce (crossover): Combina partes de dos cromosomas seleccionados para generar nuevos cromosomas.
- Mutación: Altera uno o varios genes de un cromosoma de manera aleatoria.

2.2.3 Selección y evolución

El proceso de selección y evolución en un Algoritmo genético (AG) sigue los siguientes pasos:

1. Inicializar una población aleatoria.
2. Evaluar el fitness de cada individuo de la población.
3. Seleccionar los mejores individuos, en función de su fitness, para la reproducción.
4. Aplicar operadores de cruce y mutación para generar nuevos individuos.
5. Evaluar el fitness de los nuevos individuos.
6. Reemplazar parte o toda la población actual con los nuevos individuos.
7. Repetir los pasos 3-6 hasta alcanzar un criterio de parada, como un número máximo de generaciones o un valor deseado de fitness.

2.3 Neuroevolución de Topologías Aumentadas (NEAT)

NEAT es un método para la evolución de RNA que se aplica tanto a la estructura de la red como a los pesos de las conexiones neuronales. Fue desarrollado por Kenneth Stanley durante su tesis doctoral en 2002[18] como un enfoque para superar algunas de las dificultades presentes en los métodos previos de neuroevolución.

Los métodos de neuroevolución anteriores a esta técnica a menudo comenzaban con una red neuronal de topología fija y tamaño determinado, y solo permitían la optimización de los pesos de la red. Sin embargo, este enfoque puede limitar la capacidad de las redes para evolucionar estructuras más complejas y eficientes. Por otro lado, si se permite la evolución de la topología desde el principio, el espacio de búsqueda puede ser tan vasto que el algoritmo de evolución puede tener dificultades para encontrar soluciones eficientes en un tiempo razonable.

La técnica propuesta por Stanley aborda estos problemas introduciendo una serie de innovaciones. Primero, comienza con redes de la forma más sencilla posible: sin conexiones ocultas. Esto limita el tamaño inicial del espacio de búsqueda, lo que facilita la búsqueda de soluciones inicialmente. A medida que evoluciona la población de redes, esta metodología introduce gradualmente nuevas estructuras y conexiones a través de un proceso llamado mutación estructural.

Además, este método introduce el concepto de ”especiación” para proteger las innovaciones topológicas. A medida que se introducen nuevas estructuras, las redes se agrupan en especies basadas en la similitud topológica. Esto permite a las nuevas innovaciones tener un cierto ”tiempo de protección” en el que pueden optimizarse antes de tener que competir con el resto de la población de redes.

En tercer lugar, el enfoque de Stanley utiliza una forma de codificación genética llamada ”genes de historia” que permite el seguimiento y la alineación correcta de genes a lo largo de generaciones, lo que facilita el cruce genético (o ”recombinación”) entre redes de diferentes topologías. Los genes de historia también permiten a este método identificar y proteger las innovaciones, ya que cada nuevo gen introducido por una mutación estructural se marca con un número de innovación global único.

Por último, el método propone una forma de combinar la evolución de la topología y los pesos de la red. Durante el proceso de evolución, se alternan las fases de optimización de los pesos y de introducción de nuevas estructuras, lo que permite a la Neuroevolución de Topologías Aumentadas explorar eficazmente el espacio de las posibles topologías de red y sus pesos correspondientes.

2.4 Navegación autónoma

La navegación autónoma se refiere a la capacidad de un robot para moverse y actuar en un entorno de manera segura y eficiente, sin intervención humana. Esto implica procesar información sensorial del entorno, planificar rutas y evitar obstáculos.

2.4.1 Técnicas y enfoques de navegación

Existen varias técnicas y enfoques para abordar el problema de la navegación autónoma, que se pueden clasificar en dos categorías principales:

- Métodos basados en mapas: Requieren un mapa previo del entorno y utilizan algoritmos de planificación de rutas para encontrar el camino óptimo desde un punto inicial hasta un objetivo.
- Métodos basados en comportamientos: No requieren un mapa del entorno y se basan en la interacción directa con el mismo mediante sensores y actuadores, utilizando reglas de comportamiento predefinidas o aprendidas.

2.4.2 Sensores y percepción del entorno

Para navegar de forma autónoma, los robots necesitan percibir su entorno y estimar su posición y orientación. Esto se logra mediante el uso de sensores, como:

- Sensores de distancia y proximidad: LIDAR, sensores ultrasónicos, sensores infrarrojos.
- Sensores de visión: Cámaras RGB, cámaras de profundidad, cámaras estéreo.
- Sensores de localización y movimiento: GPS, IMU (Unidad de Medición Inercial), odometría de ruedas.

2.5 Estado del Arte en Navegación Autónoma

El principal aporte de nuestro TFG radica en la capacidad de aprendizaje y adaptación a entornos complejos y cambiantes que brinda la combinación de AAGG y redes neuronales. Esta adaptabilidad resulta especialmente relevante en el ámbito de la navegación autónoma, donde las condiciones pueden variar considerablemente y surgir situaciones imprevistas.

Diversos enfoques se han explorado para resolver los retos de la navegación autónoma. Algunos se basan en técnicas tradicionales de control, mientras que otros recurren al uso de la Inteligencia artificial (IA) y el aprendizaje automático.

2.5.1 Enfoques Tradicionales

Las técnicas tradicionales de control, como el control proporcional-integral-derivativo (PID), el control de retroalimentación de estados y el control basado en modelos, han sido ampliamente utilizados en la navegación autónoma. En estos métodos, se emplean sensores para medir el estado del robot y se utiliza un controlador diseñado para minimizar el error entre el estado actual y el objetivo.

Aunque estos enfoques son efectivos en entornos bien definidos y predecibles, suelen tener dificultades para lidiar con la incertidumbre y la variabilidad inherentes a entornos reales y dinámicos. Además, estos métodos requieren una formulación matemática precisa del sistema, lo que puede ser desafiante en sistemas complejos.

Por ejemplo, este estudio[5] propone una metodología para la síntesis de autómatas en aplicaciones de navegación autónoma terrestre, basándose en el diseño de autómatas por gramáticas regulares. Este otro estudio[2] presenta una arquitectura de control basada en el paradigma reactivo y en la teoría de control *fuzzy* para el seguimiento de paredes. Como último ejemplo, este estudio[1] emplea algoritmos probabilísticos y genéticos para estimar la posición y planificar la ruta del robot.

Si bien estos enfoques pueden ser efectivos en situaciones específicas, podrían enfrentar desafíos en cuanto a su flexibilidad y adaptabilidad frente a entornos y condiciones cambiantes que son comunes en sistemas de navegación autónoma. Esto es debido a la gran incertidumbre y variabilidad que se puede encontrar en un entorno de la vida real. Contrariamente, la metodología utilizada en este TFG, que fusiona algoritmos genéticos y redes neuronales, aporta una mayor adaptabilidad, explorando un rango amplio de soluciones posibles y adaptándose eficientemente a cambios, mientras las redes neuronales son capaces de aprender patrones complejos y no lineales a partir de los datos.

2.5.2 Enfoques basados en IA

Los enfoques basados en IA, como el uso de algoritmos de aprendizaje automático y redes neuronales, han ganado popularidad en la navegación autónoma. Estos métodos buscan aprender un modelo de control a partir de datos de entrenamiento, que pueden incluir ejemplos de comportamiento deseado, recompensas y castigos, o simplemente interacciones con el entorno.

Entre estos enfoques, el Aprendizaje por refuerzo (RL) ha mostrado resultados prometedores en tareas de navegación. El RL permite que los robots aprendan a tomar decisiones óptimas al interactuar con el entorno y recibir retroalimentación en forma de recompensas o castigos. Por ejemplo, en este estudio[12] se presenta un sistema de navegación autónoma para robots móviles que utiliza técnicas de Aprendizaje por refuerzo profundo (DRL) para operar en entornos interiores con obstáculos dinámicos.

Sin embargo, estos enfoques también tienen sus desafíos. El aprendizaje automático y la IA generalmente requieren grandes cantidades de datos de entrenamiento y tiempo de procesamiento, lo que puede ser un obstáculo en algunas aplicaciones. Además, estos métodos pueden dar como resultado modelos que son difíciles de interpretar y validar, lo que puede ser un problema en aplicaciones críticas de seguridad.

3 Objetivos

El objetivo principal de este proyecto es analizar la posibilidad del uso de algoritmos genéticos como medio de entrenamiento de una red neuronal para conseguir que aprenda a navegar de manera completamente autónoma en un entorno peatonal, evitando todo choque y yendo de una posición inicial a una posición destino. Para alcanzar dicho objetivo, es necesario abordar una serie de objetivos intermedios que sirvan de guía de desarrollo para llegar al resultado deseado. Dichos objetivos son:

- Diseñar e implementar un algoritmo genético para optimizar los parámetros de la red neuronal de inteligencia artificial utilizada en el robot Turtlebot3.
- Experimentar con distintos enfoques y variaciones del algoritmo genético
- Evaluar y comparar su rendimiento en términos de velocidad, precisión y eficiencia en diferentes escenarios de navegación autónoma.
- Identificar fortalezas y debilidades de la arquitectura de la red neuronal y proponer mejoras en su diseño y configuración.
- Discutir la viabilidad y limitaciones de la aplicación de los algoritmos genéticos y neuroevolución en la navegación autónoma.

4 Metodología

En este apartado se describe la metodología empleada para llevar a cabo el desarrollo e implementación del sistema de conducción autónoma basado en algoritmos genéticos y redes neuronales.

Para ello, se expondrán las herramientas de software utilizadas, como ROS, Gazebo y Unity, destacando sus características y ventajas en el contexto de la conducción autónoma y la simulación de entornos realistas. Asimismo, se explicará cómo se establece la comunicación entre los diferentes componentes del sistema, incluyendo los nodos de ROS y los entornos de simulación, así como las estrategias de control para los vehículos autónomos en Unity. Además, se detallarán las características del robot utilizado, el Turtlebot3. La arquitectura del sistema se detallará en función de sus diferentes módulos y componentes, presentando cómo se integran y coordinan para llevar a cabo las tareas de aprendizaje y control en el contexto de la conducción autónoma.

Esta metodología tiene como objetivo proporcionar un enfoque estructurado y eficiente para la realización del proyecto, facilitando la comprensión, el diseño y la evaluación del sistema propuesto.

4.1 Software y herramientas utilizadas

Para el desarrollo e implementación del sistema de conducción autónoma propuesto en este trabajo, se han utilizado diversas herramientas de software que facilitan la comunicación, la simulación y el control de los robots autónomos. A continuación, se describen las principales herramientas empleadas:

4.1.1 ROS

ROS es *framework* de código abierto para el desarrollo de software en robótica. ROS proporciona bibliotecas y herramientas para ayudar a los desarrolladores de software a crear aplicaciones de robots, simplificando el proceso de comunicación entre los diferentes componentes del sistema y ofreciendo funcionalidades útiles para el control, la percepción y la navegación de robots.

