ROBÓTICA EVOLUTIVA PARA RESOLUCIÓN DE LABERINTOS

SIMULADOR IRSIM

Diseño, entrenamiento y análisis de controladores neuronales evolutivos para un robot autónomo en entorno simulado

ÁLVARO GARCÍA RUIZ-ESCRIBANO | GRUPO 10

ÍNDICE GENERAL

INTRODUCCIÓN

El objetivo de este proyecto es diseñar, entrenar y evaluar un controlador neuronal capaz de guiar a un robot a través de un laberinto.

El experimento se lleva a cabo en el simulador IRSIM sobre una arena cuadrada de 3x3 m compuesta por pasillos delimitados por muros, una casilla de entrada (ubicada en el centro de la arena) y una meta señalizada mediante una luz amarilla.

El entorno también incorpora obstáculos representados por zonas grises, iluminadas con luces rojas, que el robot debe aprender a evitar. El comportamiento del robot se optimiza mediante algoritmos genéticos, empleando distintas configuraciones de redes neuronales: una red artificial estática (ANN) y su extensión a una red neuronal dinámica (CTRNN), permitiendo comparar sus rendimientos.

La memoria se estructura en tres partes:

1. Descripción del entorno de simulación.
2. Implementación, análisis y mejoras de la red ANN.
3. Extensión del experimento mediante una red CTRNN y su comparación con el enfoque estático.

ARENA

La arena tiene una dimensión de 3x3 metros y representa un espacio cerrado compuesto por pasillos que simulan el interior de un laberinto. Aunque durante el desarrollo se probarán diferentes configuraciones, la red neuronal será entrenada principalmente sobre una distribución base específica, representada en la Figura 1.

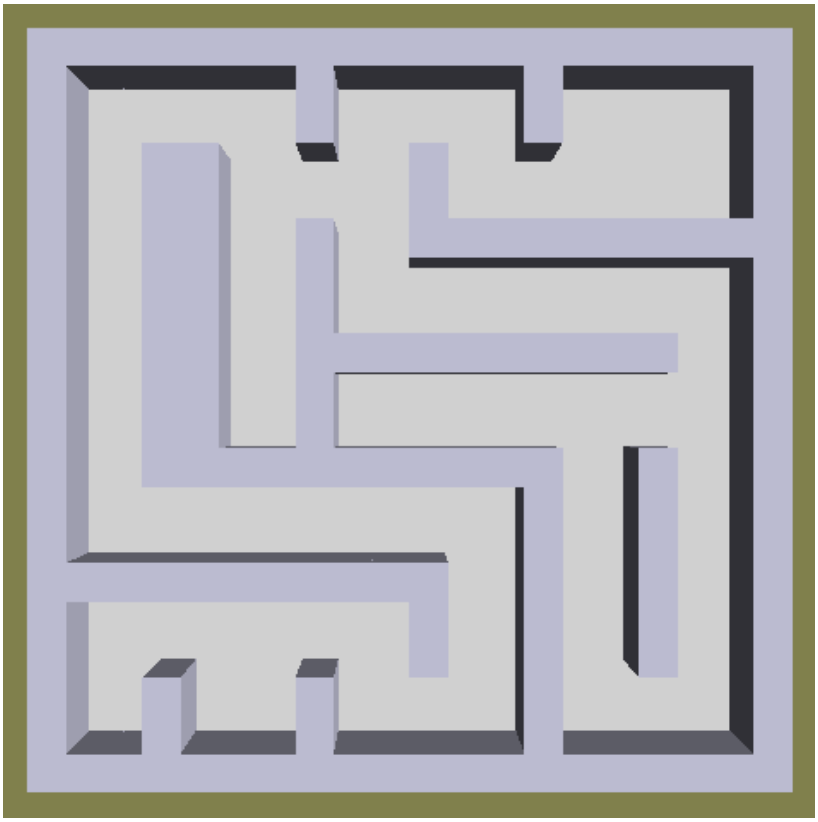

Figura 1: Distribución de los muros en la arena de entrenamiento.

Además de los muros, la arena incorpora los siguientes elementos clave:

- **Luz amarilla**: Marca la salida del laberinto y sirve como señal para indicar que el robot ha alcanzado su objetivo.
- **Zonas grises**: Representan vacíos que el robot debe evitar.
- **Luces rojas**: Situadas en el centro de cada zona gris, permiten al robot identificar estos vacíos mediante sensores de luz roja.

Todos los elementos del entorno se colocan en el centro de una malla de 20x20 divisiones. La posición inicial del robot se establece en el centro de la arena (coordenadas (0,0)) con una orientación de 0 grados.

La Figura 2 muestra las coordenadas utilizadas como referencia para distribuir los elementos en las casillas libres de la arena. En la Figura 3 se presenta la disposición final del entorno de entrenamiento, incluyendo la posición del robot, la luz amarilla, las zonas grises y las luces rojas.

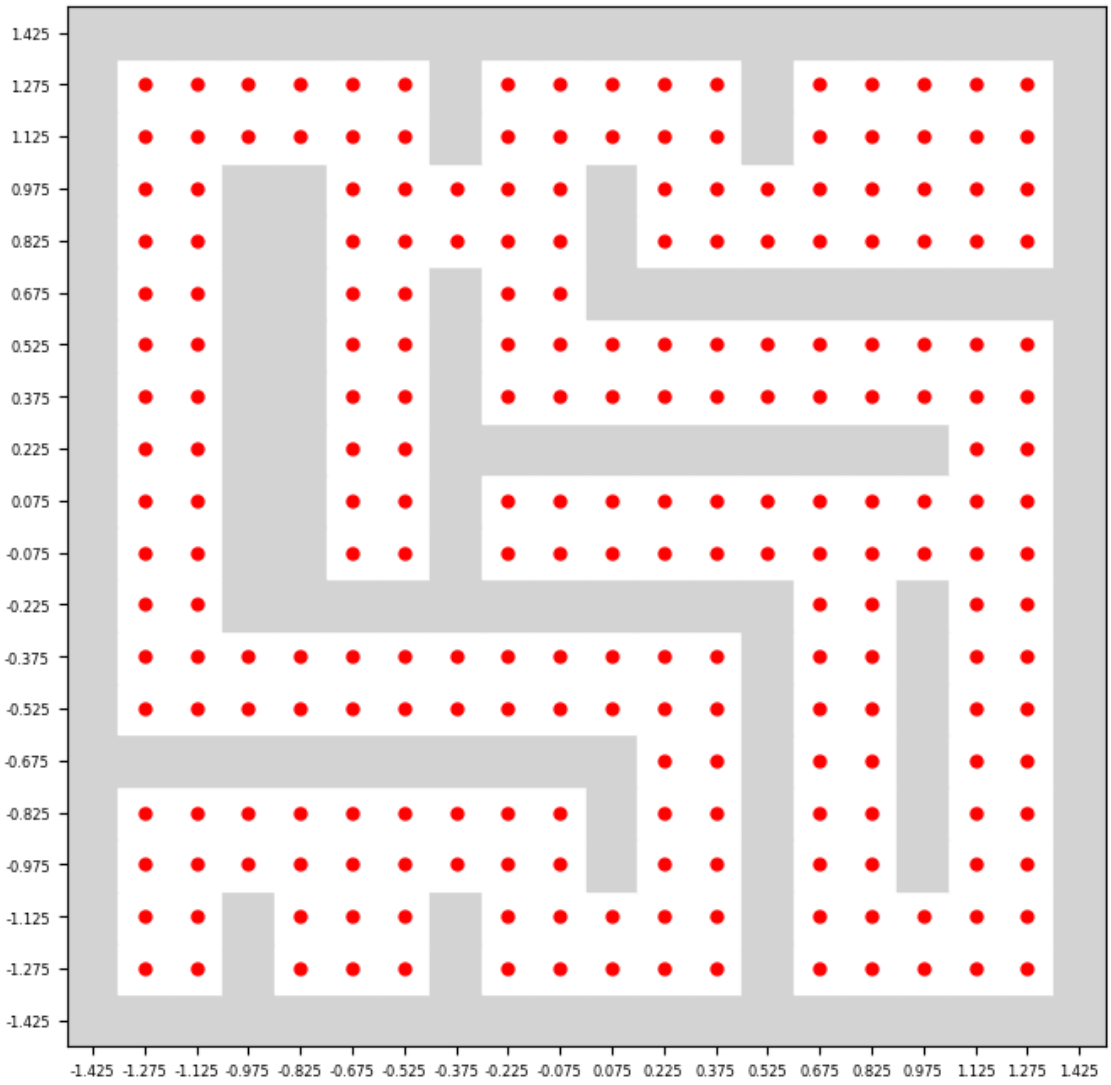

Figura 2: Grafico de las coordenadas de los centros de las casillas libres en la arena de entrenamiento

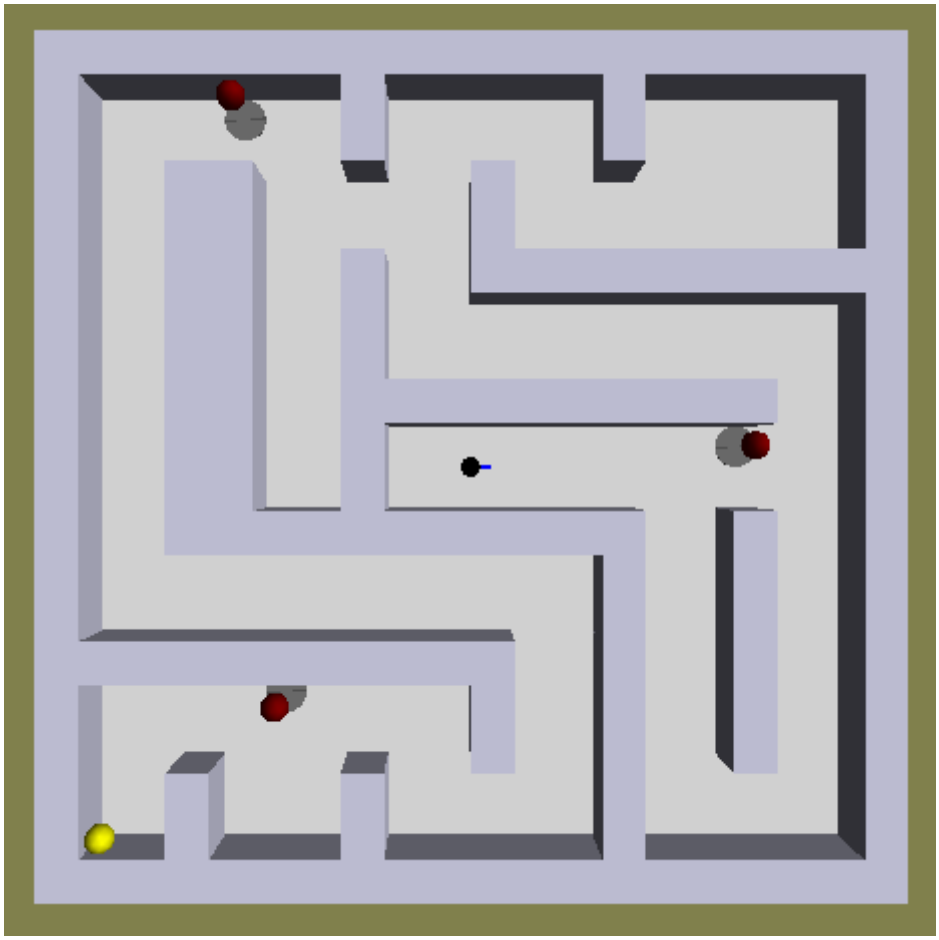

Figura 3: Distribución de los elementos y del robot en la arena de entrenamiento.

