

UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA MECANICA



**GENERACIÓN DE COMPORTAMIENTOS NO
REACTIVOS EN CONTROLADORES ROBÓTICOS
EMPLEANDO ALGORITMOS EVOLUTIVOS**

INFORME DE SUFICIENCIA

**PARA OPTAR EL TITULO PROFESIONAL DE:
INGENIERO MECATRÓNICO**

JIM MARTIN CATACORAS OCAÑA

PROMOCION 2009 - II

LIMA-PERU

2012

INDICE

PROLOGO

CAPITULO 1

INTRODUCCION	3
1.1. Antecedentes	5
1.2. Objetivo General	9
1.3. Justificación	9
1.4. Descripción del Proyecto	10

CAPITULO 2

FUNDAMENTOS DE ALGORITMOS EVOLUTIVOS	11
2.1. Definición de Algoritmos Evolutivos	11
2.2. Teoría de la Evolución Biológica	12
2.3. Algoritmo Evolutivo Básico	13
2.4. Algoritmos Genéticos	15
2.5. Representación Genética o Encoding	16
2.5.1. Representación Directa	17
2.5.2. Representación Indirecta	18
2.5.3. Representación Implícita	19
2.6. Población.....	20
2.7. Función Fitness o de Evaluación	21
2.8. Selección de Soluciones	22
2.8.1. Asignación de Fitness	22
2.8.2. Estrategias de Selección de Progenitores	25
2.8.3. Restricciones de Apareamiento	28
2.9. Reproducción	30

2.9.1. Recombinación o Crossover	30
2.9.2. Mutación	33
2.10. Modelos Evolutivos	35
2.11. Reinscripción o Reemplazo	35
2.12. Terminación o Finalización de la Búsqueda	36
2.13. Clasificación de los Algoritmos Genéticos	38
2.14. Software destinado a Computación Evolutiva	39
2.15. Software para la Simulación de Sistemas Robóticos	40
2.16. Transferencia desde la Simulación hacia la Realidad	40
CAPITULO 3	
ALGORITMOS GENETICOS Y REDES NEURONALES	43
3.1. Neuroevolución	43
3.2. Redes Neuronales Artificiales	44
3.3. Redes Perceptron	45
3.4. Redes de Función de Base Radial (RBFs)	46
3.5. Redes Recurrentes Completamente Conectadas (FRNNs)	47
3.6. Redes Echo State Network (ESNs)	47
3.7. Redes Recurrentes de Tiempo Continuo (CTRNNs)	48
3.8. Redes Neuronales de Impulsos (SNNs)	49
CAPITULO 4	
GENERACION DE COMPORTAMIENTOS NO-REACTIVOS	51
4.1. Definición de un Comportamiento No-Reactivivo	51
4.2. Planteamiento del Problema	52
4.3. Características del Hardware y Software	53
4.4. Características de la Simulación	57
4.5. Metodología Experimental	65

4.6. Experiencia #1: Evaluación de Funciones Fitness	65
4.6.1. Parámetros Experimentales	66
4.6.2. Resultados	69
4.6.3. Análisis de Resultados	74
4.7. Experiencia #2: Evolución Morfológica	81
4.7.1. Parámetros Experimentales	81
4.7.2. Resultados	83
4.7.3. Análisis de Resultados	87
4.8. Experiencia #3: Evaluación de Redes Neuronales	90
4.8.1. Parámetros Experimentales	90
4.8.2. Resultados	97
4.8.3. Análisis de Resultados	99
4.9. Validación de los Resultados	109
4.9.1. Simulación del ambiente virtual y agente robótico	109
4.9.2. Simulación de los actuadores	115
4.9.3. Simulación de los sensores de distancia	116
4.9.4. Simulación de los sensores de color	120
4.9.5. Simulación del Neurocontrolador	121
4.9.6. Simulación de colisiones	122
4.9.7. Desarrollo y Resultados de la simulación	123
CONCLUSIONES Y RECOMENDACIONES	126
BIBLIOGRAFIA	129
ANEXO A Datasheet del sensor TCND5000	146
ANEXO B Datasheet del sensor CNY70	153
ANEXO C Nota de Aplicación - Sensores Reflexivos Vishay Semiconductors ..	158
ANEXO D Datasheet del servomotor Parallax 900-00008	166

ANEXO E	Algoritmo Nondominated Sorting Genetic Algorithm-II (NSGA-II)	173
ANEXO F	Desarrollo Algorítmico – Operadores Evolutivos	177
ANEXO G	Desarrollo Algorítmico – Estructuras Neuronales	189
ANEXO H	Desarrollo Algorítmico – Simulación	192
ANEXO I	Desarrollo Algorítmico – Exp#1: Funciones Fitness	208
ANEXO J	Desarrollo Algorítmico – Exp#2: Evolución Morfológica	238
ANEXO K	Desarrollo Algorítmico – Exp#3: Comportamiento No-Reactiva	246

PROLOGO

El tema de este informe tiene como objetivo diseñar controladores robóticos que evidencien comportamientos no-reactivos empleando métodos de evolución artificial. Actualmente las instituciones educativas en el ámbito mundial desarrollan numerosos proyectos en el área de Robótica Evolutiva, en nuestro país se han elaborado escasos trabajos en dicho dominio, por tanto este informe busca aportar a la literatura existente y aumentar la discusión respecto a este campo, asimismo este trabajo pretende proveer una base para investigaciones futuras de mayor envergadura.

El **PRIMER CAPITULO** proporciona una breve introducción, contiene también información relativa a los capítulos posteriores presentes en este trabajo y además elabora una exposición sobre los trabajos previos, concernientes al campo de la Robótica Evolutiva, que motivaron la realización de este trabajo.

El **SEGUNDO CAPITULO** desarrolla una definición básica sobre los Algoritmos Evolutivos, asimismo expone acerca de las diferentes y diversas terminologías y operadores subyacentes en cualquier Algoritmo Evolutivo, especialmente en los Algoritmos Genéticos, finalmente detalla sobre las variadas herramientas de software destinadas a aplicaciones de Robótica Evolutiva y las técnicas utilizadas para la transferencia de la solución virtual al mundo real.

El **TERCER CAPITULO** trata en una sección inicial sobre la interacción entre los Algoritmos Evolutivos y Redes Neuronales, como segundo punto se explican con cierto detalle las arquitecturas y tipos neuronales que serán empleadas durante el desarrollo de este trabajo.

El **CUARTO CAPITULO** describe el problema de estudio en torno al cual se va a desarrollar el informe, adicionalmente se brinda información relevante sobre las características del ambiente en donde se llevará a cabo la experiencia y las especificaciones técnicas del agente robótico que van a ser tomadas en cuenta durante la resolución del problema. A continuación se detallan las metodologías y los parámetros utilizados en la búsqueda de la solución del problema, finalmente se revelan los resultados obtenidos y se realiza un análisis comparativo entre las diversas alternativas empleadas con el objetivo de determinar su idoneidad.

CAPITULO 1

INTRODUCCION

En los últimos tiempos la Robótica moderna se ha trasladado desde el ambiente ciertamente controlado de la industria hacia el ambiente caótico de los seres humanos, ejerciendo en estas primeras intervenciones, labores relativamente simples en las áreas de servicio y entretenimiento, en un futuro sólo podemos prever mayor interacción entre humanos y robots. Nuestra imaginación nos incita a crear robots cada vez más avanzados, que puedan resolver los mismos problemas complejos que enfrentamos a diario; adaptarse y reaccionar apropiadamente ante situaciones inesperadas y apremiantes o simplemente ser capaces de comunicarse con los seres humanos con un grado de abstracción similar al nuestro. El ambiente humano y en general el ambiente natural en que vivimos es típicamente no estructurado y cambia constantemente, resulta imposible incluso para nosotros considerar e interpretar las implicancias de cada interacción con el entorno; por ello a medida que los robots son confrontados ante situaciones más reales, la noción de diseñarlos basados en observaciones y suposiciones inferidas por nosotros mismos se manifiesta cada vez más obsoleta. Es entonces que desde hace 15 años atrás, los ingenieros en Robótica decidieron inspirarse en la biología; para entender el cambio de perspectiva basta con observar las habilidades de los seres humanos, sin embargo aún organismos de mayor simplicidad como aves, hormigas, incluso las células de nuestro sistema inmunológico, exhiben comportamientos deseables de ser replicados en nuevas generaciones de robots.

Los investigadores notaron que dichas aptitudes son intrínsecas de cada especie y dependen íntegramente del ambiente físico en el que se desenvuelven, además estas características no se adquieren simplemente a través del aprendizaje de un individuo dentro de un corto plazo, sino por el contrario es necesario concebir dicho surgimiento gracias a un prolongado desarrollo evolutivo durante el transcurso de múltiples generaciones, al cabo del cual Emergerán sujetos perfectamente adaptados a las complejidades de su entorno. Es a partir de esta idea elemental que se estableció el concepto de Robótica Evolutiva con el fin de evolucionar robots con mayor versatilidad, autonomía, adaptabilidad y robustez, capaces de desenvolverse con eficiencia excepcional en el confuso mundo real. Hasta la fecha esta rama de la Robótica ha presenciado un progreso alentador, que ha permitido avances en aplicaciones como locomoción de robots multicuerpo y humanoides, navegación autónoma de robots móviles y aéreos, localización global de robots, interacción humano-robot, comunidades o enjambres de robots, robots modulares, microrobots, entre otras. El futuro inmediato de la Robótica Evolutiva promete evolución de robots en tiempo real, hardware evolucionable, comunicación bidireccional avanzada entre robots, autodetección y asimilación de fallas, entre otras tendencias, resulta aún más importante recordar que todavía no hemos llegado al final del camino y los investigadores de todas las ramas que abarca la inteligencia artificial continuarán experimentando en pos del progreso.