4.1.2 Gazebo

Gazebo es un simulador de robótica de código abierto que permite el modelado y la simulación de entornos y robots en 3D. Este software ofrece una interfaz de usuario intuitiva, así como una amplia gama de funcionalidades que incluyen la generación de terrenos, la importación de modelos y la simulación de sensores y actuadores. Además, Gazebo se integra fácilmente con ROS, facilitando el intercambio de datos entre el simulador y el sistema de control.

4.1.3 Unity

Unity es un motor de desarrollo de videojuegos que se utiliza para crear aplicaciones en 2D y 3D, y es especialmente conocido por su flexibilidad y facilidad de uso. En este trabajo, Unity se emplea para crear entornos de simulación realistas y personalizables, permitiendo modelar escenarios complejos y variados para evaluar el rendimiento del sistema de conducción autónoma. La comunicación entre Unity y ROS se logra mediante el uso de un paquete instalable del simulador.

4.2 Comunicación entre sistemas

Para el correcto funcionamiento del proyecto, ha sido necesario implementar la conexión entre el nodo de ROS desarrollados y los dos simuladores utilizados. A continuación, se detallan los pasos seguidos para realizar la conexión en ambos simuladores:

4.2.1 Conexión de Gazebo con ROS

Como se ha mencionado con anterioridad en el apartado 4.1.2, el simulador Gazebo está preparado para la fácil integración con ROS, de manera que es capaz de conectarse a los *topics* de los modelos importados y de actualizar los mismos cuando algún mensaje es recibido. Por lo tanto, una vez implementado el nodo de ROS, la conexión con Gazebo es automática. A continuación, se muestra un diagrama de flujo de la conexión establecida entre estos dos componentes:

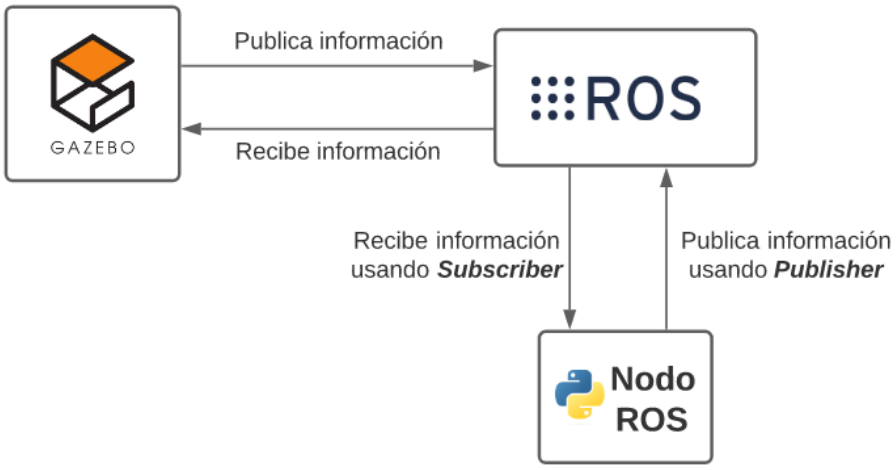


Figura 4.1: Diagrama de conexión de Gazebo con ROS y nodo

En el anterior diagrama, podemos observar como ROS hace de intermediario entre Gazebo y el nodo programado, que realiza el intercambio de conexión utilizando elementos **Publisher**, para publicar información a un *topic*, y **Subscriber**, para recibir información de un *topic*.

4.2.2 Conexión de Unity con ROS

Por otro lado, la conexión de ROS con Unity es más compleja, debido a que no está diseñado para funcionar de forma nativa con este sistema. Para lograrlo, se han utilizado dos paquetes externos, creados por el equipo de Unity Robotics, uno llamado *ROS TCP Endpoint*[15], el cual es un paquete de ROS y permite ejecutar un nodo que sirve como *endpoint* para exponer todos los *topics* que existan y para permitir crear nuevos, y otro llamado *ROS TCP Connector*[14], que consiste en un paquete que permite que Unity pueda conectarse al *endpoint* mencionado y acceder a toda la información del sistema de ROS. A continuación, se muestra un diagrama de flujo de la conexión establecida entre estos tres componentes:

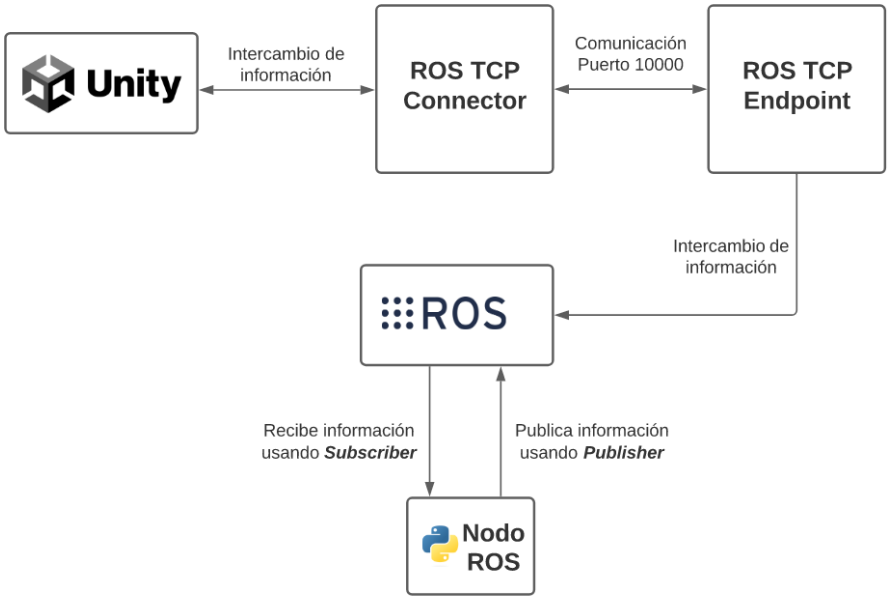


Figura 4.2: Diagrama de conexión de Unity con ROS y nodo

En el anterior diagrama, podemos observar como Unity se conecta con el *ROS TCP Connector* a través del puerto 10000, al que el *ROS TCP Endpoint* está exponiendo la información de ROS, que hace de intermediario con el nodo programado que, a su vez, realiza el intercambio de conexión utilizando elementos **Publisher**, para publicar información a un *topic*, y **Subscriber**, para recibir información de un *topic*.

4.3 Turtlebot3

El modelo de robot elegido para este proyecto es el *Turtlebot3* en su versión *waffle*[7], que es un robot móvil compacto y altamente personalizable, con capacidades avanzadas que lo hacen especialmente atractivo para este trabajo. En cuanto a la locomoción, este robot es capaz de moverse con precisión y eficacia en una variedad de entornos, gracias a su diseño dinámico y sus componentes de alta calidad. Este aspecto es esencial para las tareas de navegación autónoma, donde la habilidad para moverse de manera fluida y confiable es un

requisito fundamental.

El diseño de la locomoción del Turtlebot3 se centra principalmente en su sistema de tracción diferencial, una característica que dota al robot de una extraordinaria maniobrabilidad, incluyendo la habilidad para girar sobre sí mismo. Su chasis ligero y resistente, junto con los motores de sus ruedas, le otorgan una velocidad lineal máxima de hasta 0.26 m/s y una velocidad angular máxima de hasta 1.82 rad/s, y una velocidad mínima que puede ser tan baja como sea necesario para una operación precisa y segura.

El Turtlebot3 viene equipado con un Light Detection and Ranging (LIDAR), una tecnología clave para la percepción del entorno. Este sensor utiliza un haz de luz láser para detectar y medir la distancia a los objetos en el entorno del robot, generando así un mapa detallado de su entorno inmediato. El LIDAR es especialmente útil para la navegación autónoma, ya que proporciona datos precisos y confiables en tiempo real, permitiendo al robot evitar obstáculos y navegar de manera segura.

El modelo de LIDAR que incorpora el Turtlebot3 es capaz de proporcionar un rango de detección de hasta 3.5 metros, con una resolución angular de 0.36 grados y una frecuencia de actualización de 5 Hz. Esto le permite obtener una visión detallada y precisa de su entorno, esencial para las tareas de navegación y planificación del camino.

Figura 4.3: Turtlebot 3 Waffle

5 Desarrollo

En este apartado, se expone en detalle el desarrollo de los elementos clave que conforman el núcleo de este Trabajo de Fin de Grado.

En la primera sección, se expone el proceso de generación de entornos utilizando dos herramientas distintas de simulación, Gazebo y Unity. Se detallan los procedimientos para la importación del modelo del robot y la creación de elementos específicos del entorno, como los puntos de control (checkpoints), las paredes, el circuito y la ciudad.

A continuación, en la segunda sección, se presenta el diseño y la implementación del controlador de los robots. Se discuten dos enfoques: un controlador genérico y un controlador distribuido.

En la tercera, se detallan los diferentes tipos de controladores que hacen posible la comunicación con ROS, así como el correcto funcionamiento del entorno (sistema de físicas y colisiones).

Por otro lado, en la cuarta sección se aborda el núcleo de este trabajo: la aplicación de algoritmos genéticos a las redes neuronales. Se detalla la implementación clásica del algoritmo y se presenta una variante avanzada: la *Neuroevolución de Topologías Aumentadas* (NEAT).

En la quinta sección, se describe el flujo completo de control del proyecto, que incluye los scripts de control de Unity y el nodo de ROS, así como su comunicación.

Finalmente, en la última sección se explican los mensajes de ROS que se han implementado para el desarrollo de este proyecto.

5.1 Generación de entonos

El primer paso para el correcto desarrollo del proyecto consiste en la generación de los entornos donde se van a producir las simulaciones. Para ello, como ya se ha nombrado anteriormente, se han utilizado dos simuladores distintos.

5.1.1 Gazebo

En el periodo inicial del desarrollo del proyecto el simulador utilizado fue Gazebo. Gracias a la fácil comunicación con ROS, este software permitió que el grueso de la programación se centrase en el nodo de ROS con la implementación del algoritmo genético. Además, este simulador también está preparado para importar los modelos de los robots a través de un fichero de extensión Unified Robot Description Format (URDF), y el sistema de físicas y colisiones es automático utilizando ese formato.

No obstante, se encontró un contratiempo debido al funcionamiento del sistema de colisiones. Esto se debe a que, para poder entrenar de manera eficiente usando AG, es primordial poder ejecutar en paralelo varios robots que actúen como individuos de cada generación. Sin embargo, en Gazebo no es posible desactivar las colisiones entre los mismos robots, por lo que es imposible entrenar en el mismo entorno con más de un robot, sin que estos produjesen interferencias indeseadas en el entrenamiento.