Como parte de la configuración del entorno, fue necesario modificar el archivo redlightobject.cpp para desactivar la secuencia de encendido/apagado definida en el método GetTiming, asegurando una detección constante de las zonas grises.

SENSORES

El comportamiento del robot se basa en la información obtenida mediante diversos sensores integrados, que permiten interpretar el entorno y adaptar sus acciones. A continuación, se describen los sensores empleados, su funcionalidad principal y, cuando es relevante, su rango de actuación:

- **Sensor de proximidad:** Permite detectar y evitar obstáculos en el entorno.
- **Sensor de suelo con memoria:** Identifica si el robot ha pisado una zona gris.
- **Sensor de luz roja:** Permite identificar zonas grises antes de alcanzarlas. Tiene un alcance limitado de 0.3 metros, para ubicar las zonas grises a una casilla de distancia.

- **Sensor de luz amarilla:** Localiza la meta del laberinto. Con un alcance de 4.5 metros, es capaz de detectar la salida desde cualquier posición dentro de la arena.
- **Sensor de encoder:** Estima la posición del robot mediante odometría.

RED NEURONAL ANN

ARQUITECTURA

La primera versión del controlador del robot se basa en una red neuronal artificial estática (ANN) de arquitectura simple. Esta red conecta directamente las entradas sensoriales con una capa motora sin incluir capas ocultas o asociativas. A pesar de su simplicidad, esta configuración se ha demostrado funcional para la tarea propuesta.

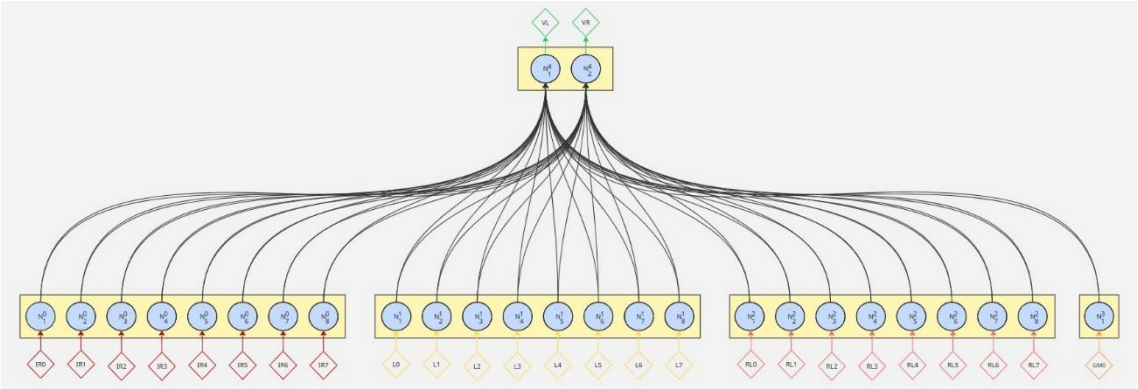

Figura 4: Arquitectura de la red neuronal ANN.

La explicación en detalle de cada capa de esta red neuronal está recogida en la siguiente tabla.

Capa	Neuronas	Naturaleza	Función de activación	Entrada	Salida
0	8	Capa sensorial	Identidad	8 sensores de proximidad	Capa 4
1	8	Capa sensorial	Identidad	8 sensores de luz	Capa 4
2	8	Capa sensorial	Identidad	8 sensores de luz roja	Capa 4
3	1	Capa sensorial	Identidad	1 sensor de suelo con memoria	Capa 4

4	2	Capa motora	Sigmoide	Demás capas	Motores izquierdo y derecho

Figura 5: Capas de la red neuronal ANN.

El cálculo de la longitud del cromosoma es el siguiente:

$$\boldsymbol{Longitud\ del\ Cromosoma} = \underbrace{(8*2) + (8*2) + (8*2) + (1*2)}_{\text{Pesos}} + \underbrace{2}_{\text{Sesgos}} = \mathbf{52}$$

Aunque se evaluó la posibilidad de incorporar una capa adicional que recibiera las lecturas de los sensores de encoder, esta fue descartada tras múltiples pruebas defectuosas. Se observó que la naturaleza instantánea de estos sensores dificultaba su aprovechamiento como entrada directa en la red. No obstante, sus datos se integran en la función de fitness como penalización por trayectorias repetidas.

Para poder hacer uso de estos sensores tanto en la red neuronal como en la función de fitness, se realizaron modificaciones en diversos archivos del experimento, como en los del controlador de red neuronal (nndistributedcontroller), en los propios del experimento (testneuronexp), en los sensores de encoder y en el main.cpp. Aunque la implementación final no incluyó esta capa, dichos cambios se entregan como parte del trabajo.

FUNCIÓN DE FITNESS

Para evaluar la eficacia del comportamiento del robot, se ha diseñado una función de fitness compuesta por múltiples componentes. Esta función combina distintos factores que premian conductas deseables (como avanzar hacia la meta o evitar obstáculos) y penalizan comportamientos ineficientes o peligrosos (como colisiones, trayectorias repetidas o pisar zonas grises).

La expresión matemática global de la función se puede representar de forma compacta como:

$$Fitness = \sum_{i=0}^{Nsteps} \frac{((0{,}7\Delta Y + 0{,}2Fwd + 0{,}1wallFactor) * wallSafe * pRevisit)}{Nsteps} * \text{factorColl} * \text{forPro}$$

Cuyos parámetros desglosados representan lo siguiente.

PROGRESO HACIA LA META

- **Significado práctico:** Incremento instantáneo del mayor valor de luz amarilla visto hasta ese paso. Aunque no equivale exactamente al progreso real hacia la meta en todos los escenarios, ayuda a tener una estimación de la cercanía.
- **Qué premia o penaliza:** Premia el acercamiento continuo a la salida.
- **Expresión expandida:**
  Sea **b** el valor de bestY persistente (inicialmente b = 0), y sea **Y** = maxLightSensorEval:

$$\Delta Y = \begin{cases} Y - b, & si\ Y > b + 10^{-3} \\ 0, & en\ caso\ contrario \end{cases}$$

$$b = \begin{cases} Y, & si\ \Delta Y > 0 \\ 0, & \Delta Y = 0 \end{cases}$$

$$bnuevo = \begin{cases} Y, & si\ Y > bviejo + 10^{-3} \\ bviejo, & en\ caso\ contrario \end{cases}$$

$$\Delta Y = bnuevo - bviejo$$

AVANCE RÁPIDO RECTO

- **Significado práctico:** Combina lo rápido que giran ambas ruedas y lo parecidas que son sus velocidades para favorecer trayectorias rectilíneas y rápidas.
- **Qué premia o penaliza:** Premia ir rápido y en línea recta. Si gira o frena Fwd decae.
- **Expresión expandida:**
  Sea **wL** y **wR** las velocidades reales de las ruedas izquierda y derecha; y **M** = m_pcEpuck->getMaxWheelSpeed() su valor máximo:

$$sL = 0.5 + \frac{wL}{2M} \quad sR = 0.5 + \frac{wR}{2M}$$

$$maxSpeedEval = |sL - 0.5| + |sR - 0.5|$$

$$\boldsymbol{sameDirectionEval} = \mathbf{1} - \sqrt{|\boldsymbol{sL} - \boldsymbol{sR}|}$$

$$\boldsymbol{Fwd} = \boldsymbol{maxSpeedEval} * \boldsymbol{sameDirectionEval}$$

SEGUIDOR DE MURO

- **Significado práctico:** Busca que el sensor lateral de proximidad izquierdo prox(2) lea ≃ 0.70, ni muy pegado ni muy lejos, pero aplicado solo cuando hay progreso (ΔY > 0).
- **Qué premia o penaliza:** Mantener una distancia cómoda al muro izquierdo mientras avanza.
- **Expresión expandida:**

$$\boldsymbol{wallGauss} = \boldsymbol{exp}\left(-\frac{(\boldsymbol{prox}[\mathbf{2}] - \mathbf{0.7})^{\mathbf{2}}}{\mathbf{2}(\mathbf{0.08})^{\mathbf{2}}}\right)$$

$$\boldsymbol{wallFactor} = \begin{cases} \mathbf{0.5} + \mathbf{0.5}\boldsymbol{wallGauss}, & \boldsymbol{\Delta Y} > \mathbf{0} \\ \mathbf{0.5}, & \boldsymbol{\Delta Y} \leq \mathbf{0} \end{cases}$$

DISTANCIA A OBSTACULOS FRONTALES

- **Significado práctico:** Margen frontal de seguridad, castiga si alguno de los dos sensores delanteros de proximidad (0 ó 7) ve el obstáculo demasiado cerca.
- **Qué premia o penaliza:** Cuanto más cerca de un obstáculo frontal, más se reduce la fitness.
- **Expresión expandida:**

$$\boldsymbol{wallSafe} = \mathbf{1} - \boldsymbol{max}\{\boldsymbol{prox}[\mathbf{0}], \boldsymbol{prox}[\mathbf{7}]\}$$