1.1. Antecedentes

La primera alusión sobre la posibilidad de diseñar criaturas artificiales mediante un proceso evolutivo fue evocada en 1984 por el neurofisiólogo Valentino Braitenberg; en su libro “Vehicles, Experiments in Synthetic Psychology” propuso un experimento mental, donde un número de robots son ubicados sobre una mesa, estos robots desplegarán distintos comportamientos y eventualmente alguno caerá al piso, ahora con ánimos de mantener una población constante, tomamos como referencia algún robot que todavía permanezca encima de la mesa y construimos otro similar, el cual agregamos al grupo existente, obviamente en el proceso de replicación siempre ocurrirán errores; conexiones inexactas, componentes distintos, etc., por tanto el nuevo robot no exhibirá necesariamente el mismo accionar que su predecesor, si por causas fortuitas alcanzase “sobrevivir” mayor tiempo que otros, entonces producirá más descendientes que aquellos. De esta manera se genera una constante creación de nuevos diseños y mejoras en los mismos sin necesidad del esfuerzo de un diseñador consciente, comparable al proceso de evolución de la vida en la Tierra [1].

Entre los años 1992 y 1994 dos equipos de investigación, uno del Instituto Federal Suizo de Tecnología en Lausanne (EPFL) liderado por Floreano y Mondada [2] y otro de la Universidad de Sussex en Brighton encabezado por Harvey, Husbands y Cliff [3], reportaron los primeros casos exitosos donde robots evolucionaron artificialmente con mínima intervención humana, desarrollando circuitos neuronales que les permitían moverse autónomamente dentro de su ambiente.

En los experimentos anteriores la evolución transcurrió completamente en el mundo físico, demorándose hasta días en alcanzar los controladores deseados; en 1994 Miglino, Nafasi y Taylor fueron los primeros en utilizar simulaciones computarizadas donde robots virtuales evolucionaban hasta alcanzar controladores óptimos, que luego se transferían a un robot real [4].

A partir de entonces, el uso de controladores evolucionados artificialmente se extendió a múltiples aplicaciones; Lewis (1994) consiguió evolucionar un controlador para la locomoción de un robot hexápedo, utilizó redes neurales de tiempo continuo y un proceso de evolución por Etapas ejecutado en el robot real [5]; Sims (1994) demostró la coevolución de mente y cuerpo en criaturas simuladas bajo un ambiente virtual, sus morfologías consistían de bloques 3D, mientras sus mentes se basaban en neurocontroladores, la evolución premiaba a quienes se movilizasen exitosamente en el paisaje virtual, las criaturas surgidas presentaban comportamientos similares a caminar, saltar, reptar o incluso nadar [6]; Thompson (1995) inicio la investigación sobre Hardware Evolucionable (EHW), creó el primer controlador evolucionado en Tiempo Real u On-Line, el Algoritmo Evolutivo fue implementado en un chip de memoria RAM utilizando una Máquina de Estado Dinámica (DSM), método desarrollado por Thompson que permitía un flujo asíncrono de señales [7]; Watson, Ficici y Pollack (1999) realizaron los primeros experimentos relativos a evolución de comportamientos grupales, emplearon una comunidad de 8 robots “Cricket”, cuya misión era acercarse en conjunto a una fuente luminosa ejecutando principalmente fotoaxis, para ello usaron controladores homogéneos y desarrollaron el llamado Algoritmo de Transferencia Genética Probabilística (PGTA) (una adaptación del modelo evolutivo de Estado Estacionario) [8];

Husbands y Reil (2002) utilizaron un Algoritmo Genético Simple (SGA) para evolucionar una red recurrente de tiempo continuo, capaz de reproducir la locomoción bípeda, los controladores finales lograron caminatas sobre superficies planas, basándose sólo en la secuencia rítmica de movimientos sin ayuda sensorial [9]; Floreano, Marocco, Kato y Sauser (2004) investigaron la coevolución de Visión Activa y Selección de Características, en simulación alcanzaron controladores que manipulaban la dirección, aceleración y freno de un vehículo, quien atravesaba circuitos automovilísticos virtuales; en una experiencia real neurocontroladores evolucionados artificialmente evitaban que un robot, equipado con una cámara motorizada, colisionase con las paredes de una arena aún expuesto a estímulos externos variables (como personas moviéndose alrededor) [10]; Barlow y Oh (2004) desarrollaron controladores para vehículos aéreos no tripulados, usaron Programación Genética y optimización multiobjetivo con la finalidad de generar tres comportamientos interdependientes: detectar una fuente de energía electromagnética, navegar eficientemente hacia la misma y por último circular cercanamente alrededor del emisor [11].

En trabajos recientes Tellez, Angulo y Pardo (2006) implementaron un neurocontrolador que efectuaba la locomoción en un robot cuadrúpedo Sony AIBO [12]; Eaton y Davitt (2007) generaron controladores para la locomoción de robots humanoides Sony QRIO, consiguiendo comportamientos como patinar y cojear [13]; Ijspeert, Crespi, Ryczko y Cabalguen (2007) diseñaron un controlador para la locomoción anfibia de una salamandra artificial capaz de caminar y nadar [14]; Tohge e Iba (2007) evolucionaron ROBOCUBE, robot modular formado de cubos interconectados capaz de alterar su morfología [15];

en el Laboratorio de Sistemas Inteligentes de la EPFL (2008) se desarrollaron los S-Bots, conjunto de agentes móviles que reproducen un enjambre robótico, en un experimento final lograron localizar un objeto específico y trasladarlo en grupo hacia su base inicial atravesando un terreno peligroso [16],[17]; Suzuki, Gritti, Floreano (2009) consiguieron la evolución de coordinación visomotriz durante la caminata del robot bípedo HOAP-2 [18]; Ruini y Cangelosi (2009) experimentaron con grupos de vehículos aéreos no tripulados, su tarea integraba navegación autónoma en ambientes desconocidos, localización y neutralización de objetivos [19]; Peniak, Marocco y Cangelosi (2009) generaron un controlador robótico destinado para la exploración de Marte, el cual ejecuta navegación autónoma [20].

Entre los proyectos en desarrollo actualmente contemplamos Swarmanoid, constituido por robots autónomos, heterogéneos y dinámicamente conectados, su objetivo yace en el diseño, implementación y control de sistemas robóticos distribuidos, siendo la meta ulterior un enjambre de robots humanoides [21]; SYMBRION persigue 3 objetivos: aplicar los principios de evolución artificial al software, hardware y funcionalidad de organismos artificiales; desarrollar bajo inspiración en biología y tecnología estrategias de adaptación, autoreparación, supervivencia, autoconfiguración aplicadas a organismos robóticos simbióticos; por último desarrollar sistemas robóticos de gran escala capaces de adaptarse a ambientes abiertos y de alta dinámica [22], REPLICATOR consistiría de un conjunto de microrobots autónomos con la habilidad de autoensamblarse para crear organismos artificiales más grandes, estos serían en extremo adaptables, robustos, escalables y profusos en capacidades sensoriales y motoras, la meta final son redes sensoriales autónomas, inteligentes y reconfigurables [23].

1.2. Objetivo General

Generar comportamientos No-Reactivos en Controladores Robóticos, mediante el uso de métodos y técnicas basadas en Algoritmos Evolutivos utilizados en la actualidad.

1.3. Justificación

En la actualidad, la noción de diseñar sistemas Robóticos utilizando solamente nuestra propia experiencia y percepción encuentra cada vez mayores dificultades a medida que aumenta la complejidad de las aplicaciones concebidas; por esta razón resulta indispensable el uso de Algoritmos de Inteligencia Artificial que guíen la evolución o aprendizaje de agentes Robóticos bajo escasa supervisión humana. En el caso de comportamientos No-Reactivos (conocidos también como No-Determinísticos o de Respuesta Temporal Retardada), estos son indispensables para un robot que se enfrente a situaciones en el mundo real, los seres Humanos y otros animales poseemos esta habilidad, y la expresamos cada vez que utilizamos nuestra memoria para obtener algún beneficio o evitar algún peligro, además representa una característica de vital importancia para una posterior consecución de comportamientos de mayor complejidad. Desde una perspectiva personal este Trabajo expresa el interés del Autor en los campos de Inteligencia Artificial y Robótica; los problemas analizados en este trabajo y la metodología de su resolución deben ser valorados como pasos básicos e iniciales hacia posteriores proyectos de mayor profundidad e innovación científica.

1.4. Descripción del Proyecto

Este Informe comprende el estudio de un problema general encontrado en la Robótica Evolutiva, este trata acerca de la generación de comportamientos No-Reactivos o No-Determinísticos en sistemas robóticos, dichas conductas son exhibidas en robots con habilidades memorísticas, es decir aquellos que toman decisiones basadas no sólo en la información sensorial actual sino también asistidos por el recuerdo de eventos pasados. La metodología de este estudio comprende tres partes diferenciadas, la primera consta de la experimentación con múltiples Funciones Fitness o de Desempeño, a fin de definir cual reporta mejores resultados en cuanto a la obtención de estrategias que permitan a un agente robótico movilizarse adecuadamente dentro de su ambiente, dichas pruebas buscan además exponer los problemas intrínsecos que existen al momento de especificar una función Fitness. La segunda sección consiste en la implementación de una evolución morfológica, con la finalidad de descubrir el posicionamiento más óptimo de los sensores de distancia del robot. Finalmente la tercera parte trata sobre la evolución propiamente dicha de comportamientos no-reactivos en el autómata, para ello a manera de estudio se probarán múltiples clases neuronales. En todas las experiencias se empleará un Algoritmo Genético Multi-Objetivo (MOGA) NSGA-II. Es importante mencionar que la totalidad del estudio se realizó en simulaciones computarizadas y que en el alcance de este proyecto no se contempló una etapa de implementación en un robot real, sin embargo sí se tomaron ciertas consideraciones respecto de una transferencia a futuro del resultado obtenido hacia el mundo real.