Por ello, se desarrolló un script capaz de comunicarse con Gazebo a través de ROS y de generar una cantidad deseada de entornos junto a un robot de manera completamente independiente, de forma que no interfieran entre sí. Las siguientes imágenes muestran un ejemplo individual de un entorno y de generación de cincuenta entornos iguales, cada uno con un Turtlebot3:

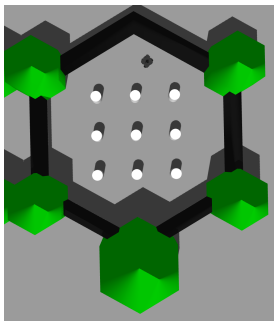


Figura 5.1: Entorno indiviual en Gazebo

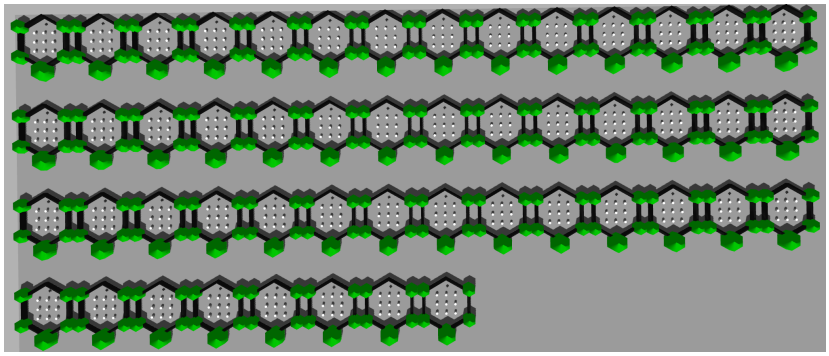


Figura 5.2: Entornos en Gazebo

A pesar de que esta preparación ya permite entrenar, se encontró un problema añadido referente a las colisiones: a la hora de crear puntos de control (checkpoints) no era posible crear un objeto que, al pasar por encima de él, desencadenase un evento. Esto es consecuencia de que no es posible detectar una colisión si estas no están activadas, pero al activarlas Gazebo automáticamente trata esos puntos de control como paredes, por lo que el robot se choca y se detiene en ellos.

Por todos estos contratiempos se tomó la decisión de cambiar a un software de simulación más flexible como Unity.

5.1.2 Unity

A la hora de realizar el cambio a Unity, el enfoque del desarrollo cambió de implementar el algoritmo genético a implementar e incluir todo lo necesario para poder utilizar Unity de la misma forma que Gazebo, añadiéndole además los puntos de control. Para ello, se ha tenido que buscar una forma de importar modelos URDF en el simulador para poder usar el propio Turtlebot3, se han tenido que crear objetos para crear los checkpoints y las paredes que servirán como delimitador del entorno.

URDF Importer

Para poder importar el modelo del Turtlebot 3 a partir del formato mencionado anteriormente, se ha hecho uso de una herramienta llamada *URDF Importer* que, como su nombre indica, permite importar ficheros URDF y convertirlos en objetos adaptándolos al propio simulador.

Elementos del entorno

En este proyecto, se han creado dos tipos de objetos para el entorno: los puntos de control, conocidos como *checkpoints*, y las paredes. Estos objetos son elementos sencillos sin texturas, a los cuales se les ha asignado controladores específicos, cuyo funcionamiento se detallará en las secciones posteriores. A continuación, se muestran ambos elementos:

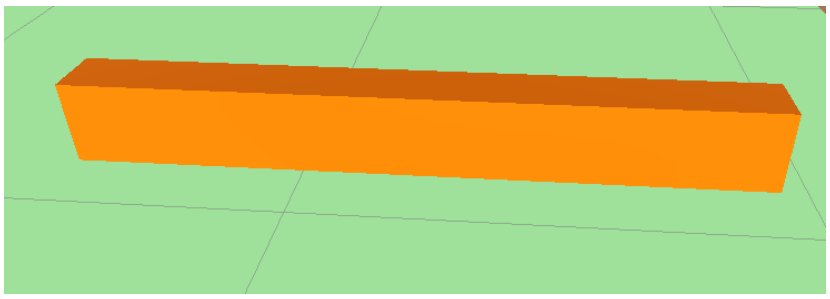


Figura 5.3: Checkpoint

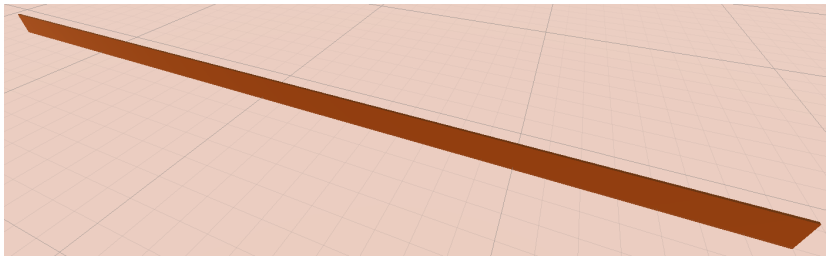


Figura 5.4: Pared

Circuito

El primer entorno de experimentación creado ha sido un circuito a partir de los objetos comentados en el apartado anterior. A continuación se muestra una imagen del mismo:

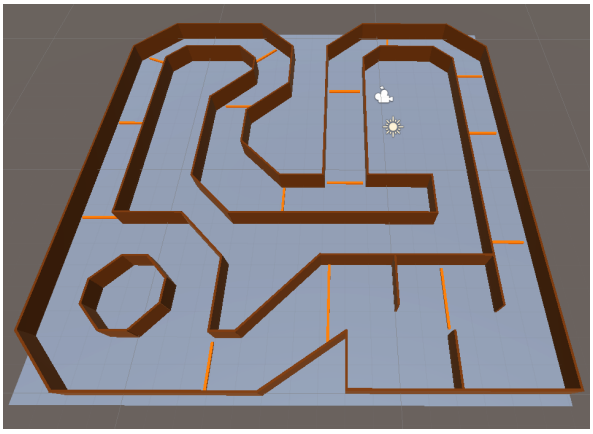


Figura 5.5: Circuito

Como se puede ver en la imagen, a lo largo de todo el circuito se han colocado una serie de checkpoints que sirven de guía para el correcto entrenamiento de los robots. Este circuito ha sido desarrollado específicamente para ser desafiante, contando con una serie de zonas engañosas o que pueden presentar un reto a los robots.

La primera de ellas consiste en un zigzag, que obliga a los robots a hacer giros a ambos sentidos. A continuación, se encuentra una rotonda, de forma que los robots podrían incluso quedar atascados dando vueltas a la misma sin avanzar. Por último, encontramos un callejón sin salida. Este callejón es, sin duda, el reto más desafiante, ya que los robots, al no poder ver más allá de su límite de visión, pueden tomar este callejón como camino válido, adentrarse en él y chocarse al final del mismo. El resto de partes, como las curvas y las rectas no suponen un gran desafío.

Ciudad

El segundo entorno de experimentación creado ha sido una zona peatonal de una ciudad. Los objetos que se han utilizado en este entorno forman parte de un conjunto de *Assets* de la *Assets Store* de Unity, llamado *Low poly City Assets*[11].

La zona peatonal cuenta con diversos edificios, señales de tráfico, vegetación y distintas piezas de mobiliario, como sillas y mesas, así como otros objetos que se pueden encontrar en las zonas peatonales.

A continuación se muestra una imagen de la misma:

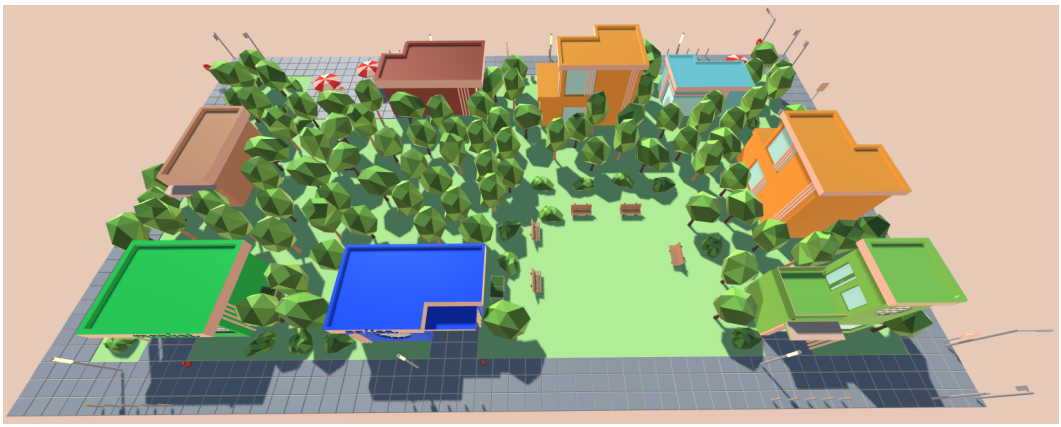


Figura 5.6: Ciudad

Este entorno supone un desafío mayor que el circuito. Esto se debe a la carencia de paredes que puedan servir a los robots como 'guías'. Además, en este entorno no existe un camino predeterminado que los robots deban seguir siempre. A esto se le añade la dificultad adicional de esquivar obstáculos, que obliga al robot a adoptar estrategias diferentes de las que pudieran adoptar en el circuito.

5.2 Controlador de robots

Una de las piezas clave para el correcto funcionamiento de los experimentos es la gestión del control de los robots dentro de Unity, de forma que sea posible dirigir a todos los individuos durante el entrenamiento de manera eficiente. Para ello, se han llevado a cabo dos implementaciones, siendo la segunda una mejora con respecto a la primera en lo que a simplicidad se refiere.

Junto con este controlador, se han desarrollado dos controladores adicionales. Uno de ellos se encarga de enviar en cada actualización del simulador los datos de todos los láseres de los robots al topic 'scan' correspondiente. El otro se encarga de recibir los valores de velocidad del topic 'cmd_vel' correspondiente y de aplicar ese movimiento a cada robot.

5.2.1 Controlador centralizado

En primer lugar, se ha desarrollado un controlador centralizado para un conjunto de robots. Este controlador se encarga de iniciar y posicionar a cada robot en su punto de inicio, además de registrar datos relevantes del entorno para cada unidad. Dichos datos incluyen los *checkpoints* por los que ha pasado cada robot, sus posiciones finales, los robots implicados en colisiones y las recompensas individuales obtenidas. No obstante, a pesar de sus fortalezas, este sistema de control presenta ciertos inconvenientes. Uno de los principales es la dependencia de un único controlador para toda el conjunto, lo que puede dar lugar a cuellos de botella y retrasos en la comunicación. Además, cualquier falla en el controlador centralizado puede afectar a todo el sistema.

En este esquema, el controlador establece una conexión con ROS para enviar y recibir mensajes relevantes. Sin embargo, el proceso puede volverse complejo y propenso a errores, especialmente con un gran número de robots y una gran cantidad de datos que manejar.