REVISITAR CELDAS

- **Significado práctico:** Penalización aplicada cuando la celda actual (0.05 x 0.05 m) se ha visitado más de 2 veces. Se calcula haciendo uso de las lecturas de los sensores de encoder, mediante odometría. Incentiva la exploración nueva y evita bucles.
- **Qué premia o penaliza:** Penaliza un 10% si se revisita una celda en la que ya se ha estado
- **Expresión expandida:**
  Sea **v** la última celda visitada:

$$pRevisit = \begin{cases} 1, & v \leq 2 \\ 0.1, & v > 2 \end{cases}$$

FACTOR DE COLISIONES

- **Significado práctico:** Reduce la fitness en función del número de colisiones.
- **Qué premia o penaliza:** Cada colisión (hasta 10) resta un 10 % de la fitness final.
- **Expresión expandida:**
  Sea **Ncoll** el número de colisiones en un step:

$$factorColl = 1 - min\left\{\frac{Ncoll, 10}{10}\right\}$$

FORZADO A PROGRESAR

- **Significado práctico:** Penalización global multiplicativa de valor fijo (0.999), forzando a progresar hacia la meta
- **Qué premia o penaliza:** Incentiva a completar el laberinto en menos tiempo.
- **Expresión expandida:**

$$forPro = 0.999$$

EVENTOS TERMINALES

Además de la expresión matemática general, se han definido dos eventos terminales:

- **Alcanzar la meta**. Si algún sensor de luz amarilla supera el umbral de 0.95, se asigna una fitness de 1.0.
- **Pisar una zona gris**. Si se detecta que el robot entra en una zona gris, su fitness se reduce automáticamente a 0.0 para el resto del episodio.

PARAMETROS DE CONFIGURACIÓN

A continuación, se detallan los parámetros principales utilizados en la configuración del entorno, el algoritmo evolutivo y la red neuronal. Estos valores han sido seleccionados tras múltiples iteraciones exploratorias, buscando un equilibrio entre capacidad de aprendizaje y estabilidad durante el entrenamiento.

MORPHOLOGY

Para evitar ruido innecesario, se han activado exclusivamente los sensores conectados a la red neuronal. La configuración es la siguiente:

PROXIMITY SENSORS USED (8 sensors: 0 OFF, 1 ON) = 1 1 1 1 1 1 1 1

CONTACT SENSORS USED (8 sensors: 0 OFF, 1 ON) = 0 0 0 0 0 0 0 0

LIGHT SENSORS USED (8 sensors: 0 OFF, 1 ON) = 1 1 1 1 1 1 1 1

BLUE LIGHT SENSORS USED (8 sensors: 0 OFF, 1 ON) = 0 0 0 0 0 0 0 0

RED LIGHT SENSORS USED (8 sensors: 0 OFF, 1 ON) = 1 1 1 1 1 1 1 1

GROUND SENSORS USED (3 sensors: 0 OFF, 1 ON) = 1 1 1

ENCODER SENSORS USED (2 sensors: 0 OFF, 1 ON) = 0 0

GENETIC

CHROMOSOME LENGTH= 52

POPULATION SIZE = 120

NUMBER OF GENERATIONS = 151 *Posteriormente ampliado a 2001*

EVALUATION TIME = 500 *Tiempo más que suficiente para completar cualquier laberinto y castigar comportamientos no deseados*

DO CROSSOVER (0 No, 1 Yes) = 1

NUMBER OF CROSSOVERS (Always 1) = 1

CROSSOVER DISTANCE (Always 1) = 1

MUTATION RATE = 0.02 *Mutación del 2%*

NUMBER OF ELITES = 4 *Individuos mejor evaluados que se conservan sin alteraciones*

FITNESS FUNCTION = 4 F*unción de fitness personalizada*

SAMPLES PER CHORMOSOME = 1

RANDOM POSITION ORIENTATION (0 NO, 1 YES) = 0 *Aunque sería una buena implementación para fomentar el aprendizaje del robot para un caso general y no el específico, por la naturaleza de ser un laberinto es más conveniente el tener un inicio fijo. Para activarse, sería necesario implementar una condición para que los individuos no fuesen generados superpuestos a muros o elementos.*

NEURAL

WEIGTH UPPER BOUND = 2.0

WEIGTH LOWER BOUND = -2.0 *Estos valores son adecuados para cubrir un rango amplio sin provocar saturación excesiva en la función sigmoide*

La definición estructural de la red neuronal (capas y conexiones) se ha analizado previamente en el apartado de arquitectura.

ANÁLISIS DE RESULTADOS

Para evaluar el rendimiento del controlador neuronal, se ha llevado a cabo una serie de experimentos con distintas configuraciones evolutivas. A continuación, se presentan y analizan los resultados más representativos:

EVOLUCIÓN DE LA FITNESS

En una primera fase de entrenamiento con 150 generaciones, se obtuvo un individuo con una fitness máxima de 0.7311. Como era de esperar, el individuo con peor desempeño en cada generación obtuvo una fitness de 0, debido a la penalización terminal por más de 10 colisiones o por pisar zonas grises.

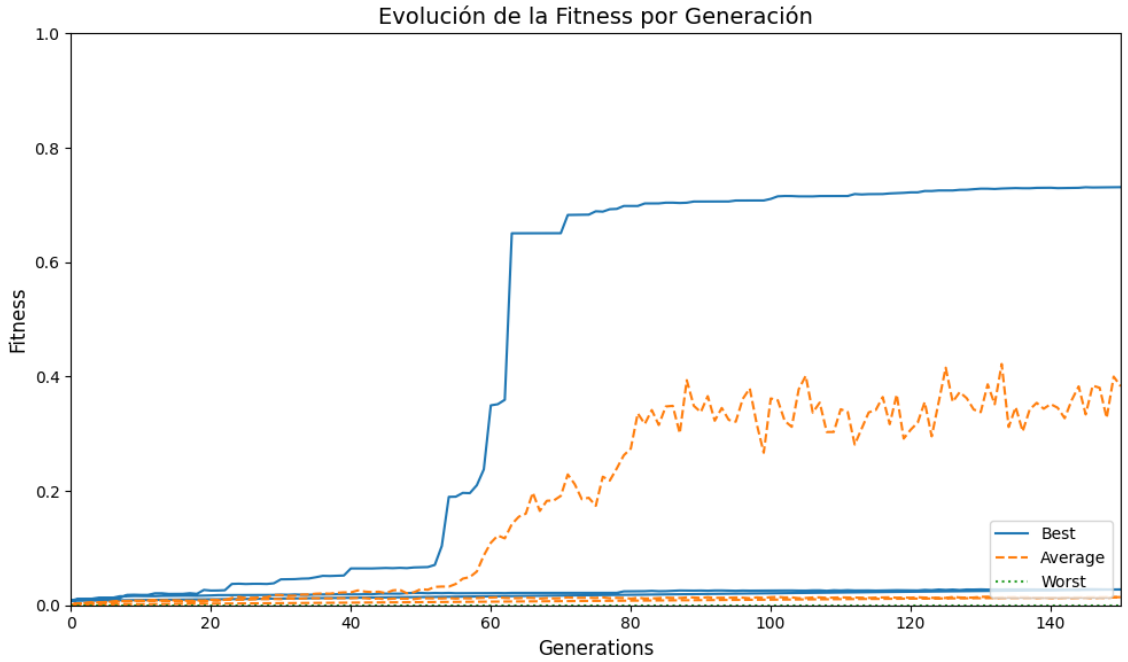

Figura 6: Gráfica de evolución de fitness en 150 generaciones

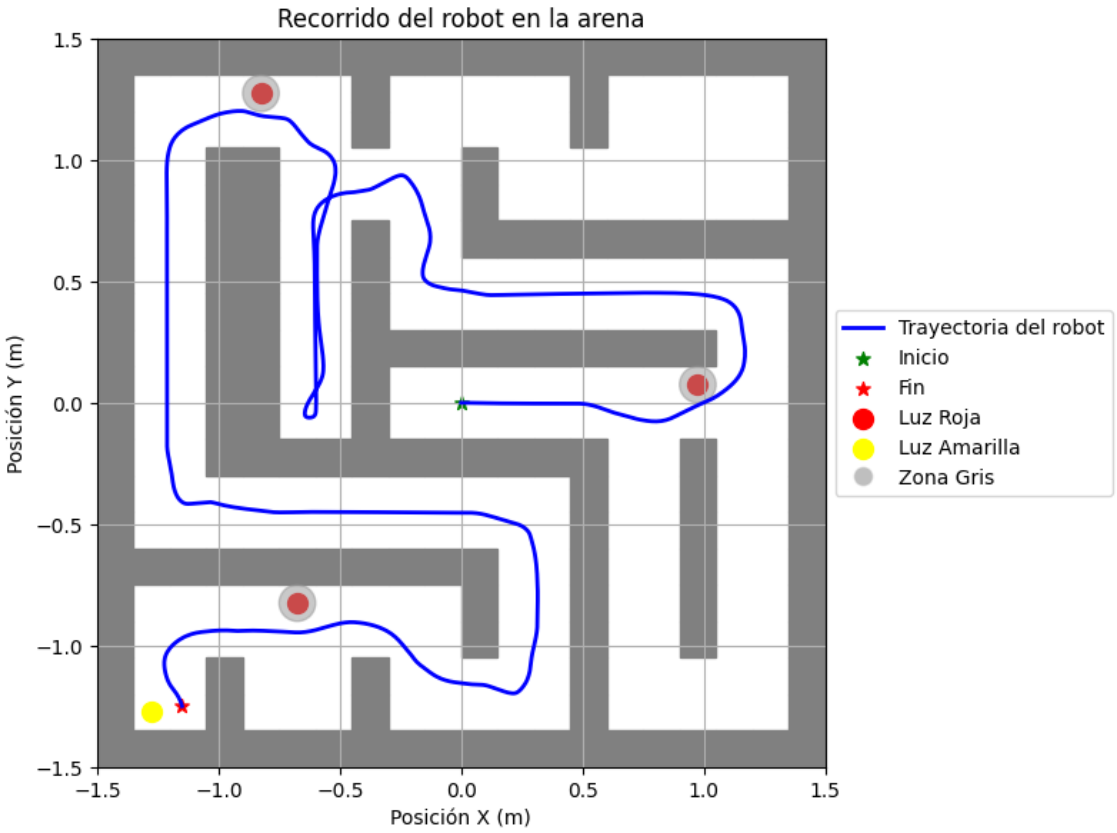

Figura 7: Trayectoria del mejor individuo en gen. 150

Este robot siguió de forma eficiente el muro izquierdo, esquivando zonas grises y deteniéndose al alcanzar la meta. El tiempo empleado fue de 141.9 steps.