CAPITULO 2

FUNDAMENTOS DE ALGORITMOS EVOLUTIVOS

2.1. Definición de Algoritmos Evolutivos

Los Algoritmos Evolutivos (EA) son técnicas utilizadas dentro del ámbito de la Computación Evolutiva, se definen como algoritmos de optimización metaheurística basados en poblaciones genéricas, utilizados para aproximar soluciones en todo tipo de problemas, puesto que no toman consideraciones previas sobre las características intrínsecas del ambiente en que se desarrolla la experiencia [24],[25].

El campo de los Algoritmos Evolutivos reúne diversas ramas de investigación creadas y desarrolladas de manera independiente. Los cuatro paradigmas de mayor influencia son las Estrategias evolutivas (ES) introducidas por Ingo Rechenberg y Hans-Paul Schwefel, Algoritmos Genéticos (GA) establecidos por John Holland, Programación Evolutiva (EP) propuesta por Lawrence J. Fogel, Alvin J. Owens y Michael J. Walsh y Programación Genética (GP) desarrollada por Nichael Cramer y John. R. Koza. Sin embargo, todos estos paradigmas comparten similaridades como la inspiración en la Biología Evolutiva, de la cual adquieren ideas como la Selección Natural, Recombinación y Mutación genética y sólo difieren entre ellos en las representaciones genéticas que utilizan [26],[27].

2.2. Teoría de la Evolución Biológica

La idea de la evolución ha existido desde épocas remotas, notablemente entre los helénicos como Epicuro, pero la teoría moderna no se estableció hasta la llegada de los siglos XVIII y XIX, con la contribución de científicos como Christian Pander, Jean-Baptiste Lamarck y Charles Darwin.

Jean-Baptiste Lamarck formuló la primera teoría de la evolución. En vista de la gran variedad de organismos, propuso que estos habían evolucionado desde formas simples; postulando que los protagonistas de esa evolución habían sido los propios organismos por su capacidad de adaptarse al ambiente; los cambios en ese ambiente generaban nuevas necesidades en los organismos y esas nuevas necesidades conllevarían a una modificación de los seres que sería heredable [28]. Luego Charles Darwin introdujo el concepto de selección natural como un proceso por el cual las mutaciones que mejoran la capacidad reproductiva perduran y se vuelven cada vez más frecuentes en las sucesivas generaciones de una población, pero Darwin no pudo explicar este mecanismo; fue Gregor Mendel quien después describió los principios de la herencia genética [29],[30].

Actualmente la Síntesis Evolutiva Moderna es la teoría de mayor aceptación científica, define la evolución como un cambio en la composición genética de las poblaciones, donde las unidades de la evolución son las poblaciones de organismos y no los tipos, además la variabilidad genotípica y genética en las poblaciones se produce por recombinación genética como resultado de la reproducción sexual y mutaciones aleatorias.

Esta teoría refuerza el rol de la selección natural y la considera como la fuerza más importante que modela el curso de la evolución fenotípica, postula además que en ambientes cambiantes, la selección direccional es de especial importancia; así como en poblaciones pequeñas la deriva genética aleatoria (o sea, pérdida de genes del pozo genético) puede ser significativa. Según otro principio importante, la Teoría Sintética nos asegura que las transiciones evolutivas en las poblaciones suelen ser graduales, es decir, nuevas especies evolucionan a partir de las variedades preexistentes a través de procesos lentos y en cada etapa se mantiene su adaptación específica [31].

2.3. Algoritmo Evolutivo Básico

Asumamos que nos encontramos con un problema de optimización. Un dominio de soluciones posibles (espacio de búsqueda) es definido, a cada una se le puede asignar un valor de desempeño (Fitness) reflejando la calidad de la misma. El objetivo es encontrar una solución de muy alta calidad.

En el contexto de los Algoritmos Evolutivos el término “individuo” es usado para referirse a una solución. Un individuo se encuentra representado por su “genotipo”, el cual dentro de un algoritmo se puede definir como la representación matemática de la solución al problema, por ejemplo una matriz de valores binarios con dimensiones fijas. El código genético expresado en el genotipo está sujeto a operadores genéticos durante el proceso de reproducción, pero el genotipo en sí mismo es inmutable, es decir, no se producen cambios en el transcurso del ciclo de vida de un individuo.

Mientras el genotipo contiene la información para construir un organismo, el fenotipo es la expresión de las propiedades codificadas en el genotipo, es decir, las características que exhibe el individuo como solución a un problema. Dos individuos pueden tener el mismo genotipo pero actuar de manera distinta porque el comportamiento de cada uno es alterado por condiciones ambientales. Sólo el fenotipo está sujeto al proceso de selección [32],[33].

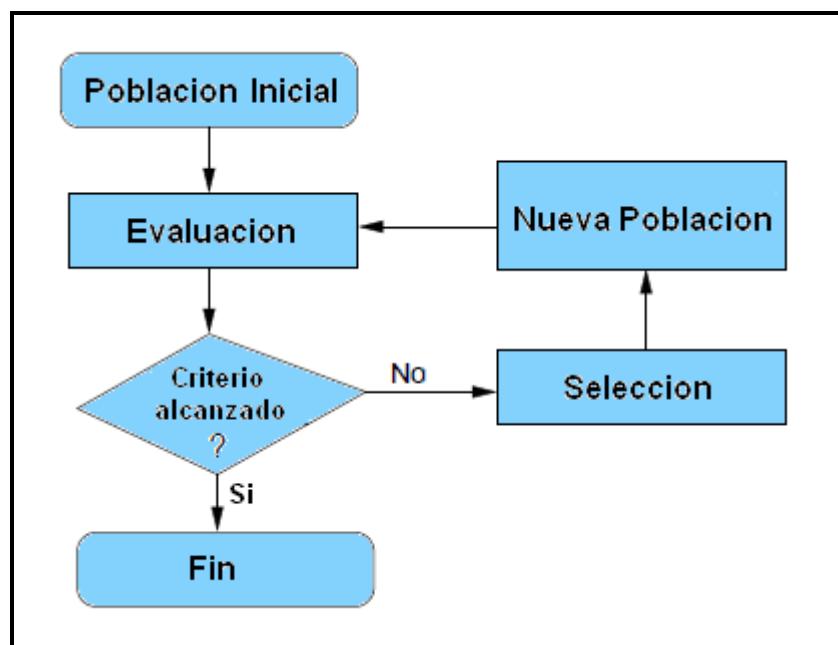


FIGURA 2.1. ALGORITMO EVOLUTIVO BASICO

La figura 2.1 ilustra el Algoritmo Evolutivo Básico. Los EAs investigan diferentes caminos de búsqueda en simultáneo. Por esta razón una población compuesta de diversos individuos es mantenida constantemente. Usualmente una población de individuos inicializados aleatoriamente sirve como punto de partida. Pero individuos de generaciones anteriores o con conocimiento previo sobre el problema particular también pueden ser sembrados para construir una población inicial.

La calidad de cada solución es determinada por un proceso de evaluación.

Para este fin se utiliza una función de desempeño o Fitness, simulaciones complejas o experimentos prácticos podrían utilizarse para obtener una función Fitness que refleje la calidad de cada solución. En caso no se encuentre una solución de suficiente calidad, entonces algunos individuos son seleccionados para producir la siguiente generación, siendo generalmente las soluciones de mayor desempeño (fitness) escogidas. El material genético de los individuos seleccionados se modificará mediante operadores genéticos (Recombinación o Mutación) con el objetivo de producir descendencia. La Mutación modifica el genotipo de un solo individuo, mientras que la Recombinación fusiona dos o más genotipos, incluso algunos individuos se replican a sí mismos sin sufrir ninguna variación, esta metodología es llamada Elitismo [34].

2.4. Algoritmos Genéticos

La idea básica de los Algoritmos Genéticos sigue que la diversidad genética de una población cualquiera contiene potencialmente la mejor solución para un problema cualquiera. Sin embargo esta solución no siempre se encuentra “activa”, debido a que la combinación genética que la representa está dividida entre varios sujetos, y sólo la asociación de los diferentes genomas conducirá hacia su descubrimiento. A partir de este concepto los GAs han sido desarrollados como métodos estocásticos de búsqueda global, que operan sobre poblaciones aplicando conceptos provenientes de la evolución biológica con el fin de producir cada vez mejores aproximaciones a la solución [35],[36].

En la implementación de un GA se debe considerar una amplia gama de métodos y operadores. A continuación se presentan de forma breve conceptos indispensables para el subsiguiente desarrollo de este trabajo.

2.5. Representación Genética o Encoding

La representación genética puede codificar apariencia, comportamiento o cualidades físicas de los individuos. A la representación de un individuo se le denomina “cromosoma” o “genoma”, y cada parámetro individual constituyente del mismo es llamado “gen” [37].

Existen tres variantes principales respecto al Encoding: Directo donde los parámetros del problema son encriptados en el genoma, Encoding Indirecto el cual guarda información relativa a reglas en base a quienes se reconstruye el genoma y Encoding Implícito donde la activación y expresión de un gen resulta de su interacción con el medio, dicha interacción modifica a su vez el ambiente y altera la activación y expresión de otros genes del cromosoma.

En caso los GAs sean empleados para optimizar Redes Neuronales o estructuras similares, la representación de éstas implica dos niveles independientes, uno es la codificación de los pesos sinápticos y demás parámetros numéricos, seguido por la representación topológica de la propia red neuronal; ambos casos son incorporados por igual en Representaciones Directas, Indirectas o Implícitas. Cualquier parámetro puede definirse mediante números binarios, reales, código Gray, código ASCII (American Standard Code for Information Interchange), clusters o algún otro sistema alfanumérico [38],[39].