Cada actualización de la simulación conlleva la ejecución de una serie de pasos, que se describen a continuación:

- En primer lugar, se envían los mensajes de colisión a ROS y se eliminan los robots que hayan colisionado.
- A continuación, si se ha recibido un mensaje de reinicio, se realizan las siguientes acciones:
  - Se actualizan las recompensas.
  - Se envía el mensaje que contiene las recompensas actualizadas.
  - Finalmente, todos los Turtlebot3 son reiniciados.

Este proceso asegura un ciclo constante de actualización, evaluación y adaptación, que es fundamental para el funcionamiento eficiente del sistema de robots.

5.2.2 Controlador distribuido

Para superar estas limitaciones, se implementó un controlador distribuido. En este enfoque, cada robot es autónomo y gestiona su propia información, incluyendo su identificación, nombre, recompensa, estado de colisión, entre otros. Esto reduce la carga de comunicación y procesamiento del controlador, permitiendo un funcionamiento más eficiente y robusto del sistema.

Además, cada robot tiene la tarea de enviar los mensajes de colisión. En cada actualización de la simulación, el robot actualiza su posición y la envía a Unity a través del *topic position*. Su *fitness* también se actualiza en función de la recompensa que obtiene de acuerdo con la función de recompensas específica del entorno en el que se encuentra.

Por otro lado, se ha diseñado un controlador para transmitir la información necesaria para el correcto desarrollo de la simulación. Este controlador es responsable de iniciar cada uno de los robots y de reiniciarlos cuando se recibe un mensaje de *reset* en el *topic* que lleva su mismo nombre. Este proceso de reinicio incluye el envío de las recompensas de todos los robots al nodo de ROS, lo que garantiza la correcta actualización y seguimiento del rendimiento de cada robot.

Sin embargo, este modelo también tiene sus desafíos. La sincronización y la coherencia de los datos pueden ser más difíciles de lograr, y los robots deben ser más complejos y capaces de manejar tareas de control más avanzadas. A pesar de la complejidad, este controlador ofrece un mayor rendimiento y escalabilidad, permitiendo una mayor flexibilidad y autonomía para cada robot.

5.3 Controladores del entorno

Además de los controladores individuales de los robots, se han desarrollado una serie de controladores dedicados a gestionar la interacción de los robots con su entorno.

El primer controlador de este conjunto es el controlador de colisiones con los *checkpoints*. Este se encarga de actualizar la información referente a los puntos de control que cada robot ha alcanzado durante la simulación.

En segundo lugar, se encuentra el controlador de colisiones con las paredes. La principal función de este controlador es actualizar el estado de los robots que han chocado con algún objeto que se considere una pared. En caso de colisión, este controlador se encarga de eliminar al robot de la simulación hasta el siguiente reinicio.

5.4 Algoritmo genético aplicado a redes neuronales

A continuación, se explicará el desarrollo del tema principal de investigación del proyecto, el algoritmo genético. Cuando se habla de este algoritmo aplicado a una red neuronal, es necesario describir una serie de cambios en el algoritmo original.

En primer lugar, los individuos pasan a ser redes neuronales que, en lugar de genes, tienen pesos y umbrales. Esto implica una variación en las operaciones de *crossover* y mutación. Con respecto al cruce, el nuevo individuo, o hijo, que se generará a partir de dos progenitores, será una red neuronal conformada por los pesos y umbrales de ambos padres. Dicha operación puede llevarse a cabo mediante dos estrategias distintas:

- Cruce a nivel de conexiones: En este enfoque, la herencia ocurre a nivel de las conexiones. Es decir, para cada una de las conexiones de los padres, existe una probabilidad de que el hijo herede los pesos y umbrales de uno u otro progenitor.

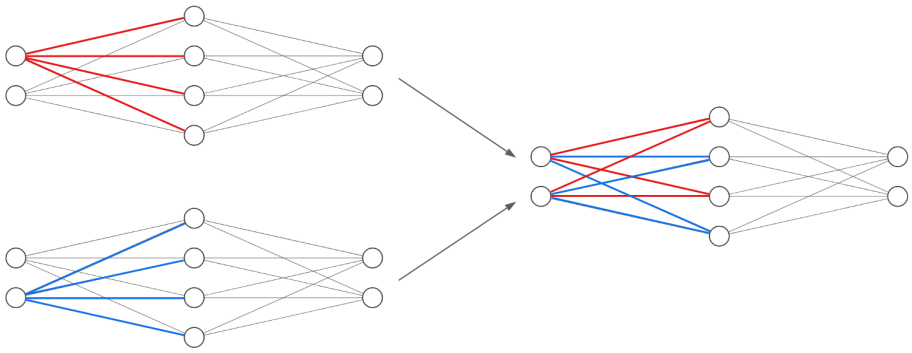


Figura 5.7: Cruce a nivel de conexiones

- Cruce a nivel de neuronas: Esta estrategia plantea que la herencia sucede a nivel de neuronas, es decir, cada neurona tiene una probabilidad de heredar todos los pesos y umbrales de esa neurona de uno de los padres. Este método generalmente resulta más efectivo, ya que las neuronas suelen trabajar en conjunto con sus atributos (peso y umbral), y al heredar neuronas completas, el comportamiento de dicha neurona

se mantiene intacto, preservando su funcionalidad en la arquitectura. Asimismo, este enfoque es el seleccionado para este proyecto.

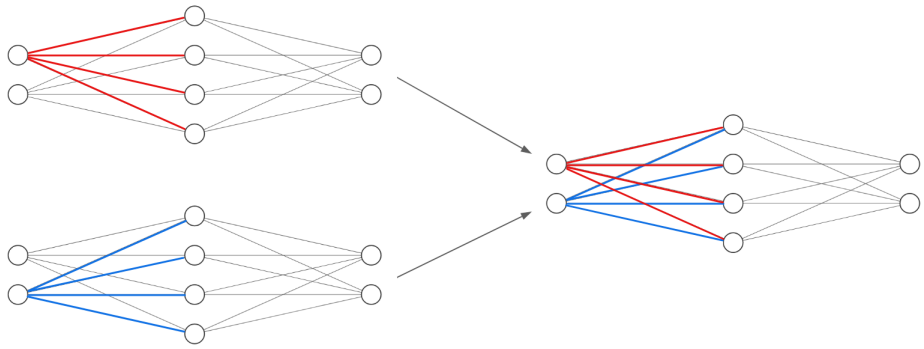


Figura 5.8: Cruce a nivel de neuronas

Por otro lado, el proceso de mutación se lleva a cabo en cada una de las conexiones, de manera que hay una probabilidad de que cada peso y umbral mute o, lo que es lo mismo, que aumente o disminuya su valor en una cantidad establecida (índice de mutación).

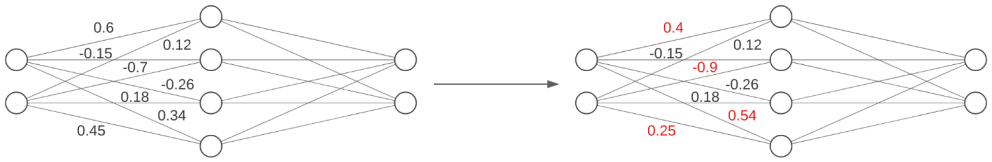


Figura 5.9: Mutación en redes neuronales

5.4.1 Algoritmo Genético Clásico

Con las diferencias ya explicadas, podemos avanzar hacia el detalle del desarrollo del nodo de ROS. Este nodo cumple con diversas funciones, entre las cuales se destacan la ejecución del algoritmo genético, la gestión del flujo de información entre él y elsimulador para asegurar un correcto funcionamiento, y el almacenamiento de información de interés, como métricas y los mejores individuos de cada generación.

El nodo de ROS está estructurado en un *script* principal que actúa como ejecutable, así como en varias clases complementarias. Entre estas clases se incluyen 'Turtlebot3', 'Generation' y 'Model'. La clase 'Turtlebot3' incorpora toda la información y los métodos necesarios para cada robot. Por su parte, la clase 'Generation' se encarga de conservar los datos de cada generación y de administrar el proceso de evolución. Finalmente, la clase 'Model' engloba la información y los métodos relacionados con el modelo de la red neuronal.

5.4.2 Neuroevolución de Topologías Aumentadas (NEAT)

Para su implementación, se ha hecho uso del paquete de Python *neat-python*[6], sin que haya sido necesario realizar modificaciones significativas. En este contexto, la clase 'Turtlebot3' se limita a contener la información del robot, incluyendo el genoma, que es el modelo que produce NEAT. Cabe destacar que, en esta circunstancia, tanto la clase 'Model' como la clase 'Generation' se vuelven prescindibles, dado que sus funcionalidades ya están integradas en el paquete de NEAT.

5.5 Flujo de control

Una vez explicados todos los componentes por separado, se va a explicar detalladamente y con la ayuda del siguiente diagrama de flujo la interacción que existe entre ellos.

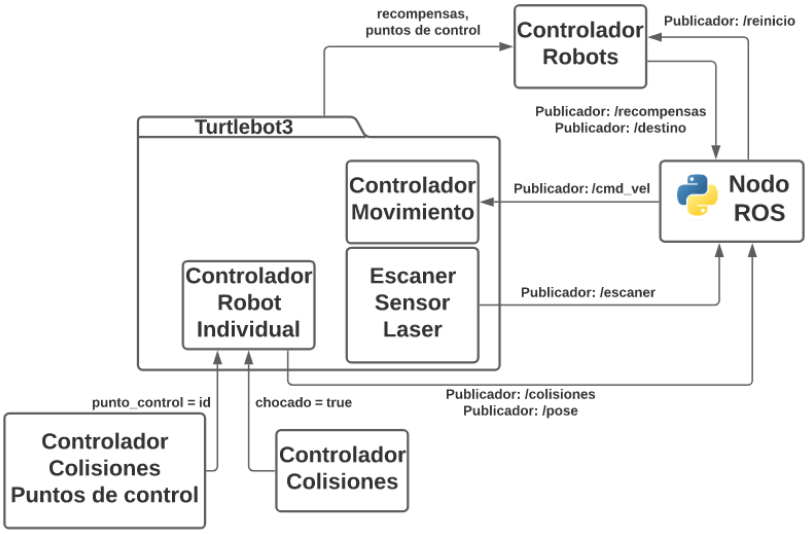


Figura 5.10: Diagrama de flujo de interacción entre controladores

En el simulador 3D, el objeto *Turtlebot3* integra tres *scripts*: el *Controlador de Movimiento*, el *Controlador de Robot Individual* y el *Escáner del Sensor Láser*. El *Controlador de Movimiento*, por un lado, tiene la responsabilidad de recibir la velocidad del robot, que es suministrada por el nodo de ROS a través del *topic* /cmd_vel correspondiente.

Además, el *Controlador de Robot Individual*, por su parte, recibe las ids de los puntos de control que el robot atraviesa, información que proviene del *Controlador de Colisiones de Puntos de Control*. Asimismo, este controlador maneja las señales de colisión enviadas por el *Controlador de Colisiones*. También tiene a su cargo la emisión de la id del robot, en caso de colisión, al topic /colisiones, así como la transmisión de la pose del robot al topic /pose para su procesamiento por el nodo de ROS.