Posteriormente, se prolongó la evolución hasta 2000 generaciones, alcanzando una fitness máxima de 0.7724 en la generación 1326. A partir de la generación 500, las mejoras fueron marginales, indicando una posible estabilización del aprendizaje.

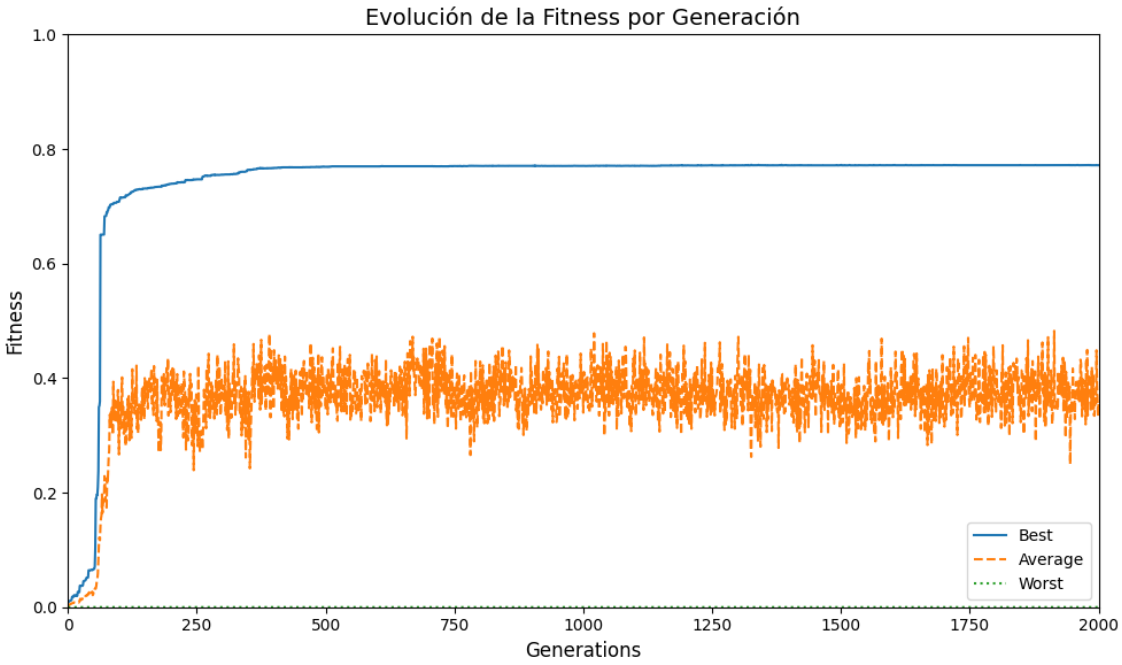

Figura 8: Fitness hasta generación 2000

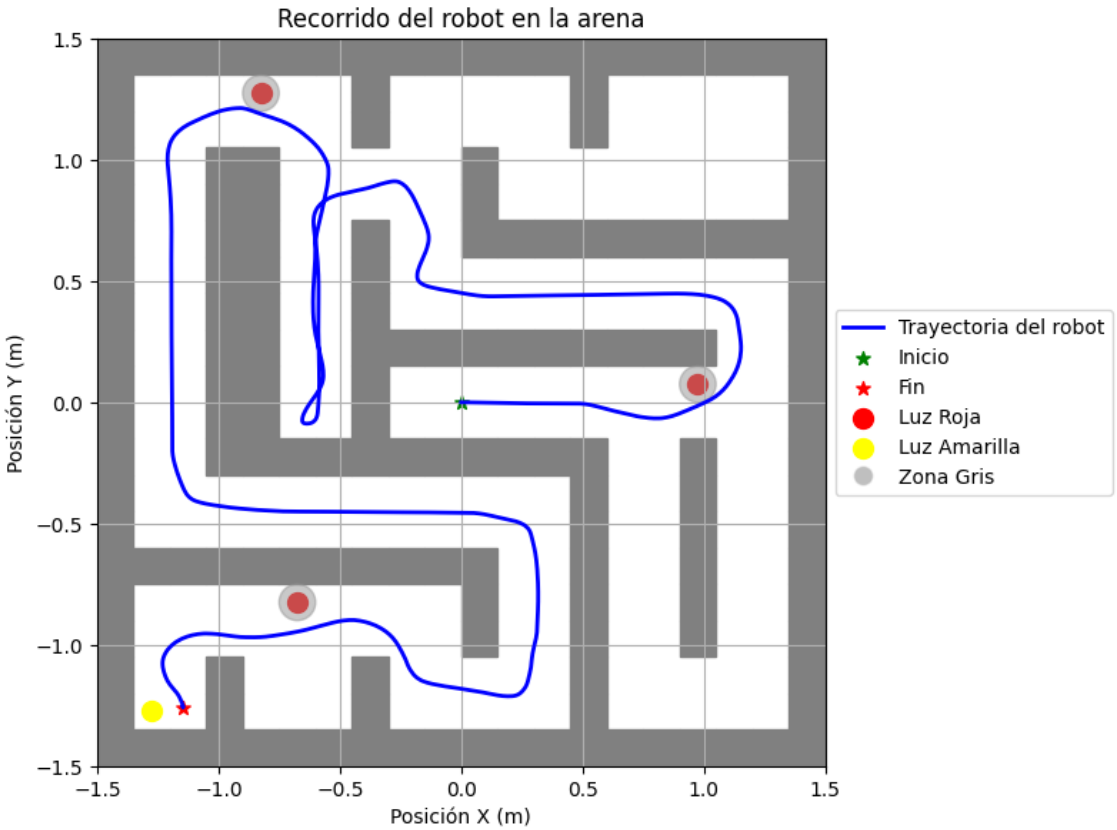

Figura 9: Trayectoria del mejor individuo en gen. 2000

En esta versión optimizada, el robot repitió esencialmente la misma trayectoria que en la generación 150, pero con un movimiento más limpio y veloz, reduciendo el tiempo a 120.7 pasos.

COMPORTAMIENTO EN GENERACIONES TEMPRANAS

Un análisis curioso que se puede hacer es ver cómo se comportaba el robot en una fase aún más temprana de la evolución. Para ello vamos a analizar el comportamiento del individuo que consiguió la mejor fitness en la generación 53. La elección de esta generación se debe a ser la primera que "aprende" a quedarse en la luz amarilla de manera estática una vez completado el laberinto.

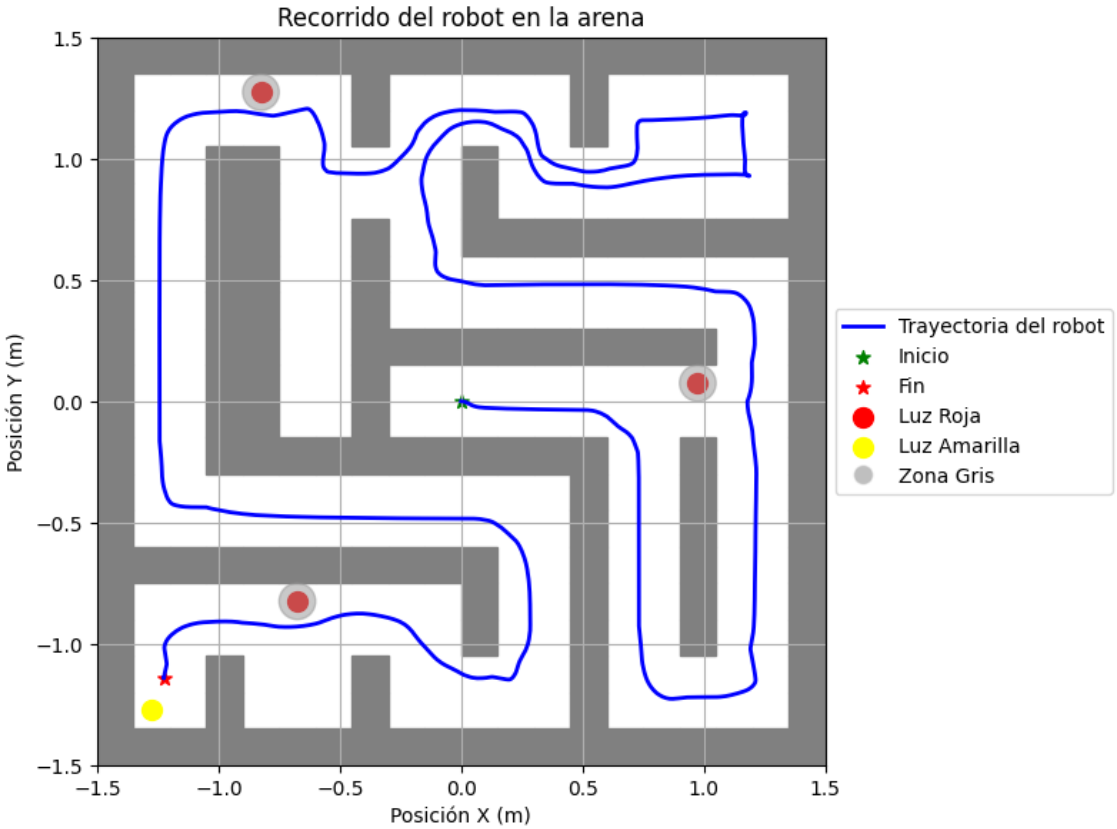

Figura 10: Trayectoria mejor individuo gen. 53.

Aunque este individuo consigue una fitness reducida de 0.1038, se está comportando de manera aceptable, ya que completa el laberinto sin volver sobre sus pasos y sin caer en las zonas grises. Lo que es curioso es su manera de completar el laberinto, y es que aunque la función de fitness este codificada de forma que va a premiar el funcionar como un seguidor del muro izquierdo, este se comporta de forma totalmente contraria, siguiendo el muro derecho.