2.5.1. Representación Directa

Utilizadas con éxito en Redes de escasas dimensiones, diversas metodologías se diferencian por su unidad básica (conexiones, nodos, capas o caminos). Una representación basada en conexiones requiere una arquitectura fija, en ella el genoma es una cadena de valores de pesos sinápticos, codificados dentro de un rango de acción (por ejemplo un binario de 8 bits); métodos más avanzados como Encoding Dinámico permiten una longitud variable, agregando así mayor adaptabilidad a este mapeo. Las representaciones basadas en nodos, capas o caminos relevan al usuario de tomar la decisión sobre la forma y número de neuronas que integran la arquitectura más óptima, ya que permiten la optimización topológica de la red neuronal [40]. Existen esquemas de mayor complejidad, Symbiotic Adaptive Neuro-Evolution (SANE) [41], Enforced Sub-Populations (ESP) [42] o Neuro-Evolution of Augmenting Topologies (NEAT) [43], este último método se muestra en la figura 2.2.

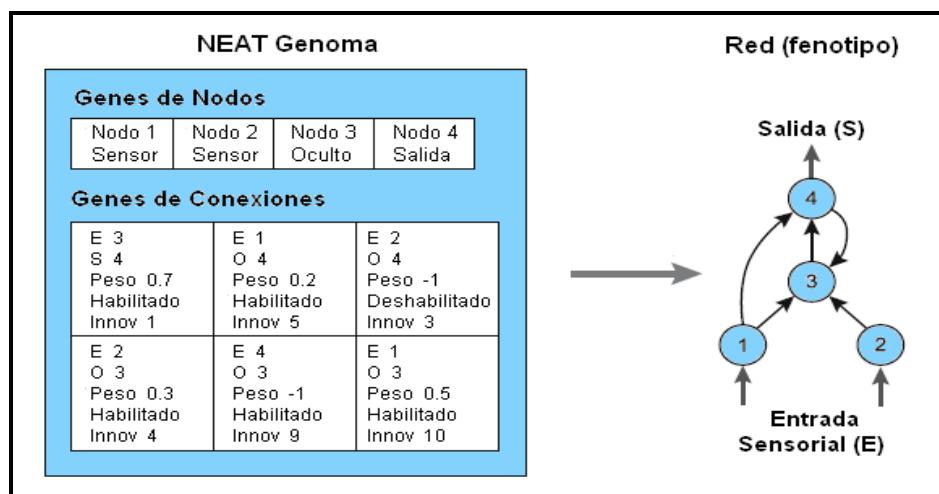


FIGURA 2.2. ENCODING DIRECTO BASADO EN NEAT

2.5.2. Representación Indirecta

Su uso ha sido motivado por la biología, los organismos no emplean Encoding Directo dado que ello resultaría ineficiente para producir largas redes de genes. Uno de los métodos más populares es la Representación Celular, el genoma codifica un grupo de instrucciones que luego son aplicadas a una red inicial.

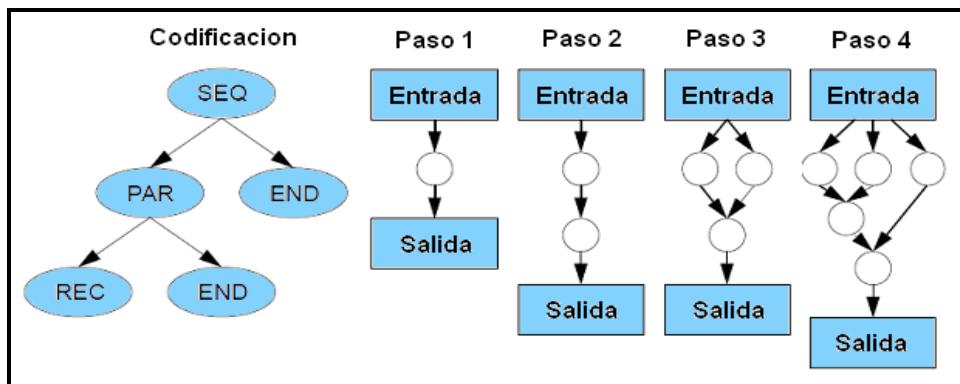


FIGURA 2.3. REPRESENTACION CELULAR

La figura 2.3 muestra el desarrollo celular de una red, inicia con un único nodo oculto, la instrucción SEQ divide el nodo primigenio en dos nodos secuenciales, cada rama que nace a partir de la instrucción SEQ, hace referencia a uno de los nodos recientemente creados; el subárbol del segundo nodo consiste sólo de la instrucción END, convirtiendo dicho nodo en uno Terminal, en cambio en el subárbol del primer nodo el comando PAR indica que se debe duplicar en paralelo dicho nodo, finalmente la instrucción REC se define como una llamada recursiva hasta el comando SEQ, por tanto repite el proceso previo de división secuencial (SEQ) y posterior duplicación de nodo (PAR) [44].

2.5.3. Representación Implícita

Inspirada en las Redes biológicas Reguladoras de Genes (GRN), donde las interacciones entre genes se instituyen a partir del medio químico en que se encuentra inmerso el genoma. La activación de un gen resulta de interacciones entre proteínas del ambiente y Secuencias Reguladoras del gen, además la expresión del gen corresponde a la síntesis de proteínas en la Secuencia Codificadora cuyo inicio y fin presentan marcadores denominados Secuencias Promotora y Terminadora. Asimismo las proteínas producidas por un gen interactúan con las Secuencias Reguladoras de otros e influencian su expresión [45]. Este proceso puede extenderse a Redes Neuronales Artificiales, un modelo exitoso es la Representación Genética Analógica (AGE) (figura 2.4), se basa en un genoma discreto compuesto por secuencias de caracteres ASCII, las distintas neuronas son codificadas en secuencias del genoma y separadas por delimitadores, análogo a las Secuencias biológicas, la interacción bioquímica se simula mediante una función llamada Mapa de Interacción, que recibe como argumento 2 secuencias y entrega el valor del peso sináptico entre neuronas [46].

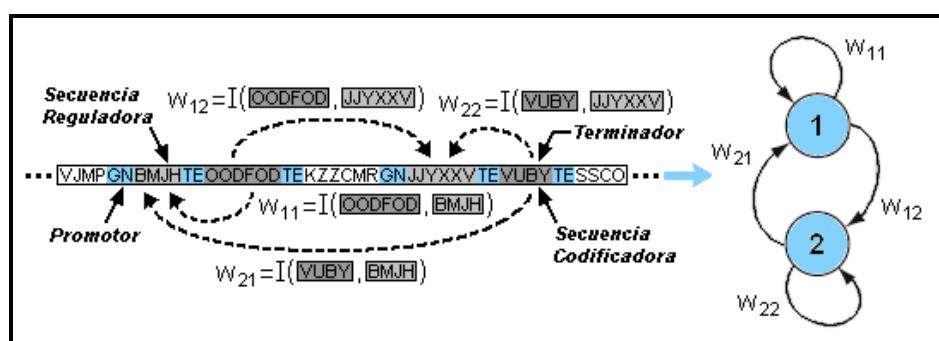


FIGURA 2.4. REPRESENTACION GENETICA ANALOGICA

2.6. Población

Una población se define como una colección de individuos (soluciones posibles). En general existen dos aspectos cruciales al especificar una población: Inicialización y Dimensión.

Idealmente la primera población debe presentar la mayor variabilidad genética posible, para alcanzar este objetivo es comúnmente utilizada una Inicialización Aleatoria. Para representaciones binarias se consigue mayor variabilidad creando la primera mitad de la población aleatoriamente y la segunda mitad negando lógicamente la mitad anterior. Otra técnica es la Inicialización Aleatoria Extendida donde se prueban múltiples inicializaciones y sólo se escoge a quien exhiba el mejor rendimiento. Sin embargo, en ciertos casos se consiguen buenos resultados mediante una Inicialización Heurística, aquí la población inicial es sembrada con individuos cuya aptitud óptima ya es conocida, de esta manera la población inicial ya muestra un desempeño elevado y puede ayudar al algoritmo a encontrar mejores soluciones rápidamente, pero también es probable que si la población inicial carece de diversidad genética el algoritmo solo explorará una pequeña porción del espacio de búsqueda y nunca localizará soluciones óptimas globales.

En lo referente a la dimensión de la población resulta muy obvio que cuanto mayor sea la población, más simple resultará explorar el espacio de búsqueda; mientras que los costos computacionales de memoria y tiempo se manifestarán más restrictivos. De esta manera la delimitación de dimensión poblacional depende de la capacidad computacional disponible [47],[48],[49].

2.7. Función Fitness o de Evaluación

Es un tipo particular de función objetivo que cuantifica la idoneidad de una solución dentro del contexto de los Algoritmos Evolutivos, para así poder clasificar a los individuos pertenecientes a una misma generación al compararlos unos frente a otros. Una función Fitness ideal se correlaciona estrechamente con la meta esperada del algoritmo, y al mismo tiempo permite ser calculada rápidamente. En general para un problema dado, no se encuentran mayores inconvenientes para determinar una función Fitness correspondiente, incluso, la mayoría de veces es definida implícitamente por el propio problema que desea optimizarse. Pero existen algunos casos donde es muy difícil o hasta imposible concebir siquiera una suposición de que función debería establecerse, ante esta situación los Algoritmos Genéticos Interactivos enfrentan dicha dificultad al tercerizar la evaluación de los individuos hacia agentes externos (normalmente seres humanos), este tipo de categorización también se conoce como Selección Estética [50].

Existen dos grandes clases de funciones Fitness, una donde la función no varía, permanece estática hasta encontrar la solución óptima, y otra dinámica donde la función muta constantemente. En cierta literatura se hace una distinción entre función objetivo y función Fitness, donde la función objetivo indica explícitamente el desempeño de una solución, mientras la función Fitness representa una transformación de la función objetivo, ya sea un escalamiento o incluso a través de una clasificación o ranking previo donde se le reasigna a cada individuo un valor Fitness basándose en funciones matemáticas o probabilísticas [51].