Por último, el *Escáner del Sensor Láser* gestiona el envío de los datos del LIDAR al nodo de ROS mediante el topic /escaner del robot.

Por su parte, el nodo de ROS se comunica con el *Controlador de Robots* a través del topic /reinicio para enviarle el mensaje de reinicio de la simulación. Este controlador también se ocupa de enviar al nodo de ROS las recompensas de los robots (previamente obtenidas del Turtlebot3) a través del topic /recompensas. Finalmente, también tiene la responsabilidad de comunicar la posición del destino utilizando el topic /destino.

5.6 Mensajes de ROS

Con el objetivo de facilitar la comunicación entre ROS y Unity, se han creado una serie de mensajes de ROS personalizados. Para realizar esto, se ha seguido el tutorial de creación de mensajes personalizados de la documentación oficial de ROS[10]

El primero de estos mensajes, denominado *Rewards*, contiene las recompensas de cada grupo de robots. En el caso del segundo mensaje, titulado *Pose*, se almacenan los datos de posición y rotación de cada robot. El tercer mensaje, conocido como *Destiny*, guarda la información de la posición del punto de destino. El cuarto mensaje, *Reset*, lleva consigo la señal de reinicio para una nueva ejecución de la simulación. Por último, el quinto mensaje, *Collision*, contiene la id de un robot que ha experimentado una colisión. A continuación se presenta un diagrama para visualizar de forma más clara la información de estos mensajes:

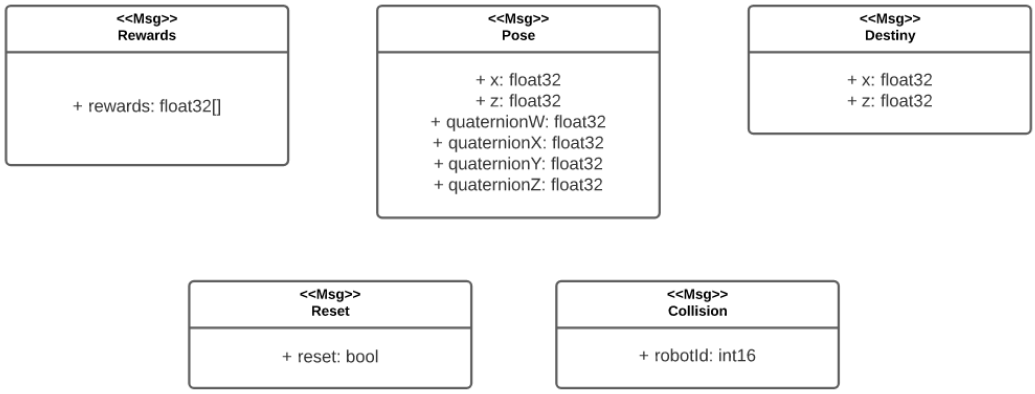


Figura 5.11: Diagrama de los mensajes

6 Resultados

Este apartado está dedicado a la presentación y análisis de los resultados obtenidos a lo largo de los diversos experimentos llevados a cabo en el marco de este proyecto. Para evaluar de manera objetiva y sistemática el desempeño de los modelos y estrategias implementadas, se han establecido varias métricas críticas que nos permitirán cuantificar sus logros y limitaciones.

Primero, se utilizará la medida de *fitness* máxima y media, proporcionando una indicación de la calidad de las mejores soluciones encontradas y la calidad promedio de todas las soluciones en la población en cada generación, respectivamente. Ambas métricas son fundamentales para entender la eficacia de la evolución y la diversidad de la población.

En segundo lugar, consideraremos el tiempo transcurrido hasta llegar al destino y la distancia recorrida para alcanzarlo. Estas medidas nos proporcionarán una evaluación en términos de eficiencia temporal y espacial de las soluciones obtenidas, indicadores claves en muchos problemas de optimización.

Finalmente, en el caso de NEAT, estaremos analizando el número de individuos por especie, una métrica que nos ayudará a entender la diversidad genética dentro de la población y a evaluar la capacidad de los algoritmos para explorar el espacio de soluciones.

A través de estos parámetros, se espera proporcionar una visión exhaustiva y profunda de los resultados obtenidos, identificando tanto los logros como las áreas de mejora potencial. A continuación, se presenta un desglose detallado de los resultados obtenidos para cada uno de los experimentos realizados.

En este proyecto, las pruebas se desarrollaron en dos escenarios que se describen en la sección 5.1.2. Cada uno de estos entornos plantea un objetivo distinto para la inteligencia artificial: en el primer caso, la meta es completar una vuelta entera al circuito, es decir, avanzar desde el inicio hasta la meta; mientras que, en el segundo, el desafío consiste en navegar desde un punto de partida hasta un destino específico.

Para alcanzar estos objetivos, se ejecutaron varios experimentos utilizando distintas arquitecturas de redes neuronales, cuyos detalles se describirán en las secciones siguientes.

6.1 Circuito

En este primer entorno, las redes neuronales obtienen sus entradas a partir de las mediciones del sensor LIDAR, lo que permite a los individuos determinar la ubicación de las paredes del circuito.

En todos los experimentos realizados en este entorno, la función de fitness, que valora la eficacia de los individuos, es la misma: la distancia total que el robot ha recorrido, complementada con una puntuación adicional por cada checkpoint superado. Aquellos robots que logran alcanzar la meta reciben una penalización en función del tiempo, favoreciendo a aquellos que llegan más rápido. La función se expresa de la siguiente manera:

$$f = \begin{cases} D_r + n^{\circ}checkpoints * R_1 & \text{si robot no ha llegado a la meta} \\ D_r + n^{\circ}dcheckpoints * R_1 - T & \text{si robot ha llegado a la meta} \end{cases} \tag{6.1}$$

Donde R_1 es la constante de recompensa por superar un checkpoint, D_r denota la distancia recorrida y T es el tiempo de ejecución del robot.

Es relevante destacar que, en todas las pruebas realizadas en el circuito, la capa de salida de las arquitecturas consta de dos salidas, definiendo la velocidad lineal y angular del robot, respectivamente. La capa de salida utiliza la función de activación tanh, que genera un rango de valores posibles entre [-1, 1]. Sin embargo, para adaptarse a las necesidades específicas del robot, estos valores son normalizados. En concreto, para la velocidad lineal, los valores se normalizan dentro del rango [-0.26, 0.26] y para la velocidad angular, se normalizan en el rango [-1.82, 1.82].

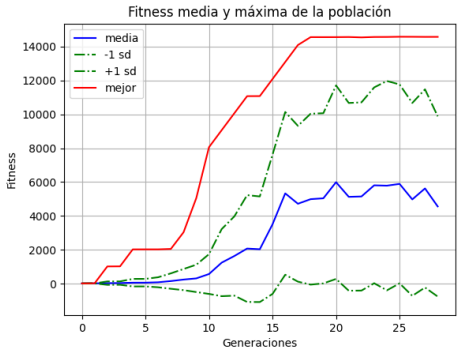
Las métricas que se evaluarán en este estudio incluyen tanto la función de fitness como el rendimiento de los individuos en cada generación con respecto a la distancia recorrida y el tiempo mencionados anteriormente. A pesar de ello, se dará más énfasis al tiempo, debido a que este parámetro ofrece un mayor margen de optimización en el contexto de un circuito en comparación con la distancia recorrida.

Cada uno de estos experimentos se realizó con poblaciones de 100 individuos y, para garantizar una comparación realista, se entrenaron durante el mismo número de generaciones, en concreto, 28, en todos los casos.

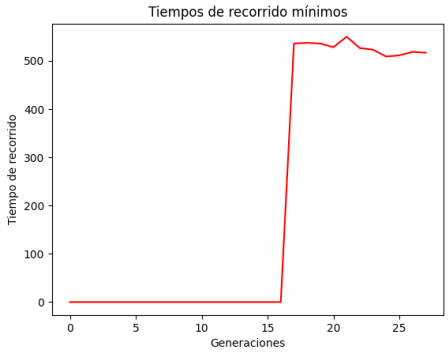
Algoritmo genético clásico con arquitectura simple

En este primer experimento, se ha empleado la implementación de un algoritmo genético clásico en conjunción con una arquitectura simple. La mencionada arquitectura consiste en una capa de 5 entradas, que corresponden a los ángulos del robot: 0º, 45º, 90º, 135º y 180º, seguido por dos capas ocultas densas de seis y cuatro neuronas, respectivamente, y finalmente la capa de salida.

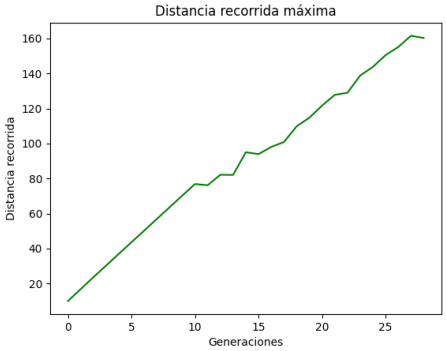
Después de realizar 28 generaciones, se obtuvieron los siguientes resultados:



(a) Fitness



(b) Tiempo de vuelta



(c) Distancia recorrida

El análisis de la gráfica de la función de fitness sugiere que los robots han logrado completar el circuito en la 17ª generación. Adicionalmente, la tendencia creciente del promedio de las puntuaciones fitness implica que cada generación está incrementando la cantidad de individuos con un mejor desempeño.

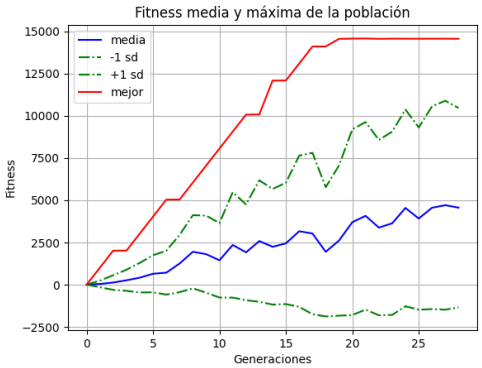
En concordancia con esto, la gráfica de tiempo de recorrido muestra que a partir de dicha generación el tiempo empieza a ser registrado y, conforme avanza cada generación, el tiempo general tiende a reducirse. No obstante, se observa un incremento en la 21ª generación, que no necesariamente significa un empeoramiento de la población, sino que puede ser atribuido a la variabilidad de las acciones de un mismo individuo, causado por el ruido en la información proporcionada por el sensor LIDAR.

Por último, como se esperaba, la distancia recorrida por los robots va incrementándose a medida que pasan las generaciones, ya que estos deben avanzar a través del circuito.