EVALUACIÓN EN OTROS ESCENARIOS

Se probaron distintos individuos en un escenario alternativo diferente al usado en el entrenamiento, manteniendo reglas similares (mayoria de pasillos de 2 casillas, sin islas, zonas grises de diametro 1 casilla), pero con diferente disposición de muros y obstáculos.

- **Mejor individuo**: Sigue correctamente el muro izquierdo, pero falla al esquivar ciertas zonas grises.
- **Gen. 53**: Comportamiento seguidor de muro derecho con mejor comportamiento ante pasillos amplios, pero se queda atascado frente a una zona gris.
- **Gen. 100**: Se estanca en un bucle ante un pasillo ancho.

Figura 11: Trayectoria escenario alternativo mejor individuo.

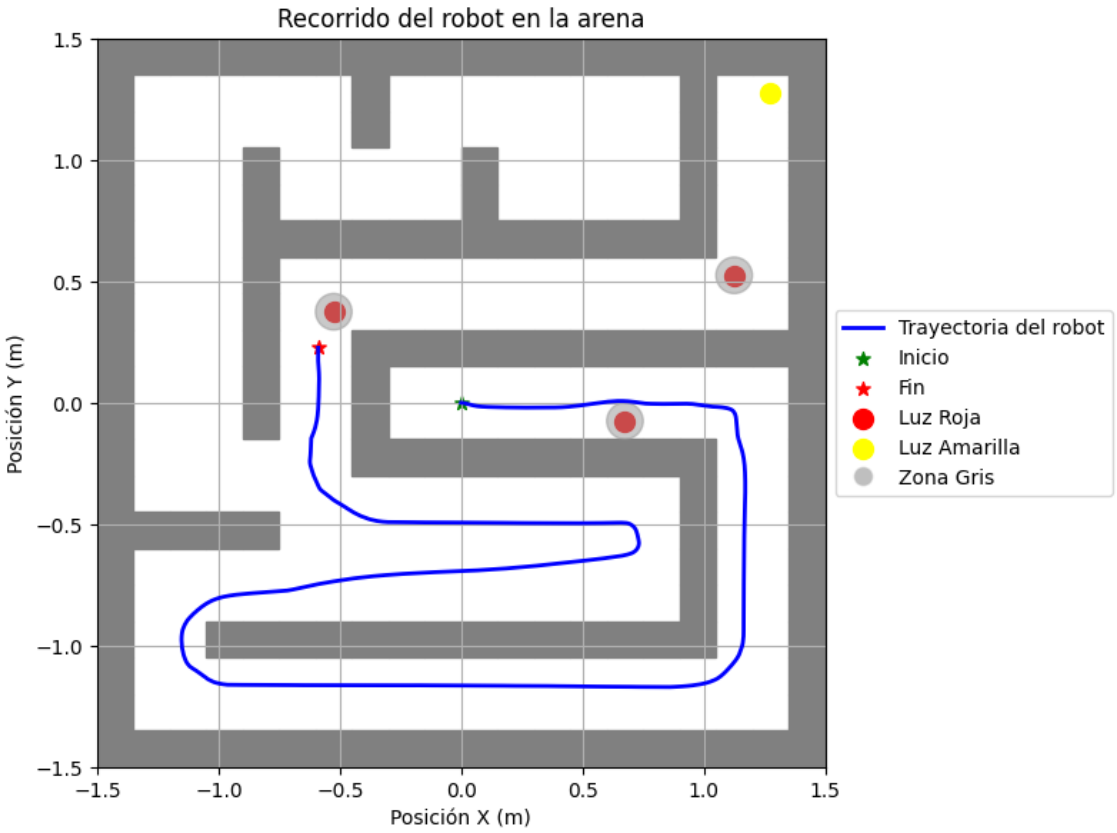

Figura 12: Trayectoria escenario alternativo gen. 53.

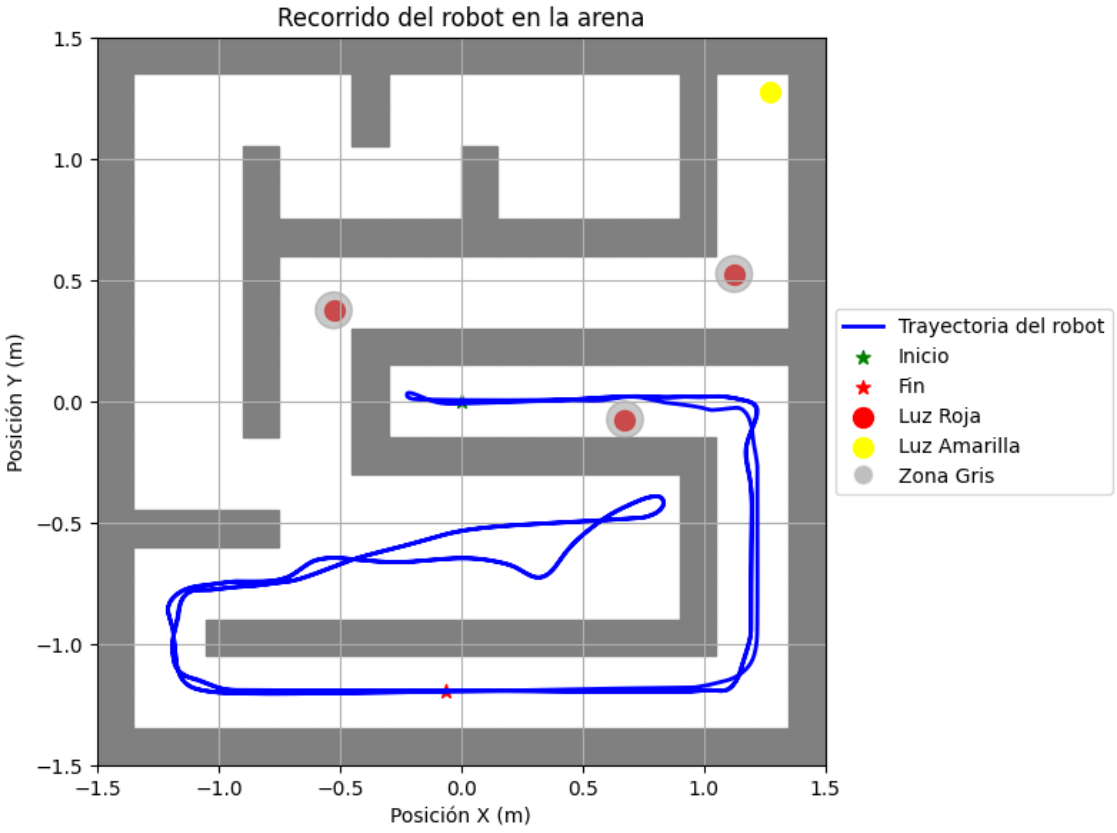

Figura 13: Trayectoria escenario alternativo gen. 100.

Estos resultados indican cierta falta de generalización, debida a un underfitting en cuanto a la variedad de situaciones diferentes durante el entrenamiento. Aunque se probaron generaciones más avanzadas, en ciertos escenarios no superan el desempeño de generaciones más tempranas. Lo que podría considerarse un principio de overfitting funcional.

En conjunto, se concluye que el diseño de la función de fitness ha guiado correctamente el aprendizaje dentro del entorno entrenado, pero que la falta de diversidad en el entrenamiento limita la transferencia del comportamiento aprendido a otros escenarios.

MODIFICACIONES Y MEJORAS

Tras observar ciertas limitaciones en el comportamiento del robot, especialmente su escasa capacidad de generalización ante nuevos escenarios, se plantearon diferentes modificaciones estructurales en el sistema, con el fin de mejorar su rendimiento global. A continuación, se analizan tres enfoques experimentales.

CAMBIO EN LA ARQUITECTURA DE LA RED NEURONAL

Se diseñó una arquitectura más compleja incorporando una capa oculta adicional entre las capas sensoriales de proximidad, luz roja y suelo con memoria; y la capa motora. Esta modificación buscaba que el robot desarrollase una representación más abstracta de su entorno, especialmente para aprender a bordear eficazmente zonas grises según su la distribución de muros que las rodean.

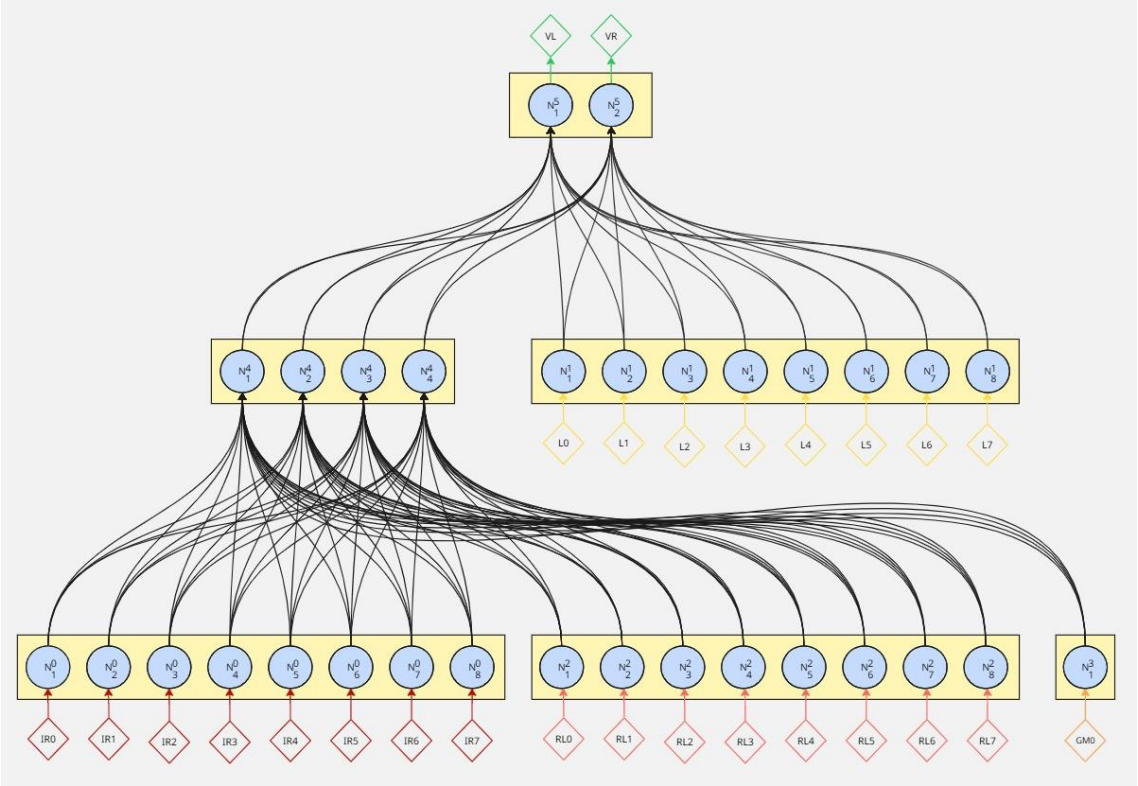

Figura 14: Arquitectura de la nueva red neuronal.

$$\textbf{\textit{Nueva Longitud del Cromosoma}} = \underbrace{(8*4) + (8*2) + (8*4) + (1*4) + (4*2)}_{\text{Pesos}} + \underbrace{2+4}_{\text{Sesgos}} = \mathbf{98}$$

El entrenamiento se llevó a cabo durante 1000 generaciones, sin obtener mejoras sustanciales: la mejor fitness alcanzada fue 0.021. El robot no logró completar el laberinto, indicando que la combinación de esta arquitectura con la función de fitness actual no es funcional.