2.8. Selección de Soluciones

El proceso de selección se encarga de escoger a un grupo de individuos, a los que se les brinda la posibilidad de generar descendencia. Aquí intervienen dos etapas o pasos secuenciales, primero se debe asignar un valor Fitness o calificación a cada solución dentro de la generación actual, este valor cuantifica el desempeño mostrado por un individuo durante la simulación previa, luego se procede a seleccionar a cada uno de los padres necesarios para producir descendientes, en esta etapa también es común presentar criterios para restringir el emparejamiento entre padres [52].

2.8.1. Asignación de Fitness

En este punto, es importante considerar la dimensión de la salida obtenida a partir de una función Fitness, es decir es necesario distinguir entre un simple escalar o un vector, un escalar indica la optimización de un único parámetro, mientras un vector toma en cuenta como mínimo dos parámetros independientes. Para una función Fitness de forma escalar se emplean métodos como: asignación proporcional, asignación basada en rango y distribución de Fitness. Una asignación proporcional de Fitness consiste simplemente en normalizar los valores obtenidos de la función de evaluación (ecuación 2.1) [53].

$$\text{Fitness_Proporcional}(S_i) = \frac{\text{fitness}(S_i)}{\sum_j \text{fitness}(S_j)}, j = 1, 2, \dots, P \quad (2.1)$$

En una asignación de Fitness basada en rango, la población es ordenada primero de acuerdo al desempeño real obtenido durante la evaluación, a continuación a cada individuo se le asigna una Fitness que depende sólo de la posición que ocupa. Para este método han sido propuestas diversas fórmulas matemáticas y/o estadísticas, como la ecuación 2.2, donde el símbolo PS indica la presión de selección [54].

$$\text{Fitness}(\text{Posicion}) = \frac{\#individuos \cdot X^{\text{Posicion}-1}}{\sum_{i=1}^{\#individuos} X^{i-1}} \quad (2.2)$$

$$0 = (PS - \#ind) \cdot X^{\#ind-1} + PS \cdot X^{\#ind-2} + PS \cdot X + PS, X \rightarrow \text{raiz}$$

Asignación distribuida de Fitness, también conocida como Fitness Sharing o asignación basada en nichos, son técnicas donde el valor de Fitness de un individuo se obtiene al compararlo sólo con otros individuos similares a él, a este conjunto reducido de la población global se llama nicho. El método más conocido es la distribución explícita de Fitness, la Fitness de un individuo disminuye proporcionalmente a la cantidad de otros similares a él dentro de la población, la siguiente ecuación 2.3 ejemplifica este método:

$$\text{Fitness Distribuida}(i) = \frac{\text{fitness}(i)}{\sum_{j=1}^N sh(d_{ij})} \quad (2.3)$$

$$sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_s} \right)^\alpha, & \text{si } d < \sigma_s \\ 0, & \text{en otro caso} \end{cases}$$

Donde N es el número total de individuos, d_{ij} la distancia entre los individuos i y j, $sh(d_{ij})$ es la función de distribución, σ_s marca un umbral de disimilaridad, mientras que α es un parámetro que regula la forma de la función [55]. Otros métodos basados en nichos son: Niching secuencial, Especiación, Fitness Sharing implícito, Crowding y Clearing [56],[57],[58],[59].

En problemas con múltiples objetivos, las técnicas previas ya no son útiles, es necesario concebir nuevas formas de categorizar las soluciones dentro de una población. Para ello la mayoría de métodos recurre a la Dominancia de Pareto, a través de este método se busca definir la superioridad de una solución sobre la otra, la formulación de dicho concepto orientado a un problema de minimización se muestra a continuación (ecuación 2.4).

$$\begin{aligned}
 & Si \quad \forall i \in \{1, \dots, \text{Cantidad de Objetivos}\}; f_i^{solucion_1} \leq f_i^{solucion_2} \\
 & \wedge \exists i_0 \in \{1, \dots, \text{Cantidad de Objetivos}\}; f_{i_0}^{solucion_1} < f_{i_0}^{solucion_2} \quad (2.4) \\
 & \Rightarrow solucion_1 p < solucion_2
 \end{aligned}$$

Si la $solucion_1$ es parcialmente menor ($p <$) que la $solucion_2$, se asegura que la $solucion_1$ domina a la $solucion_2$, por tanto es superior. Si en cambio ningún individuo domina al otro se consideran equivalentes. El rango de un individuo depende del número de soluciones que lo dominan [60],[61]. Métodos basados en Pareto son MOGA, NSGA-II, SPEA2 (Strenght Pareto Evolutionary Algorithm) [62],[63],[64].

Además de las técnicas basadas en Pareto, podemos encontrar métodos más simples dirigidos a problemas multiobjetivo concernientes a aplicaciones prácticas, en estos casos resulta ventajoso estimar la importancia relativa de cada objetivo. Una técnica se conoce como Agregación o Combinación lineal de Objetivos, aquí a cada criterio se le asigna un peso específico, luego por medio de una suma ponderada se obtiene un valor Fitness combinado (ecuación 2.5) [65],[66].

$$F_{combinada} = \sum_{i=1}^{\#objetivos} W_i \cdot f_i, \quad W \rightarrow peso \wedge f \rightarrow funcion\ Fitness(objetivo) \quad (2.5)$$

2.8.2. Estrategias de Selección de Progenitores

Una vez que a cada individuo le ha sido asignado un valor Fitness, a continuación se procede a escoger a las soluciones que van a participar en el proceso de reproducción, entre las técnicas más usadas distinguimos: Rueda de Ruleta (Roulette Wheel), Muestreo Universal Estocástico, selección por Truncamiento, por Torneo y Boltzman. Un algoritmo de selección ideal muestra un bias reducido (preferiblemente de cero), una dispersión (spread) mínima y una alta eficiencia; donde el bias mide la precisión, a partir de la diferencia absoluta entre la probabilidad actual de algún individuo de ser escogido, respecto del número esperado de copias generadas del mismo; dispersión es una medida de consistencia y se define como el rango de valores de probabilidad esperada que se asignan a un individuo; y la eficiencia trata sobre el tiempo computacional requerido por el algoritmo [52],[67].

La selección mediante Rueda de Ruleta (figura 2.5) es el esquema más simple, los individuos son mapeados como segmentos contiguos, tal que cada segmento es proporcional al valor de Fitness del individuo, luego un número (puntero) es generado aleatoriamente; así el elemento en cuyo segmento se ubique el puntero resulta seleccionado [68].

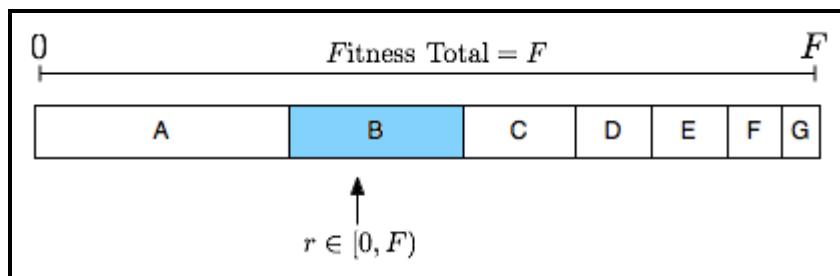


FIGURA 2.5. RUEDA DE RULETA

El Muestreo Universal Estocástico (figura 2.6) provee bias cero y dispersión mínima, los individuos son mapeados como en la selección por Rueda de Ruleta, luego punteros equidistantes se posicionan sobre los segmentos concatenados (se crean tantos punteros como individuos requieran ser escogidos). La posición del primer puntero se determina por un número aleatorio entre cero y la distancia entre punteros [69].

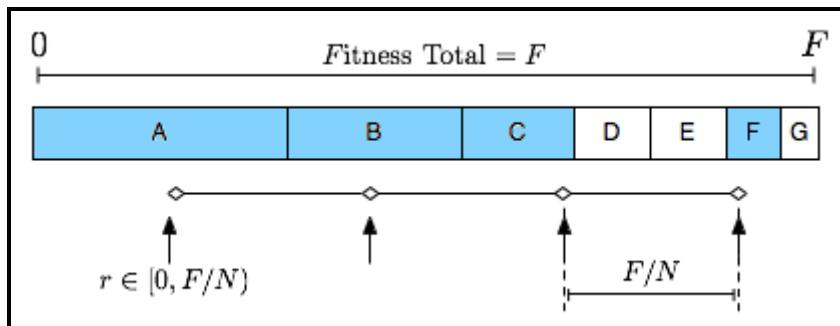


FIGURA 2.6. MUESTREO UNIVERSAL ESTOCASTICO

Comparado con otros procedimientos, selección por truncamiento es considerada un método artificial, sin embargo se utiliza a menudo cuando se tiene una población elevada. De acuerdo a este algoritmo los individuos son ordenados con relación a su Fitness y sólo los mejores son escogidos para convertirse en padres. El umbral de truncamiento es un parámetro, el cual indica la proporción de población escogida como progenitores, dicho argumento adquiere en la práctica valores entre 10% y 50%. Sujetos debajo del umbral no producen descendencia [70].

En la selección por torneo, un número de individuos (Tour) es seleccionado al azar entre la población, después atraviesan una etapa de competición, al final de la cual sólo el ganador del grupo es escogido como parente. En este método el parámetro “Tour” refiere al tamaño del grupo de competidores, y ostenta magnitudes entre 2 hasta el número total de elementos en la población. Esta metodología permite ajustar la presión de selección fácilmente al modificar el Tour, mientras mayor sea este entonces sujetos más débiles tendrán menos oportunidades de ser elegidos, asimismo existirá menor diversidad en la población [71].

La selección Boltzman simula el fenómeno de enfriamiento lento del metal fundido, una temperatura variable controla la tasa de selección de acuerdo a un itinerario preestablecido; la temperatura inicia elevada (baja presión de selección), luego decae gradualmente reduciendo así el espacio de búsqueda y conservando una diversidad apropiada en la población. Normalmente un descenso logarítmico resulta propicio para lograr la convergencia del GA sin estancarse en un mínimo local [72].