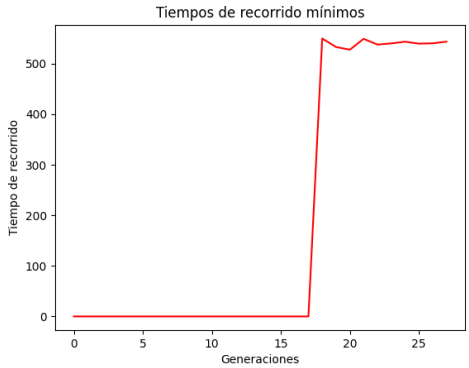
NEAT con arquitectura simple

En este segundo experimento, se ha empleado NEAT en una arquitectura simple. Esta arquitectura, similar a la del experimento anterior, cuenta con una capa de 5 entradas correspondientes a los ángulos del robot: 0º, 45º, 90º, 135º y 180º, y la capa de salida. Aquí, la arquitectura inicia con la capa de entrada conectada directamente a la de salida, puesto que NEAT se encargará de agregar y eliminar neuronas y conexiones a lo largo de las generaciones.

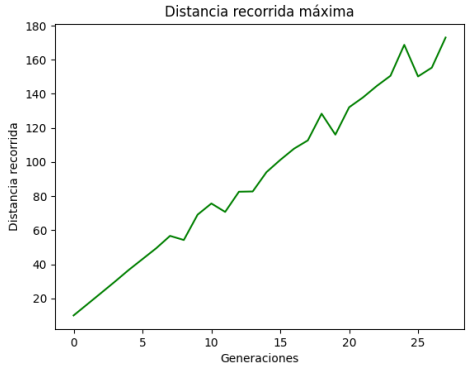
Luego de 28 generaciones, se obtuvieron los siguientes resultados:



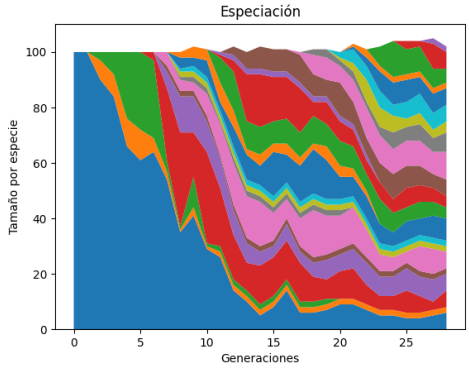
(a) Fitness



(b) Tiempo de recorrido



(c) Distancia recorrida



(d) Especiación

En la gráfica de la función de fitness se puede apreciar que, al igual que en el experimento anterior, los robots lograron completar el circuito en la 17ª generación. No obstante, es relevante destacar que en este caso el valor de fitness ha incrementado con mayor rapidez durante las primeras generaciones, probablemente debido a la diversidad de individuos que NEAT crea. Esto favorece la búsqueda de mejores soluciones en un menor lapso al explorar un rango más amplio de estrategias. De manera similar al experimento anterior, el promedio de las puntuaciones fitness tiende a incrementarse, lo que sugiere que cada generación va aumentando la cantidad de individuos con un mayor desempeño.

Es importante resaltar que el valor máximo de la función de fitness se estabiliza a partir de la generación 18. Este resultado indica que, una vez que los robots completan la vuelta al circuito, optimizar la solución se vuelve una tarea cada vez más desafiante.

De la misma manera, en la gráfica del tiempo de recorrido, es notable cómo el conteo del tiempo se inicia a partir de la generación 18, dando lugar a una tendencia descendente conforme avanzan las generaciones. Este patrón sugiere una optimización progresiva del tiempo necesario para completar el circuito.

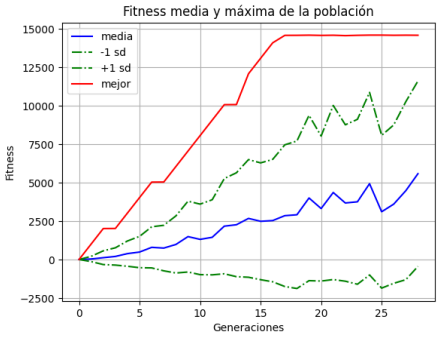
Por otro lado, centrándonos en la gráfica de la distancia recorrida. Al examinarla, se observa que la distancia que logran recorrer los robots aumenta con cada generación que transcurre. Sin embargo, también es posible apreciar ciertos empeoramientos en algunas generaciones. Tal como se mencionó anteriormente, estos contratiempos se vinculan al ruido inherente en el uso del LIDAR.

Finalmente, al analizar la gráfica de especiación, se hace evidente la formación de diversas especies a lo largo del proceso. También es posible observar cómo la población de cada especie experimenta incrementos o reducciones en función de su desempeño en la simulación. Esto sugiere que la supervivencia y evolución de las especies está directamente relacionada con su capacidad de adaptación y rendimiento en el contexto de las pruebas de simulación.

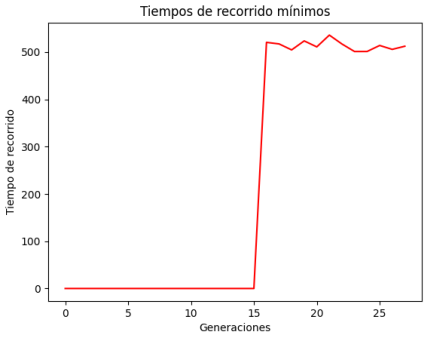
Algoritmo genético clásico con arquitectura compleja

Este experimento se ha llevado a cabo utilizando el algoritmo genético clásico en combinación con una arquitectura compleja. Esta arquitectura está compuesta por una capa de 180 entradas correspondientes a los ángulos del robot desde el 0º hasta el 180º, proporcionándole una visión completa de su parte frontal. Asimismo, incluye dos capas densas ocultas, una con 90 y otra con 45 neuronas, respectivamente, y finalmente, la capa de salida.

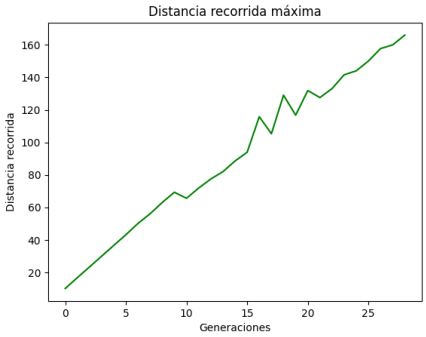
Los resultados obtenidos después del entrenamiento han sido los siguientes:



(a) Fitness



(b) Tiempo de vuelta



(c) Distancia recorrida

Al analizar los resultados tras el entrenamiento, se puede observar en la gráfica de la función de fitness que los robots lograron completar la vuelta al circuito en la generación 16. Notablemente, el valor de fitness se incrementó rápidamente en las primeras generaciones en comparación con el experimento que utilizó una arquitectura simple. Esta diferencia sugiere que la arquitectura compleja posee una mayor capacidad de aprendizaje. Además, se puede observar que el valor máximo de fitness se estabiliza a partir de la generación 17, y la media de las puntuaciones de fitness también muestra una tendencia ascendente.

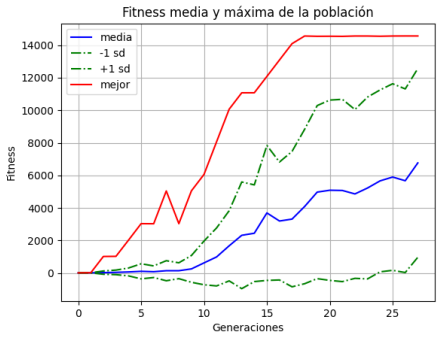
En la gráfica del tiempo de recorrido, se aprecia que el tiempo comienza a ser contabilizado a partir de la generación 16, y se puede ver cómo este disminuye a medida que avanzan las generaciones.

Por otra parte, la gráfica de la distancia recorrida evidencia un aumento en la distancia con cada generación. Al igual que en los experimentos anteriores, cualquier deterioro visible está vinculado al ruido del LIDAR.

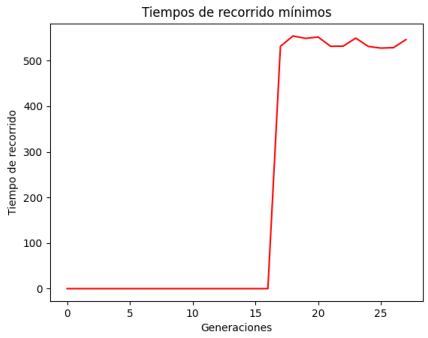
NEAT con arquitectura compleja

El cuarto experimento implementa el algoritmo NEAT con la arquitectura compleja antes mencionada. En este caso, la arquitectura ha comenzado con la capa de entrada conectada directamente con la de salida.

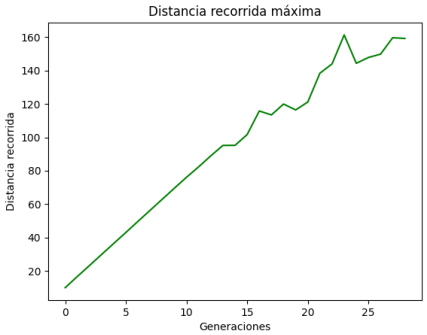
Los resultados obtenidos después de las 28 generaciones han sido los siguientes:



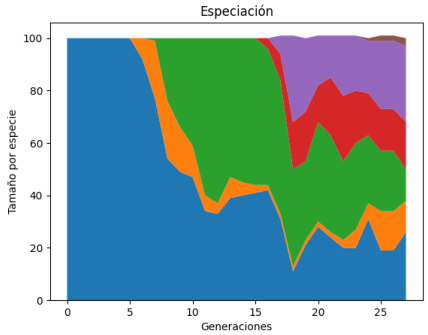
(a) Fitness



(b) Tiempo de recorrido



(c) Distancia recorrida



(d) Especiación

Los resultados muestran que los robots completaron la vuelta al circuito en la generación 17. El valor de fitness ha aumentado rápidamente, similarmente al experimento previo. La media de las puntuaciones de fitness también muestra una tendencia al alza. Sin embargo, se observa una disminución en el valor máximo en la generación 8, que se cree se debe al ruido del LIDAR, un factor ya mencionado en los experimentos anteriores. Asimismo, en este caso, el valor máximo de fitness se estabiliza a partir de la generación 18.

Similarmente a lo observado en el tercer experimento, la gráfica del tiempo de recorrido en este experimento muestra que el tiempo empieza a contarse a partir de la generación 17, y disminuye conforme avanzan las generaciones. La ausencia de una mejora significativa puede significar que el tiempo es difícil de optimizar, pudiendo necesitar un número elevado de generaciones para poder encontrar la mejor trazada del circuito.

El análisis de la gráfica de la distancia recorrida indica que la distancia aumenta con cada generación, y los deterioros visibles, como en la generación 24, se deben al ruido del LIDAR.