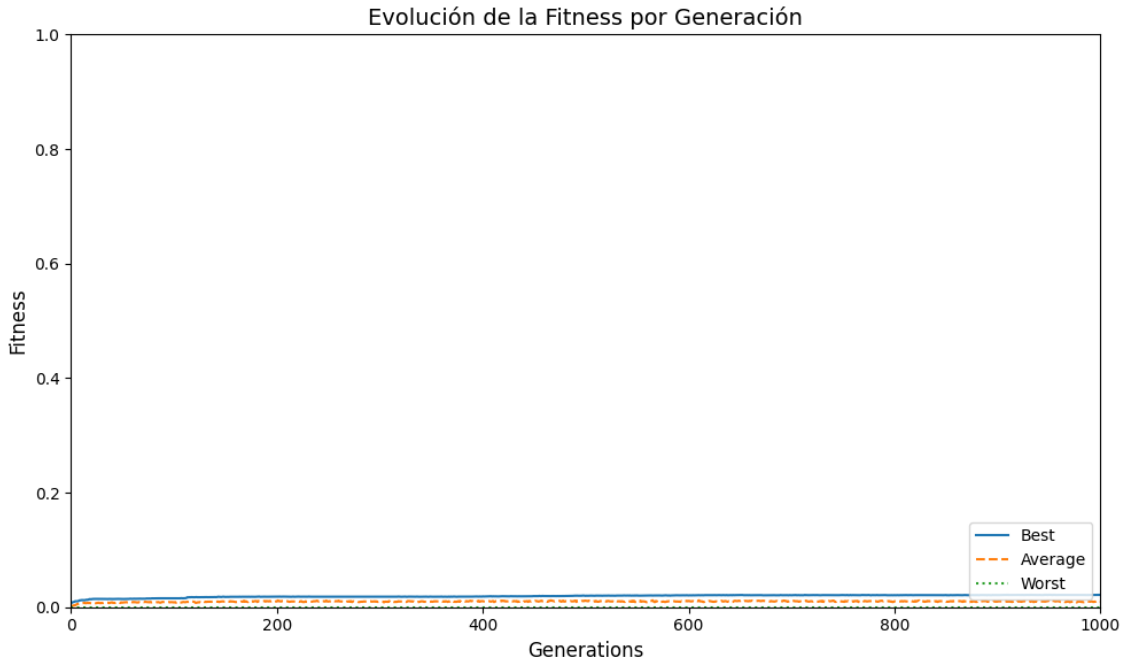

Figura 15: Gráfica evolución de la fitness con cambio de red neuronal.

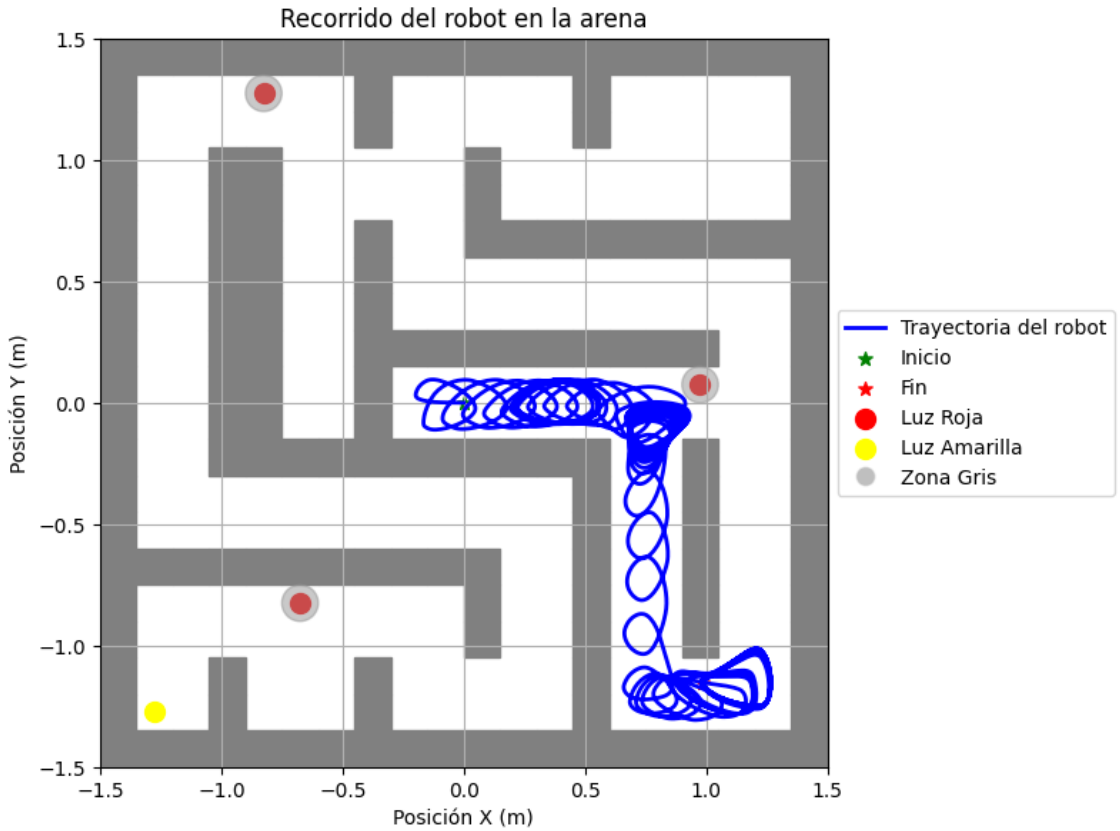

Figura 16: Trayectoria del mejor individuo con cambio de red neuronal.

Esta falta de progreso sugiere que el problema no radica necesariamente en la complejidad de la arquitectura, sino en una incompatibilidad con la función de evaluación actual. Una línea futura prometedora podría ser aumentar la penalización por revisitar celdas.

CAMBIO DEL ESCENARIO DE ENTRENAMIENTO

Siendo el principal problema el underfitting, específicamente en lo relativo al comportamiento ante zonas grises, esta segunda estrategia consistió en incrementar la variedad obstaculos, duplicando el número de zonas grises y luces rojas en el entorno de entrenamiento.

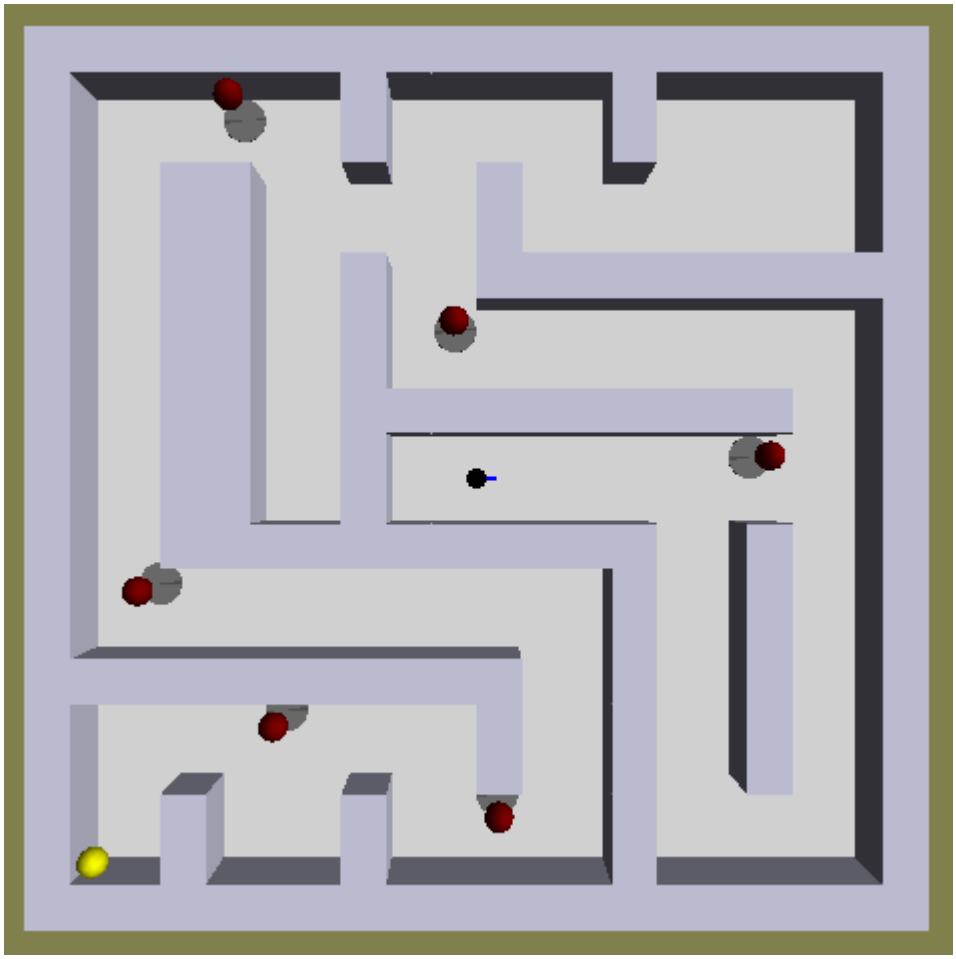

Figura 17: Escenario de entrenamiento con el doble de zonas grises.