2.8.3. Restricciones de Apareamiento

Los procedimientos previos permiten la elección de al menos un parente, al escoger los subsiguientes progenitores se deben considerar ciertas restricciones, fundamentadas en Modelos Poblacionales. Se definen tres modelos básicos: global, local y regional (figura 2.7).

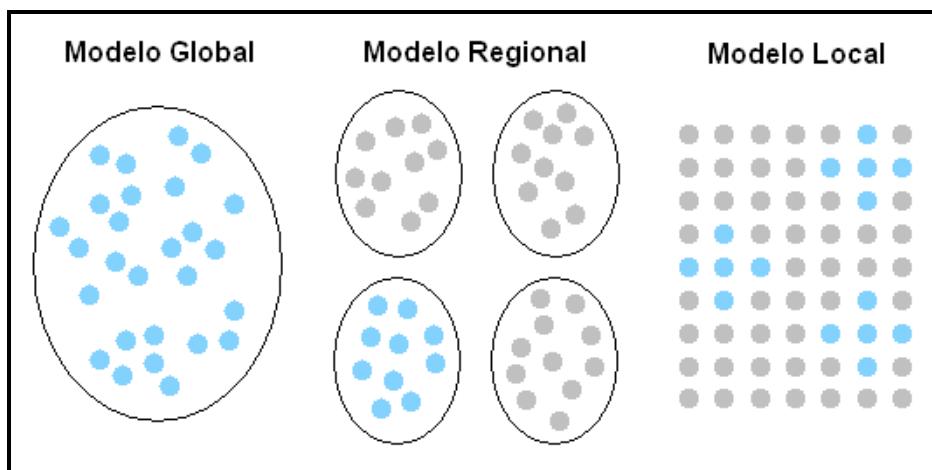


FIGURA 2.7. MODELOS POBLACIONALES

En el modelo global la selección se desarrolla dentro de la totalidad de la población, esto significa que dos o más individuos cualesquiera, son escogidos para producir descendencia, sin restricción alguna [73].

En el modelo Local un individuo habita en un vecindario local, y exclusivamente dentro de éste se selecciona la pareja de apareamiento; sin embargo la información aún se propaga entre todos los individuos debido a la superposición de vecindades. Existen topologías de vecindad Lineal (Full Ring o Half Ring), o Bidimensional (Full Cross, Half Cross, Full Star y Half Star) (ver figura 2.8) [74],[75].

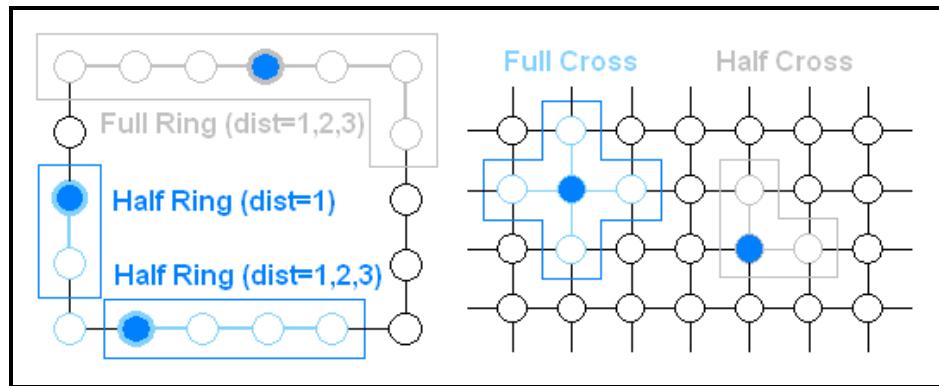


FIGURA 2.8. TOPOLOGIAS DE VECINIDADES LOCALES

El modelo Regional o Migracional, divide la población en múltiples subpoblaciones, que evolucionan independientemente durante cierto número de generaciones (tiempo de aislamiento). Después del cual una cantidad de individuos (tasa de migración) es redistribuida entre las subpoblaciones. La selección de migrantes usa las técnicas ya vistas (rueda de ruleta, etc.). Existen varios modos de estructurar la migración (ver figura 2.9), la estrategia general es migración sin restricciones (topología net), otros esquemas conciben la migración sólo entre subpoblaciones vecinas (topologías anillo o rejilla) [76].

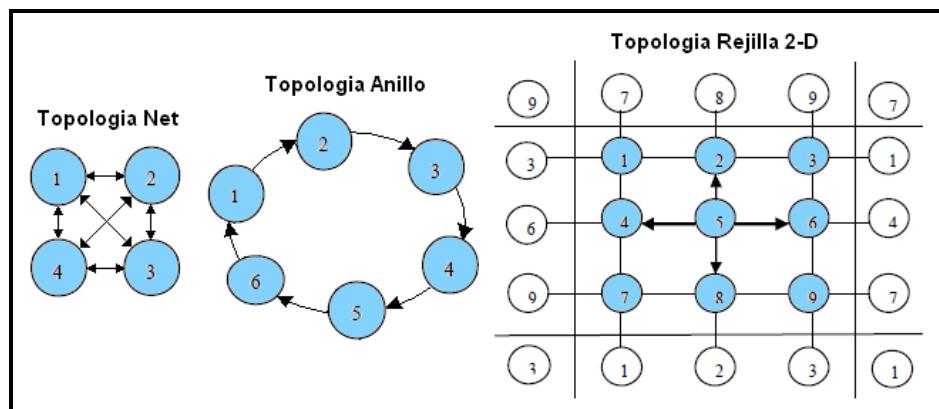


FIGURA 2.9. TOPOLOGIAS DE SUBPOBLACIONES REGIONALES

2.9. Reproducción

Una vez seleccionado el grupo de posibles progenitores (Mating Pool) que darán origen a la siguiente generación, es necesario el uso de operadores genéticos para producir la descendencia requerida. Básicamente son aplicados dos operadores Recombinación o Crossover y Mutación.

2.9.1. Recombinación o Crossover

Es análogo a la Recombinación genética en la naturaleza, y al igual que su contraparte genera nuevos sujetos quienes presentan fracciones del material genético de sus padres. Esta operación tiene relación con la representación genética (encoding), de esta forma cada representación ofrece una variante propia de Recombinación, razón por la que podemos encontrar una abundancia de algoritmos dentro de la literatura, algunos de los cuales son explicados en seguida [77].

Recombinación Discreta (ecuación 2.6), destinada hacia cualquier representación genética, de acuerdo a este algoritmo para cada posición genética se escoge aleatoria y uniformemente cual progenitor contribuirá con su variable hacia la descendencia. La Recombinación discreta genera posible descendencia sólo en las esquinas del hipercubo definido por los padres [78].

$$G_i^{Desc} = a_i \cdot G_i^{Padre1} + (1 - a_i) \cdot G_i^{Padre2}, \quad i \in \{1, \dots, \# genes\} \quad (2.6)$$

$a_i \in [0,1] \rightarrow \text{distribucion uniforme}$

Ciertas formas de Recombinación se usan específicamente para Representaciones con Números Reales como las siguientes:

- Recombinación Intermedia (ecuación 2.7), este método crea nuevas soluciones alrededor y entre los progenitores (ver figura 2.10).

$$G_i^{Desc} = a_i \cdot G_i^{Padre1} + (1-a_i) \cdot G_i^{Padre2}, \quad i \in \{1, \dots, \# genes\} \quad (2.7)$$

$$a_i \in [-d, 1+d]$$

Aquí “ a ” es un factor de escala seleccionado aleatoriamente para cada gen, el parámetro “ d ” indica la magnitud de la región vectorial donde se encontrará la posible prole.

- Recombinación Lineal: similar a la Recombinación Intermedia, salvo que se utiliza un único factor de escala (a), así la prole se genera dentro de la región lineal comprendida por ambos progenitores.
- Recombinación Lineal Extendida: El dominio de genes define el tamaño de la región de descendencia. La descendencia tiende a concentrarse alrededor del padre de mejor Fitness (figura 2.10) [79].

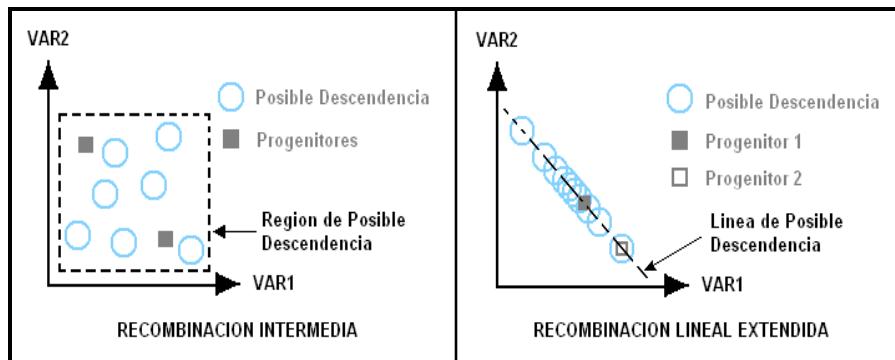


FIGURA 2.10. RECOMBINACION PARA NUMEROS REALES

La Recombinación en Representaciones Binarias es comúnmente conocida como Crossover o Sobrecrezamiento, aquí tenemos:

- Crossover Uniforme: es la Recombinación Discreta aplicada a representaciones binarias (figura 2.11).
- Crossover con Punto de corte único/doble/múltiple: Primero se definen aleatoriamente los Puntos de corte, a continuación los bits ubicados entre dos cortes sucesivos se intercambian entre ambos padres, produciendo así dos nuevos descendientes (figura 2.11).
- Otras formulaciones son el Crossover Mezclado (Shuffle Crossover), Crossover Ordenado, Crossover Uniforme Medio (HUX), Crossover Aritmético, Crossover Heurístico o Crossover de Tres Padres [80].