Reflexión

A continuación, se muestra una tabla con la recopilación de resultados obtenidos en estos 4 experimentos:

Experimento	Convergencia	Fit. máx.	Fit. media	Tiempo	Distancia	Especies
Experimento 1	17	14585.18	5991.20	508.86	161.59	-
Experimento 2	17	14569.21	4709.66	527.65	**173.02**	21
Experimento 3	**16**	**14591.50**	5585.32	**501.30**	166.00	-
Experimento 4	17	14569.63	**6766.43**	527.63	161.37	6

Cuadro 6.1: Tabla comparativa de los experimentos del circuito

Para finalizar, de acuerdo con la tabla recopilatoria de resultados, se puede concluir que la configuración experimental que ha mostrado un desarrollo más prometedor ha sido la del tercer experimento, ya que fue la primera en converger, y obtuvo el valor más alto de fitness y el mejor tiempo de vuelta. Además, la población con la mayor fitness media corresponde al cuarto experimento. A la luz de estos resultados, es seguro afirmar que los experimentos con la arquitectura compleja demostraron un rendimiento superior a aquellos con arquitectura simple, posiblemente debido a su mayor capacidad de aprendizaje y su amplio campo de visión que les permite desarrollar estrategias más precisas y óptimas.

6.2 Ciudad con objetos estáticos

En el segundo entorno, las entradas para las redes neuronales se originan de la posición del robot y la ubicación del punto de destino. Este sistema proporciona información continua sobre la ubicación actual del robot y el punto hacia el que debe dirigirse. Además, las mediciones del sensor LIDAR les permiten determinar la localización de los obstáculos en el entorno.

Todos los experimentos efectuados en este entorno emplean la misma función de fitness para evaluar la eficiencia de los individuos. Dicha función es la inversa de la distancia euclídea entre la posición actual del robot y el destino, siempre y cuando los robots no hayan alcanzado su objetivo. Este criterio fomenta la proximidad al objetivo. Sin embargo, para aquellos robots que logran alcanzar su meta, la función de fitness considera la distancia total recorrida y el tiempo necesario para llegar. La función se puede expresar de la siguiente manera:

$$f = \begin{cases} 1/(D_e + C_1) & \text{si robot no ha llegado al punto} \\ C_2/(2 * D_r + 3 * T) & \text{si robot ha llegado al punto} \end{cases} \tag{6.2}$$

En esta función, C_1 y C_2 son constantes con valores arbitrarios. C_1 es de un valor muy pequeño, aproximadamente 10^{-7}, para prevenir una división por cero. Por su parte, C_2 es de un valor elevado, asegurando que los robots que alcancen el objetivo obtengan una puntuación de fitness superior, lo que a su vez favorece su reproducción. Las variables D_e y D_r representan la distancia euclídea hasta el destino y la distancia recorrida, respectivamente, mientras que T denota el tiempo de ejecución del robot. Cabe destacar que el tiempo y la distancia son ponderados por 3 y 2, respectivamente, para enfatizar la importancia de alcanzar el destino en el menor tiempo posible.

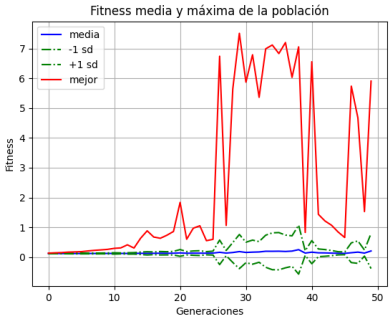
Las métricas que se mostrarán son idénticas a las del entorno anterior: la función de fitness, la distancia recorrida, el tiempo de recorrido y la especiación, en el caso de NEAT.

En estas simulaciones, los robots se han ubicado en una zona repleta de obstáculos, como arbustos, bancos y árboles, y el destino se ha establecido a una distancia considerable. Dada la presencia de estos obstáculos, la ruta hacia el objetivo se convierte en un problema complejo. Así, se evalúa la habilidad de las redes para navegar hacia el objetivo mientras evaden obstáculos.

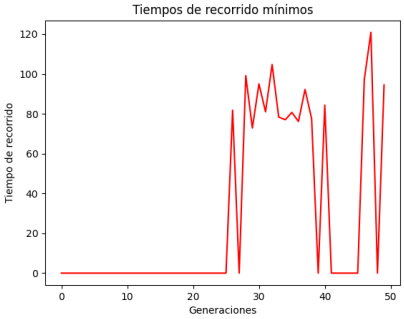
Algoritmo genético clásico con arquitectura simple

En este primer experimento, se ha empleado la implementación del algoritmo genético clásico en combinación con una arquitectura simple. Dicha arquitectura consta de una capa de 9 entradas, correspondientes a los ángulos del robot: 0º, 45º, 90º, 135º y 180º, así como la ubicación del robot y la ubicación del objetivo. Esto se complementa con dos capas ocultas densas de 7 y 4 neuronas, respectivamente, finalizando en la capa de salida.

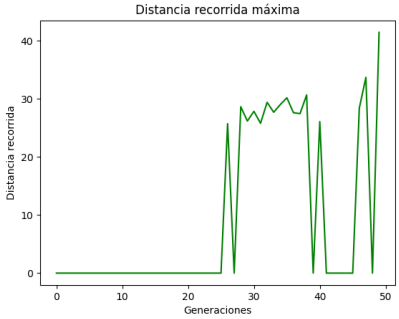
Tras realizar 50 generaciones, se obtuvieron los resultados siguientes:



(a) Fitness



(b) Tiempo de vuelta



(c) Distancia recorrida

El análisis del gráfico de la función de fitness sugiere que los robots cumplieron el objetivo en la generación 26ª. No obstante, a diferencia del experimento paralelo en el entorno anterior, la tendencia no es claramente ascendente, mostrando una serie de deterioros en la puntuación. Esto se puede deber a la dificultad que presentan los obstáculos, además de la falta de señales claras que guíen la ruta. A esto se suma el factor de ruido del LIDAR previamente discutido, que puede afectar negativamente a los individuos. En relación a la media de fitness, se puede observar que no parece mejorar, lo que puede indicar una falta de entrenamiento.

En consonancia con esto, el gráfico de tiempo de recorrido muestra que, a partir de la generación mencionada, se comienza a registrar el tiempo. Se puede apreciar claramente el impacto del tiempo en la función de fitness: cuando el tiempo disminuye, la función de fitness aumenta, y viceversa.

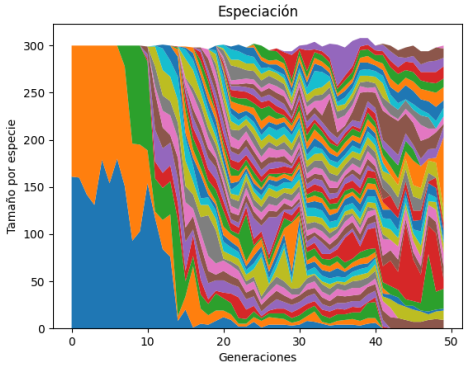
Por último, el gráfico de distancia recorrida muestra un patrón similar al del tiempo en relación con la función de fitness. Se puede percibir la correlación existente entre esta métrica, el tiempo y la función de fitness.

NEAT con arquitectura simple

En este segundo experimento, NEAT ha sido empleado en combinación con una arquitectura simple, parecida a la utilizada en el primer experimento. Esta arquitectura consta de una capa de 9 entradas que corresponden a los ángulos del robot (0º, 45º, 90º, 135º, y 180º), la posición del mismo y la posición del destino. Es relevante destacar que esta arquitectura comienza con la capa de entrada conectada directamente a la de salida, debido a que NEAT se responsabiliza de añadir y eliminar neuronas y conexiones a lo largo de las generaciones.

Los resultados de este experimento, después de 50 generaciones, son los siguientes:

(a) Fitness



(b) Especiación

De la gráfica de la función de fitness, es evidente que el modelo no ha logrado converger en este caso. Sin embargo, un aspecto a resaltar es el incremento progresivo del valor de fitness en general, sugiriendo que con más tiempo de entrenamiento, la convergencia podría ser alcanzada.

Dado que la convergencia no se produjo, las gráficas de tiempo y distancia no han sido incluidas, ya que no se disponen de datos sobre ellas.

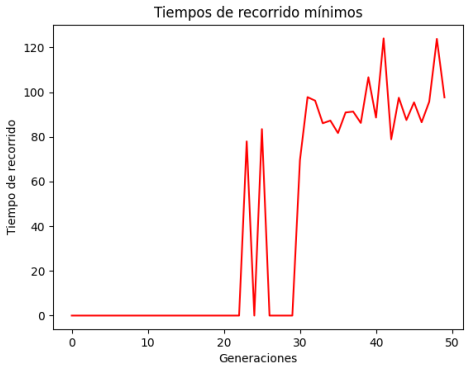
Al llegar a la gráfica de especiación, se observa la creación de un gran número de especies a lo largo de las generaciones, culminando con un total de 24. En particular, desde la generación 20 hasta la 40, se manifiesta una mayor cantidad de especies, pero con un número reducido de individuos. Esto puede atribuirse al hecho de que NEAT, en su esfuerzo por preservar estrategias que potencialmente puedan evolucionar hasta ser óptimas en el futuro, establece un porcentaje de supervivencia para cada especie. Esta gran cantidad de especies, todas con casi el mínimo de individuos, da a entender que el algoritmo aún no ha encontrado una solución satisfactoria y está en proceso de explorar más a fondo el espectro de la solución.

Algoritmo genético clásico con arquitectura compleja

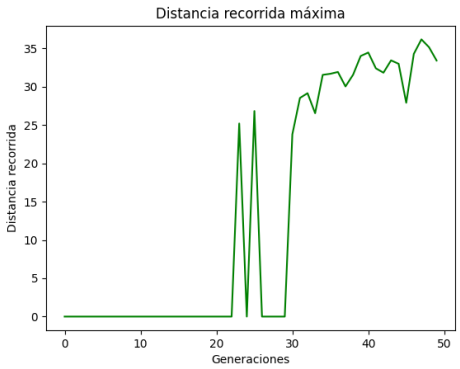
En este experimento se ha empleado el algoritmo genético clásico con una arquitectura compleja. Esta arquitectura está compuesta por una capa de 184 entradas, correspondientes a los ángulos del robot desde 0º hasta 180º. Esta configuración proporciona al robot una visión completa de su parte frontal. La arquitectura también incluye dos capas densas ocultas, una con 92 y la otra con 46 neuronas, y termina con la capa de salida.

Tras realizar 50 generaciones, los resultados obtenidos fueron los siguientes:

(a) Fitness



(b) Tiempo de vuelta



(c) Distancia recorrida

Analizando la gráfica de la función de fitness, se observa que los robots lograron llegar al punto de destino en la generación 23. A diferencia del experimento con la arquitectura simple, el incremento del valor de fitness ha sido más lento al principio, con una mejora notable a partir de la generación 23. Este comportamiento sugiere que la arquitectura compleja tiene una mayor capacidad de aprendizaje, a pesar de que inicialmente se ralentice el proceso. Como en el primer experimento, la función de fitness no sigue una forma ascendente constante, presentándose picos más bajos de lo esperado, lo cual puede ser debido al mismo motivo que en el primer experimento. Aquí, la media de la función comienza a ascender lentamente, posiblemente por la mayor capacidad de la red.