Sin embargo, el resultado tampoco fue satisfactorio: tras 1000 generaciones, la fitness máxima alcanzó apenas 0.029, sin que el robot desarrollase una estrategia efectiva para esquivar las nuevas zonas.

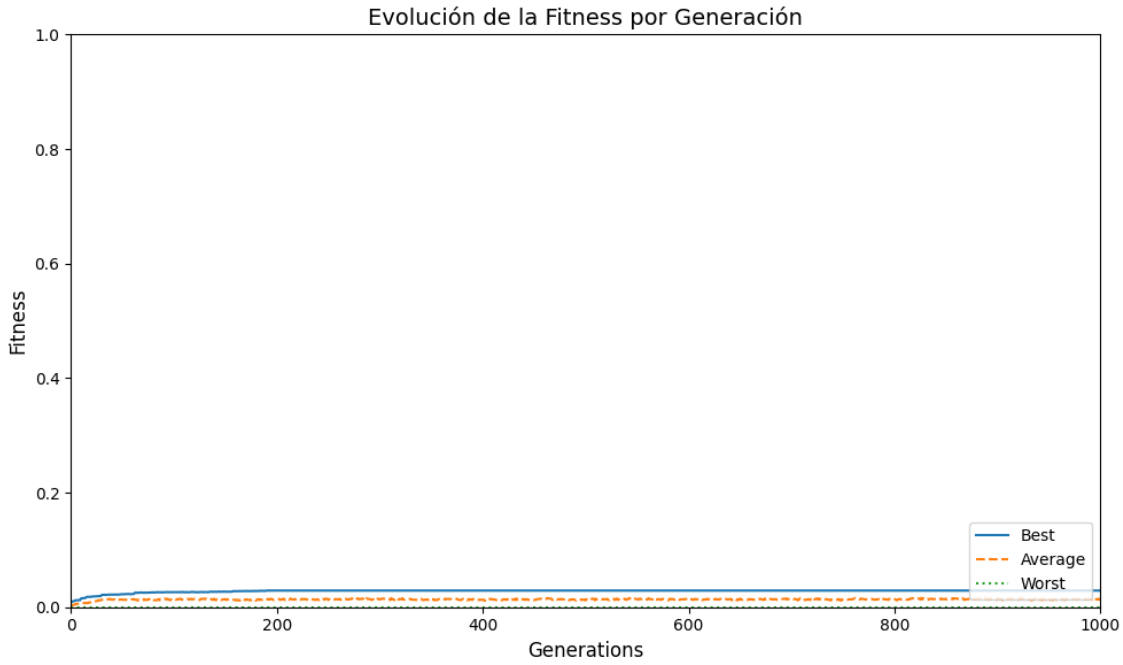

Figura 18: Evolución de la fitness con el doble de zonas grises.

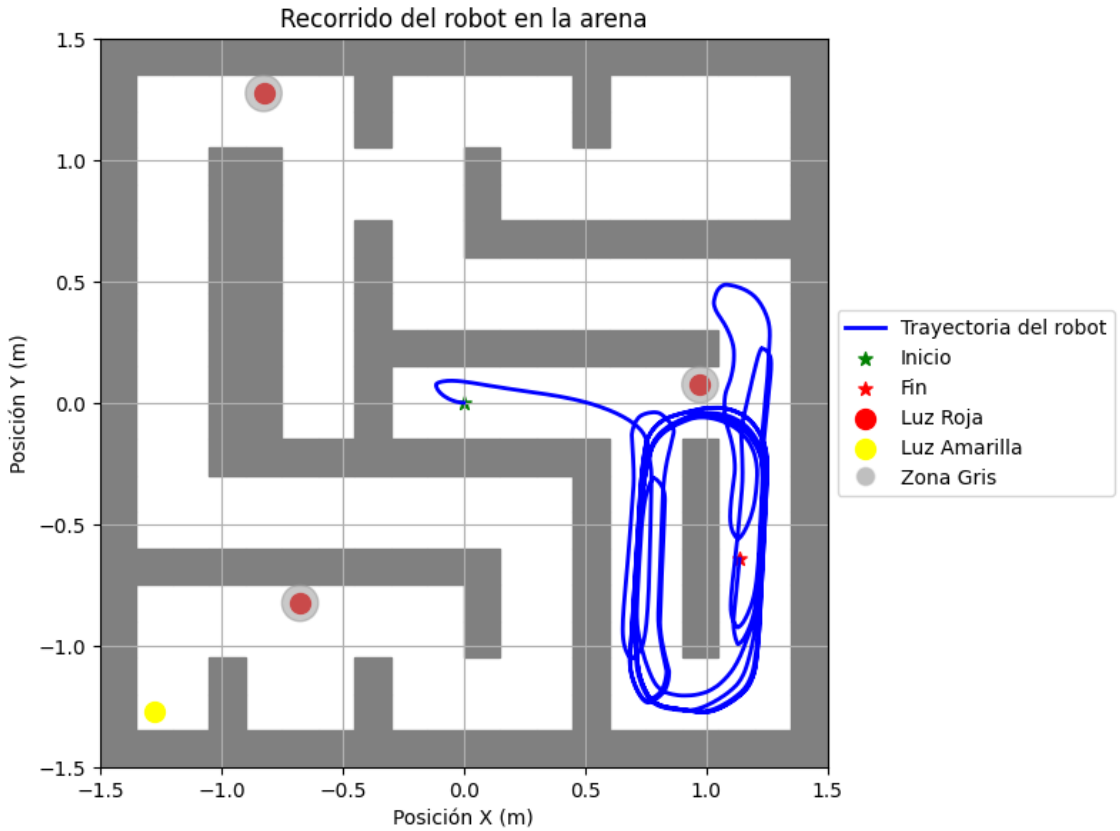

Figura 19: Trayectoria del mejor individuo con el doble de zonas grises.

La introducción de más obstáculos sin una redefinición de la función de fitness parece haber incrementado la complejidad del entorno sin proporcionar señales claras de aprendizaje, agravando el problema de underfitting.

CAMBIO DE LA FUNCIÓN DE FITNESS

Finalmente, se diseñó una nueva versión de la función de fitness incorporando un parámetro adicional denominado redSafe, que penaliza la proximidad excesiva a luces rojas. El objetivo era reforzar el aprendizaje de trayectorias que bordearan las zonas grises sin necesidad de pisarlas.

Esta es la nueva función de fitness:

$$Fitness = \sum_{i=0}^{Nsteps} \frac{\left((0{,}7\Delta Y + 0{,}2Fwd + 0{,}1wallFactor) * wallSafe * pRevisit * \text{forPro} * \text{redSafe}\right)}{Nsteps} * \text{factorColl}$$

El parámetro de añadimos es redSafe, vamos a desglosarlo como los demás parámetros.

- **Significado práctico:** Margen de seguridad ante las luces rojas, castiga si alguno de los sensores de luz roja ve el obstáculo demasiado cerca.
- **Qué premia o penaliza:** Cuanto más cerca de una luz roja, más se reduce la fitness.
- **Expresión expandida:**

$$\boldsymbol{redSafe = 1 - max\{redLight[i]\}\ \forall i \in [0, 7]}$$

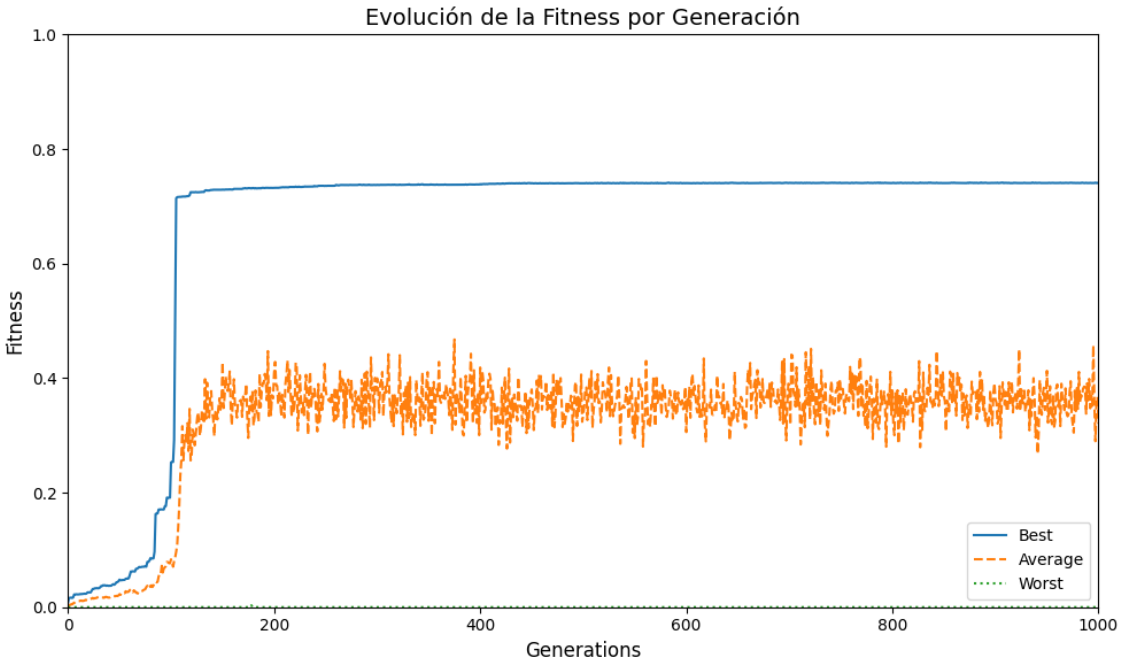

Figura 20. Evolución de la fitness con penalización por luz roja.