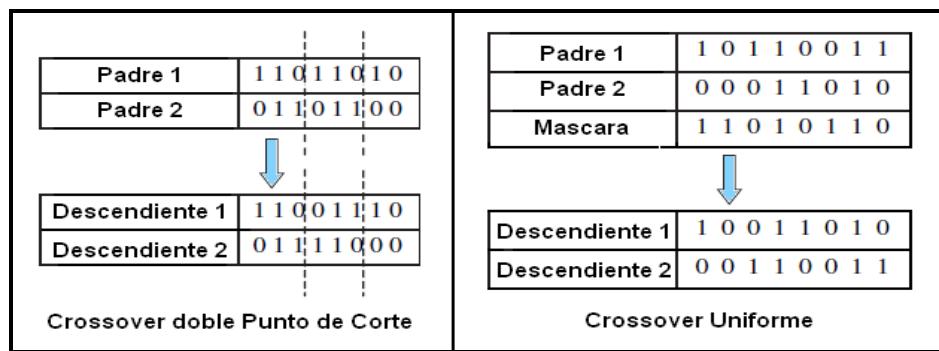


FIGURA 2.11. CROSSOVER EN REPRESENTACIONES BINARIAS

Probabilidad de Recombinación (P_c): Parámetro que describe la frecuencia de ejecución del operador Crossover dentro de un Algoritmo Evolutivo. Si es 100% entonces toda la descendencia nace a partir de Recombinación, de ser 0% la población resultaría un duplicado de la anterior. En general se recomienda un valor P_c alrededor del 70% [81].

2.9.2. Mutación

Este operador tiene como finalidad preservar e introducir diversidad genética en una población. Tradicionalmente se estima que la Mutación previene que un Algoritmo Evolutivo se estanke en un Mínimo Local de la función Fitness al evitar un aumento en la semejanza de los genes en la población; también cumple el rol de recuperar material genético perdido durante la selección o Recombinación. Si todos los parámetros implicados en la Mutación permanecen constantes entonces es vista como estática, pero si en cambio estos factores varían de acuerdo a mutaciones previas, es denominado un comportamiento autoadaptativo. La literatura ofrece diversos operadores de Mutación y su uso depende estrechamente de la representación genética elegida [82].

Una Mutación Estática en Representaciones con Números Reales consiste en agregar valores aleatorios con una baja probabilidad de ocurrencia a cada variable genética. Para ello se definen 2 parámetros, la probabilidad de mutar de un gen (tasa de mutación) y la magnitud de los cambios (paso de mutación). La ecuación 2.8 representa un operador de Mutación Estática que permite generar mutantes dentro del hipercubo definido por el individuo original y el rango de mutación [83].

$$\begin{aligned}
 G_i^{Mutante} &= G_i + s_i \cdot r_i \cdot a_i, i \in \{1, 2, \dots, \# genes\} \\
 s_i &\in \{-1, +1\} \\
 r_i &= r \cdot dominio_i, r : \text{rango de mutacion} \\
 a_i &= 2^{-u \cdot k}, u \in [0, 1], k : \text{factor de precision}
 \end{aligned} \tag{2.8}$$

Una Mutación Estática en Representaciones Binarias altera el valor de cada gen en base a una probabilidad ínfima, cuya magnitud es usualmente la inversa de la longitud del genoma ($1/L$); además en éste ámbito se manejan tres conceptos relevantes: conmutación, intercambio e inversión de bits (ver figura 2.12). Comutar involucra una negación lógica de bits, basada en una máscara de mutación aleatoria. El intercambio de bits implica la permutación aleatoria de dos genes dentro de una cadena. En la inversión de genes, una posición aleatoria es escogida, luego los dos bits más cercanos son permutados entre sí [82].

<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">Padre</td><td style="padding: 2px;">1 0 1 1 0 1 0 1</td></tr> <tr><td style="padding: 2px;">Mascara</td><td style="padding: 2px;">1 0 0 0 1 0 0 1</td></tr> <tr><td style="padding: 2px;">Descendiente</td><td style="padding: 2px;">0 0 1 1 1 1 0 0</td></tr> </table> <p>Mutacion por Comutacion</p>	Padre	1 0 1 1 0 1 0 1	Mascara	1 0 0 0 1 0 0 1	Descendiente	0 0 1 1 1 1 0 0	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">Padre</td><td style="padding: 2px;">1 0 1 1 0 1 0 1</td></tr> <tr><td style="padding: 2px;">Mascara</td><td style="padding: 2px;">0 1 0 0 0 1 0 0</td></tr> <tr><td style="padding: 2px;">Descendiente</td><td style="padding: 2px;">1 1 1 1 0 0 0 1</td></tr> </table> <p>Mutacion por Intercambio</p>	Padre	1 0 1 1 0 1 0 1	Mascara	0 1 0 0 0 1 0 0	Descendiente	1 1 1 1 0 0 0 1
Padre	1 0 1 1 0 1 0 1												
Mascara	1 0 0 0 1 0 0 1												
Descendiente	0 0 1 1 1 1 0 0												
Padre	1 0 1 1 0 1 0 1												
Mascara	0 1 0 0 0 1 0 0												
Descendiente	1 1 1 1 0 0 0 1												
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">Padre</td><td style="padding: 2px;">1 0 1 1 0 1 0 1</td></tr> <tr><td style="padding: 2px;">Mascara</td><td style="padding: 2px;">0 0 0 0 0 0 0 1</td></tr> <tr><td style="padding: 2px;">Descendiente</td><td style="padding: 2px;">1 0 1 1 0 1 1 0</td></tr> </table> <p>Mutacion por Inversion</p>		Padre	1 0 1 1 0 1 0 1	Mascara	0 0 0 0 0 0 0 1	Descendiente	1 0 1 1 0 1 1 0						
Padre	1 0 1 1 0 1 0 1												
Mascara	0 0 0 0 0 0 0 1												
Descendiente	1 0 1 1 0 1 1 0												

FIGURA 2.12. MUTACION EN REPRESENTACIONES BINARIAS

Existen otros operadores como la Mutación Adaptativa, donde es posible aprender la dirección y tamaño de paso de mutaciones exitosas, dichos métodos requieren una mayor capacidad computacional y en general utilizan poblaciones reducidas (≤ 3 individuos) [84]. Otros métodos parametrizan la Mutación, de forma que la tasa de Mutación decrece al converger hacia la solución e incluso existen operadores que ejecutan Mutación sólo en los individuos de menor Fitness [85].

2.10. Modelos Evolutivos

Una vez definidos los operadores reproductivos que se van a emplear en el Algoritmo Genético, el siguiente paso consiste en determinar la cantidad de descendencia a ser creada, para luego combinar cierto porcentaje de la misma con la generación anterior, con la intención de producir una nueva Población.

Son ampliamente aceptados tres modelos evolutivos, el más tradicional llamado Modelo Generacional consiste en reemplazar completamente la población original en cada paso generacional por la nueva descendencia producida. Un segundo enfoque conocido como Estado Estable o Actualización de uno a la vez, implica procrear por cada ciclo generacional sólo uno o dos individuos e insertarlos en la población original. Finalmente el modelo de Algoritmos Genéticos Secuenciales, se asume como un método intermedio, donde sólo un porcentaje de la población original es reemplazada por la descendencia, dicho porcentaje se le conoce como GAP generacional (GAP: Genetic Algorithm Percentage) [86],[87],[88].

2.11. Reinserción o Reemplazo

Los esquemas de Reinserción respetan los conceptos definidos en los modelos evolutivos así como las restricciones relativas a modelos poblacionales vistas anteriormente, por ello se puede hablar de Reinserción Global, Regional o Local, de acuerdo a la subpoblación o vecindario correspondiente. Asimismo el Modelo Evolutivo Secuencial presenta múltiples alternativas, entre las más empleadas se mencionan las siguientes:

- Producir tanta descendencia como padres y reemplazar todos los padres por la descendencia (Reinserción pura – modelo generacional).
- Producir menos descendencia que padres y reemplazarlos aleatoriamente, o mediante alguna metodología de selección. En otras variantes la descendencia reemplaza a sus propios padres o también se sustituye al miembro más similar dentro de la población existente.
- Producir menos descendencia que padres y reemplazar sólo a los peores padres (Reinserción Elitista).
- Producir mayor descendencia que padres e introducir sólo la mejor descendencia (Reinserción basada en Fitness), es necesario implementar una selección por truncamiento en la prole como paso preliminar.
- Una combinación de las dos técnicas previas implica introducir la mejor descendencia y reemplazar a los peores progenitores, siempre que la prole sea mejor que los padres (Reinserción Elitista basada en Fitness) [89],[90].

2.12.Terminación o Finalización de la Búsqueda

Una vez revisadas todas las definiciones previas, sólo resta proceder a implementar el Algoritmo Genético, en caso nuestro programa funcione perfectamente y alcance la solución óptima, es importante detener la evolución a fin de evitar la posibilidad que dicha solución desaparezca y sea reemplazada por individuos deficientes; también se pueden encontrar complicaciones durante la evolución, ya sea por una elección inadecuada de parámetros o simplemente debido a la aleatoriedad inherente al GA, todo lo cual puede provocar la convergencia prematura del algoritmo hacia soluciones indeseables o requerir tiempos inviables de convergencia.

Para resolver estas contingencias debemos prever nuestro algoritmo con criterios de convergencia y de terminación que detengan o finalicen respectivamente nuestro programa.

Condiciones de Parada a razón de Convergencia Prematura:

- Máximo Número de Generaciones: el algoritmo se detiene al llegar al límite de ciclos generacionales preimpuesto.
- Tiempo Transcurrido: el proceso concluye al cabo de un tiempo prefijado.
- Fitness estable: el programa finaliza sino existe un progreso en la mejor Fitness de la población durante un número particular de generaciones.
- Generaciones Paralizadas: el algoritmo se interrumpe si no existe mejora en la Fitness promedio durante una serie de generaciones sucesivas.
- Límite de Tiempo Paralizado: el proceso se suspende cuando no se halla incremento en la Fitness durante un tiempo específico.