En la gráfica del tiempo de recorrido, se observa que el tiempo comienza a registrarse a partir de la generación 23, lo cual refuerza el impacto del tiempo en la función de fitness.

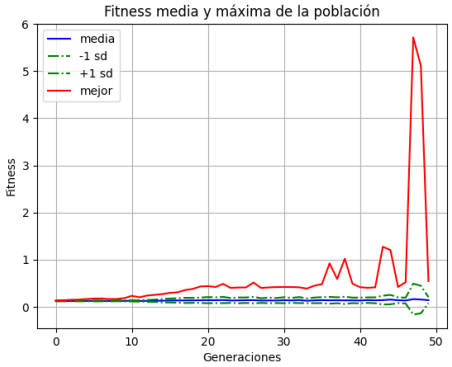
Finalmente, la gráfica de la distancia recorrida ayuda a entender más la descomposición de la función de fitness junto con el tiempo. Sin embargo, no se percibe una mejora permanente clara, lo que puede indicar que la arquitectura puede no ser óptima o que necesita más entrenamiento para empezar a optimizarse.

Aun así, teniendo en cuenta que la función del tiempo sí ha experimentado una mejora, se puede afirmar que el modelo logra optimizar el recorrido.

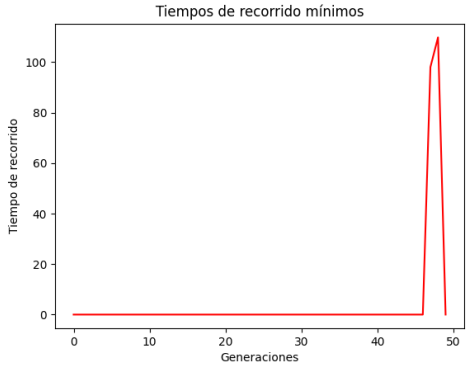
NEAT con arquitectura compleja

Este experimento se llevó a cabo utilizando el algoritmo NEAT con la arquitectura compleja descrita previamente. Dado que NEAT se encarga de modificar dinámicamente el número de neuronas y conexiones, la arquitectura inicial conecta directamente la capa de entrada con la de salida.

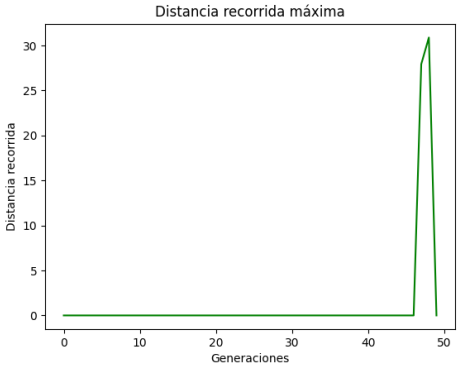
Tras realizar 50 generaciones, se obtuvieron los siguientes resultados:



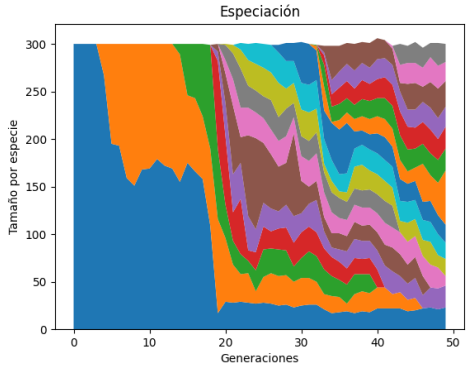
(a) Fitness



(b) Tiempo de recorrido



(c) Distancia recorrida



(d) Especiación

Los resultados indican que los robots lograron completar el objetivo en la generación 46. Se observa un incremento muy lento del valor de fitness, con un aumento notable en la generación 46. La media de las puntuaciones de fitness permanece estable, lo que sugiere un número reducido de especies capaces de alcanzar el objetivo en este experimento.

En relación a las gráficas del tiempo y la distancia, como en los experimentos anteriores, se observa la relación inversamente proporcional en la función de fitness en función de estas dos métricas.

En este experimento, los individuos han aprendido en una etapa tardía del entrenamiento y no han conseguido optimizar el recorrido. Este comportamiento puede estar relacionado con la naturaleza de NEAT, que puede encontrar dificultades para optimizar la arquitectura en un entorno complejo y con muchas entradas en el modelo.

Finalmente, en la gráfica de especiación, se percibe un incremento en el número de especies respecto al experimento homólogo en el circuito. Esto puede atribuirse al aumento de la complejidad, que impulsa al algoritmo a generar mayor diversidad para encontrar soluciones más eficaces, y al prolongado tiempo de entrenamiento.

Reflexión

A continuación, se muestra una tabla con la recopilación de resultados obtenidos en estos 4 experimentos:

Experimento	Convergencia	Fit. máx.	Fit. media	Tiempo	Distancia	Especies
Experimento 1	26	7.51	0.25	72.87978	41.44	-
Experimento 2	-	0.67	0.15	-	-	24
Experimento 3	**23**	**7.80**	**0.35**	**69.58034**	36.17	-
Experimento 4	46	5.71	0.16	98.05556	**30.88**	13

Cuadro 6.2: Tabla comparativa de los experimentos de la ciudad

Según la tabla recopilatoria de los resultados, se puede deducir que el experimento con la configuración más prometedora ha sido el tercero, que también se destacó en el circuito. Este fue el primero en converger, consiguió el valor de fitness más elevado, la media más alta y el mejor tiempo de recorrido. Estos resultados, al ser analizados, indican que los experimentos que emplearon la arquitectura compleja tuvieron un rendimiento superior a los que se valieron de la arquitectura simple. Este resultado puede atribuirse a la mayor capacidad de aprendizaje de la arquitectura compleja y a su campo de visión extendido, que les facilita la creación de estrategias más precisas y optimizadas.

7 Conclusiones

La presente investigación ha explorado en profundidad la combinación de algoritmos genéticos y redes neuronales para el desarrollo de sistemas de navegación autónoma. Los resultados obtenidos han demostrado que esta combinación de técnicas presenta potenciales ventajas sobre enfoques más tradicionales, principalmente en su capacidad de adaptabilidad y aprendizaje frente a entornos cambiantes y complejos, elementos fundamentales en la navegación autónoma.

Se ha comprobado que, mediante los algoritmos genéticos, es posible entrenar y optimizar redes neuronales para gestionar eficazmente tareas de navegación en diversas situaciones, desde circuitos hasta entornos urbanos. El rendimiento obtenido ha sido satisfactorio en casi todos los casos analizados, evidenciando la robustez de la metodología propuesta.

Aunque los resultados de este estudio son prometedores, aún existen muchas áreas por explorar. Recomendamos futuras investigaciones para aplicar y extender esta metodología a una variedad aún mayor de entornos y tareas. Adicionalmente, sería de interés analizar la eficacia de este enfoque en sistemas de navegación autónoma reales, más allá de los entornos simulados.

Bibliografía

[1] Mariana Natalia Ibarra Bonilla. Navegación autónoma de un robot con técnicas de localización y ruteo. *Instituto Nacional de Astrofísica, Óptica y Electrónica*, 2009.

[2] José Gallardo y Ricardo Pérez Gustavo Acosta. Reactive control architecture for autonomous mobile robot navigation. *Ingeniare. Revista chilena de ingeniería*, 24(1):173–181, 2016.

[3] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 1992.

[4] John J. Hopfield. Artificial neural networks. *IEEE Circuits and Devices Magazine*, 4(5): 3 – 10, 1988.

[5] Jorge Luis Martínez Valencia y Germán Andrés Holguín Londoño Jorge Luis Martínez Valencia. A methodology to automaton synthetize in ground autonomous navigation systems control. *Scientia Et Technica*, 23(4):490–500, 2018.

[6] Matt y Miguel Cesar G. y Feher de Silva Carolina y Netto Marcio Lobo McIntyre, Alan y Kallada. Neat python package. https://github.com/CodeReclaimers/neat-python, 2022. URL `https://github.com/CodeReclaimers/neat-python`.

[7] Inc. Open Source Robotics Foundation. Turtlebot 3: waffle. https://www.turtlebot.com/turtlebot3/, 2017. URL `https://www.turtlebot.com/turtlebot3/`.

[8] Open Robotics. Gazebo. https://gazebosim.org/home, 2002. URL `https://gazebosim.org/home`.

[9] Open Robotics. Robot operating system (ros). https://www.ros.org/, 2007. URL `https://www.ros.org/`.

[10] Open Robotics. Ros messages. http://wiki.ros.org/msg, 2023. URL `http://wiki.ros.org/msg`.

[11] Paulina Sroka. Low poly city assets. https://assetstore.unity.com/packages/3d/environments/urban/low-poly-city-assets-234586, 2023. URL `https://assetstore.unity.com/packages/3d/environments/urban/low-poly-city-assets-234586`.

[12] Matteo Caruso Paolo Gallina y Eric Medvet Stefano Seriani, Luca Marcini. Crowded environment navigation with neat: Impact of perception resolution on controller optimization. *Journal of Intelligent Robotic Systems*, 101(36), 2021.

[13] Unity Technologies. Unity. https://unity.com/es, 2005. URL `https://unity.com/es`.

[14] Unity Technologies. Unity ros tcp connector. https://github.com/Unity-Technologies/ROS-TCP-Connector, 2021. URL https://github.com/Unity-Technologies/ROS-TCP-Connector.

[15] Unity Technologies. Unity ros tcp endpoint. https://github.com/Unity-Technologies/ROS-TCP-Endpoint, 2021. URL https://github.com/Unity-Technologies/ROS-TCP-Endpoint.

[16] Tomas van Rietbergen. Toward reliable robot navigation using deep reinforcement learning. http://resolver.tudelft.nl/uuid:8dda87ed-5acd-44fc-9a31-e8a60b20f43b, 2022. URL http://resolver.tudelft.nl/uuid:8dda87ed-5acd-44fc-9a31-e8a60b20f43b.

[17] Donal E. Brown y Chelsea C. White. *Operations Research and Artificial Intelligence: The Integration of Problem-Solving Strategies*, pages 29–53. Kluwer Academic Publishers, 1990.

[18] Kenneth O. Stanley y Risto Miikkulainen. Evolving neural networks through augmenting topologies. *The MIT Press Journals*, 10(2), 2002.

Lista de Acrónimos y Abreviaturas

AAGG	Algoritmos genéticos.
AG	Algoritmo genético.
DRL	Aprendizaje por refuerzo profundo.
IA	Inteligencia artificial.
LIDAR	Light Detection and Ranging.
NEAT	Neuroevolución de Topologías Aumentadas.
RL	Aprendizaje por refuerzo.
RNA	redes neuronales artificiales.
ROS	Robot Operating System.
TFG	Trabajo Final de Grado.
URDF	Unified Robot Description Format.