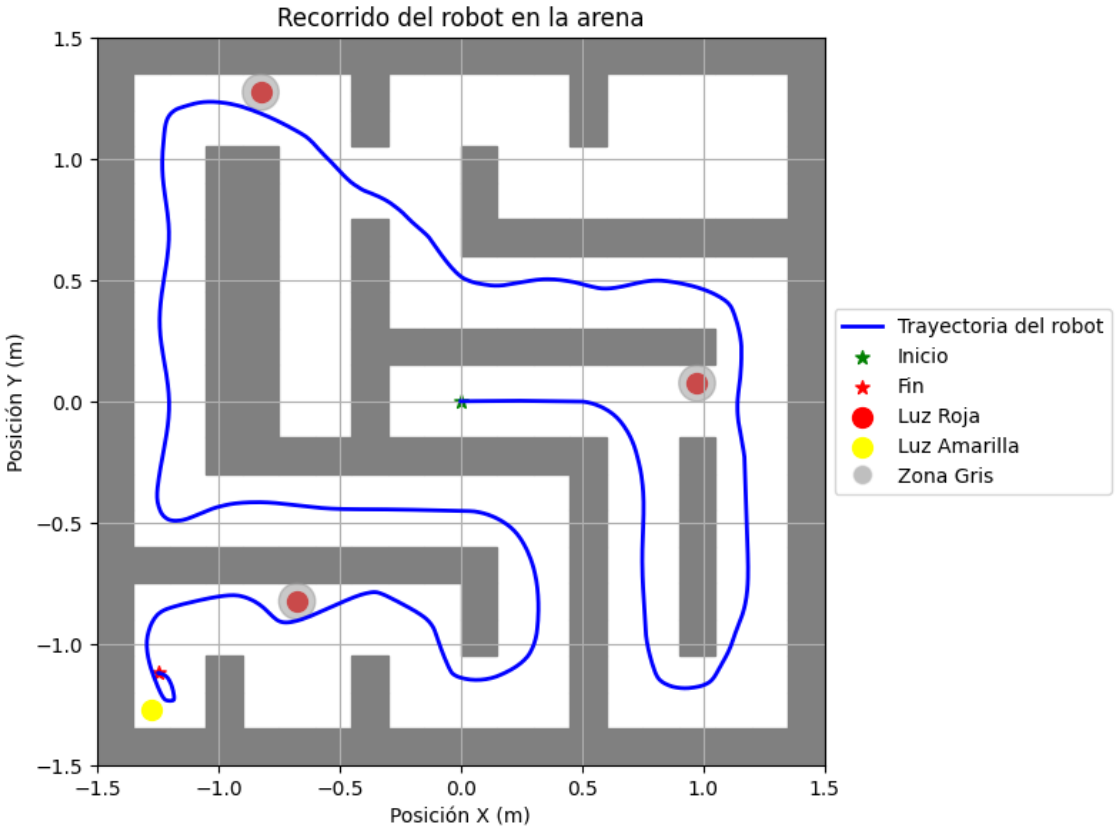

Figura 21: Trayectoria del mejor individuo con penalización por luz roja.

El robot logra completar el laberinto, y su comportamiento es más cauteloso, evitando recorridos que lo expongan a zonas críticas. Sin embargo, esta estrategia provoca efectos secundarios: en un entorno nuevo, el robot no consigue avanzar, bloqueándose al interpretar como intransitables zonas cercanas a luces rojas.

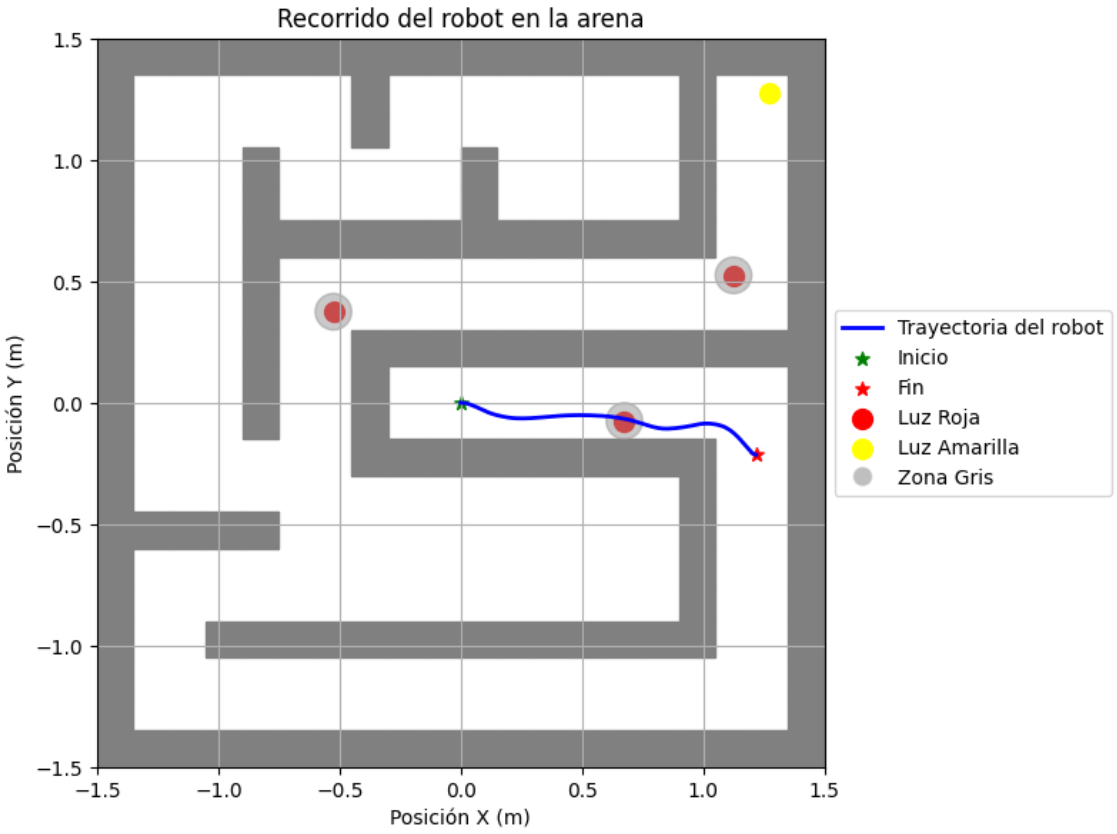

Figura 22: Trayectoria del mejor individuo con penalización por luz roja en escenario distinto al entrenado.

Este caso sugiere un sobreajuste inducido por la nueva penalización. Una posible mejora sería redefinir el término redSafe para que solo penalice si la luz roja está extremadamente próxima, evitando rechazar caminos válidos por un exceso de precaución.

PROPUESTA DE LA QUE POSIBLEMENTE SEA LA MEJOR MODIFICACIÓN

A lo largo del desarrollo de este proyecto, se ha detectado un patrón recurrente, el robot tiende a especializarse en el escenario sobre el que ha sido entrenado, mostrando dificultades para adaptarse a entornos nuevos. Esta falta de

generalización ha motivado la reflexión sobre una mejora estructural que, si bien no ha podido implementarse por complejidad programática y sobre todo falta de tiempo, se considera altamente prometedora.

La propuesta consiste en aprovechar el parámetro del simulador SAMPLES PER CHROMOSOME, que permite evaluar un mismo cromosoma sobre distintas muestras. A nivel práctico, esto implicaría que cada individuo de la población evolutiva sea probado en varios laberintos diferentes, con diferentes distribuciones de muros, zonas grises y luces rojas, pero manteniendo reglas comunes de estructura.

Esta técnica tendría múltiples ventajas:

- Reducción del overfitting al escenario específico.
- Estimulación del aprendizaje de estrategias generales de navegación (e.g., seguir muros, esquivar luces rojas, avanzar hacia la meta) en lugar de memorizar trayectorias fijas.

La implementación requeriría:

- Generar múltiples mapas compatibles con las reglas definidas.
- Modificar el experimento para que se utilice uno de estos mapas para cada muestra.
- Adaptar la función de fitness para evitar contradicciones entre escenarios.

Aunque este enfoque no ha sido materializado en este trabajo, se entrega como una línea de desarrollo prioritaria para futuras versiones del sistema. Por su capacidad para atacar directamente el problema más crítico detectado, se considera la modificación más prometedora.

RED NEURONAL RECURSIVA CTRNN

Como parte de la extensión del experimento a una arquitectura dinámica, se implementó una red neuronal recursiva (CTRNN) basada en la misma configuración estructural que la red ANN utilizada inicialmente. La única modificación introducida fue de naturaleza recursiva, al conectar entre sí las dos neuronas motoras.

Esta conexión recursiva permite a cada neurona motora influir sobre su propio estado y el de su compañera en el tiempo, dotando al sistema de una forma básica de memoria temporal o persistencia de activación. Esta característica es

clave para el modelado de comportamientos adaptativos más complejos, como la navegación en entornos ambiguos o con secuencias de decisiones.

A nivel genético, esta modificación supuso un aumento de 4 unidades en la longitud del cromosoma, pasando de 52 a 56 genes, para codificar los pesos recursivos adicionales.

Aunque toda la arquitectura fue correctamente codificada y es funcional desde el punto de vista de simulación, no fue posible realizar una evolución completa con un número de generaciones suficientes como para extraer conclusiones significativas. Esta limitación se debió a la falta de tiempo.

A pesar de ello, se hace entrega del código completamente adaptado a esta nueva arquitectura, incluyendo los archivos modificados necesarios para su compilación y evaluación:

- ctrnnexp.cpp, ctrnnexp.h
- ctrnndistributedcontroller.cpp, ctrnndistributedcontroller.h
- paramFileNeuronMazeExp.txt

ARCHIVOS MODIFICADOS

1. nndistributedcontroller.cpp
2. nndistributedcontroller.h
3. paramFileNeuronMazeExp.txt
4. testneuronexp.cpp
5. testneuronexp.h
6. irifitnessfunction.cpp
7. irifitnessfunction.h
8. redlightobject.cpp
9. encodersersor.h
10. encodersensor.cpp
11. main.cpp
12. ctrnnexp.cpp
13. ctrnnexp.h
14. ctrnndistributedcontroller.h
15. ctrnndistributedcontroller.cpp

BIBLIOGRAFÍA

Material de Introducción a la Robótica Inteligente - https://www.robolabo.etsit.upm.es/subjects.php?subj=irin&tab=tab1