Condiciones de Terminación a razón de Resultados Óptimos Alcanzados:

- Mejor Individuo: la búsqueda se paraliza una vez que el máximo valor de Fitness dentro de la población descienda por debajo de un valor prefijado.
- Peor Individuo: el proceso concluye cuando el peor individuo presenta un valor Fitness inferior a un criterio preestablecido.
- Suma de Fitness: la convergencia es satisfactoria cuando la suma de Fitness de todos los individuos sobrepasa o iguala un valor preseñalado.
- Fitness Promedio: aquí por lo menos la mitad de los individuos deberán ser mejores o iguales a un criterio exitoso de convergencia [91],[92],[93].

2.13. Clasificación de los Algoritmos Genéticos

- Algoritmo Genético Simple (SGA): Básicamente está compuesto por tres operadores Reproducción, Crossover y Mutación, exhibe buenos resultados cuando el espacio de búsqueda es grande, complejo o poco conocido, y uno no dispone de un análisis matemático profundo sobre el problema.
- Algoritmo Genético Paralelo (PGA): Ejecutan múltiples SGAs en paralelo con el fin de reducir tiempos computacionales y abarcar un mayor espacio de búsqueda, a la vez permiten una fácil implementación en redes de computadoras. La forma más sencilla de PGA requiere combinar copias de un mismo SGA y ejecutarlas en paralelo con diferentes subpoblaciones iniciales, adicionalmente se pueden agregar conceptos como la migración de individuos entre subpoblaciones. PGAs de grano fino o CGAs (GAs celulares) sólo admiten la Reproducción en la vecindad próxima a un individuo, mientras PGAs de grano grueso o DGAs (GAs distribuidos) permiten la Reproducción dentro de subpoblaciones, la combinación de ambos conduce a los PGAs jerárquicos (HPGAs) donde los DGAs ocupan el nivel superior, entretanto CGAs o DGAs el nivel inferior.
- Algoritmo Genético Híbrido (HGA): Cualquier GA que incorpora otras técnicas de búsqueda u optimización como Q-learning, procesos de decisión de Markov, optimización local basada en Gradiente, Hill Climbing, Simulated Annealing, Particle Swarm Optimization (PSO), etc.
- Algoritmo Genético Adaptativo (AGA): Son GAs cuyos parámetros evolutivos, como tamaño de la población, probabilidad de Recombinación o Mutación varían o se adaptan durante la ejecución misma del programa.

- Algoritmo Genético FmGA: Estocástico, con representación binaria de longitud variable, la mayor diferencia con otros GAs se advierte en su habilidad para manipular explícitamente Bloques de Construcción (BBs) de material genético. Propone tres fases; inicialización, donde se determina la dimensión apropiada de la población; filtrado, donde se eliminan ciertos bits de los cromosomas de cada individuo y se seleccionan sólo las mejores cadenas parciales; y yuxtaposicional, cuando se combinan los BBs mediante un operador Crossover de Corte y Empalme.
- Independent Sampling GA: Busca el desempeño idealizado de un GA al combatir el problema de Hitchhiking Genético. Utiliza dos fases; la primera de muestreo independiente donde una Estrategia de Detección de Bloques de Construcción (BBDS) extrae información sobre los BBs, tal que un sujeto sea capaz de construir secuencialmente mejores soluciones parciales (evolución en paralelo de esquemas). La subsiguiente fase de Cruzamiento facilita a los individuos elegir su propia pareja bajo la premisa que tengan una Fitness similar pero un genotipo diferente entre ellos [94],[95].

2.14. Software destinado a Computación Evolutiva

Muchos métodos y operadores empleados dentro del ámbito de los EAs, en la actualidad se encuentran ya incorporados en las librerías de lenguajes de programación como C++ [96], C# [97], Java [98], Python [99], MatLab [100], entre otros. Existen aplicaciones más sofisticadas, que brindan por ejemplo la posibilidad de ejecutar EAs paralelos en múltiples ordenadores conectados a una red local, asimismo favorecen la interacción entre distintos métodos heurísticos como PSO, Hill-Climbing, Simulated Annealing.

Entre estos software encontramos EvA2 [101], ECJ [102], JCLEC [103], JGAP [104] desarrollados en Java, ECF [105], EO [106] en C++, HeuristicLab en C# [107], por mencionar algunos. Son encontrados además programas para Minería de Datos (Keel [108]) y software comerciales (Evolver [109]) orientados a aplicaciones estadísticas, económicas o de planeamiento.

2.15. Software para la Simulación de Sistemas Robóticos

La simulación de Robots incluye Dinámica de Cuerpo Rígido, Reacciones Dinámicas, Cinemática, Colisiones, Fricción, Sensores, Motores; en el presente se encuentra una miscelánea de Simuladores Físicos tales como ODE (Open Dynamics Engines) [110] o Vortex SDK [111] que posibilitan recrear todo tipo de mecanismos, asimismo las interacciones entre ellos y con el ambiente simulado; por otro lado existe software destinado a la Robótica, por ejemplo Webots [112], Player Project [113], Simbad [114]; que incorporan librerías de sensores y actuadores e incluso ya cuentan con la simulación de robots reales.

2.16. Transferencia desde la Simulación hacia la Realidad

Una vez evolucionado virtualmente el controlador, el siguiente paso es transferirlo al robot real y evaluarlo en un entorno tangible; sin embargo esta etapa no aparece libre de problemas, la evolución explota peculiaridades de la simulación no siempre generalizables a la realidad, así una tabla que describa el comportamiento de un sensor no será igual de precisa para otro similar, debido a diferencias de fabricación, igualmente una variación considerable en algún factor simulado (iluminación, etc.) resultará desconocida para el robot.

Es así que si previamente no se toman las previsiones adecuadas, ya en la instancia de transferencia no se encontrarán soluciones satisfactorias [115], [116],[117]. Con el fin de superar este obstáculo han sido propuestas muchas alternativas, entre ellas son 4 los métodos más conocidos y que prometen mejores resultados; el más utilizado debido a su simplicidad consiste en agregar ruido a los valores de sensores y actuadores generados por funciones u obtenidos desde tablas, así como también a la posición instantánea del robot computada por el simulador o si fuera el caso de cada una de las partes que lo constituyen, de esta forma se evita el origen de soluciones que dependan de particularidades del ambiente de simulación; el mayor inconveniente aquí se observa al momento de definir el modelo de sensores o actuadores, puesto que aunque resulta factible generar tablas de valores a partir del muestreo de los componentes reales, dicha técnica pierde efectividad para dispositivos de mayor dimensionalidad como los sensores de visión [118],[119].

Un segundo método llamado simulaciones mínimas, consiste en modelar sólo aquellas características del robot y del ambiente relevantes para la emergencia del comportamiento deseado, dichos rasgos básicos necesitan ser modelados con precisión en la simulación, en cambio las condiciones restantes referidas como aspectos de implementación deben modificarse aleatoriamente a través de las pruebas de los individuos, para de esta manera asegurar que las soluciones dependan solamente de los rasgos básicos, entretanto dichas características elementales también precisan ser alteradas hasta cierto punto con la finalidad de garantizar una mayor robustez, una crítica en contra de este método parte de la dificultad para predecir cuales rasgos básicos son relevantes para una problema determinado [120].

Un tercer procedimiento consiste en codificar genéticamente y evolucionar reglas de aprendizaje para el sistema de control, los parámetros del controlador son inicializados con valores aleatorios al principio de cada prueba para cada individuo, luego estos valores deberán auto-organizarse empleando las reglas de aprendizaje, bajo dicha perspectiva se fomenta el surgimiento de sistemas de control que sin embargo permanecen adaptables a entornos parcialmente desconocidos, una vez en el ambiente concreto el robot perfeccionará en tiempo real sus parámetros de control considerando las particularidades del mundo físico [121],[122].

El último método se fundamenta en la coevolución del robot (control y/o morfología) y aquellos parámetros de simulación con mayor probabilidad de diferir respecto al ambiente real; el procedimiento presenta dos etapas, en primer lugar una población aleatoria de robots es evolucionada en un simulador por defecto, luego el mejor individuo es evaluado en el robot real y se registran las lecturas de sus sensores, actuadores o de cualquier otro parámetro requerido por el simulador, ya en la segunda etapa una población de simuladores es evolucionada con el objetivo de reducir la diferencia entre la data obtenida por el simulador y el registro recabado desde la realidad, el mejor simulador es a continuación usado para una nueva primera etapa con una nueva población aleatoria de robots y así este proceso se repite una y otra vez hasta que el error entre simulación y realidad se reduzca al mínimo posible, a la vez que el controlador logre ejecutar el comportamiento esperado sumido en el ambiente real [123].

CAPITULO 3

ALGORITMOS GENETICOS Y REDES NEURONALES

3.1. Neuroevolución

Ambos Algoritmos Genéticos y Redes Neuronales toman inspiración de los sistemas biológicos, por tal motivo no resulta sorprendente que sean utilizados en conjunto para la resolución de diversos problemas. El empleo de GAs en combinación con Redes Neuronales se orienta hacia tres facetas distintas, en un primer caso los GAs se usan para encontrar pesos sinápticos y parámetros de aprendizaje apropiados en arquitecturas prefijadas. En segunda instancia los Algoritmos Genéticos son utilizados a fin de evolucionar topológicamente las Redes Neuronales. Una tercera aplicación importante dispone de GAs para seleccionar data de entrenamiento e interpretar el comportamiento resultante de las Redes Neuronales Artificiales (ANNs) [124].

A diferencia de un aprendizaje ontogenético donde un único agente aprende de manera incremental, la Neuroevolución (NE) utiliza una población de soluciones quienes aprenden de forma colectiva o filogenética. Unas de las razones que favorecen la aplicación de NE, es la generalidad exhibida por los GAs quienes toleran elevados niveles de ruido en la información recibida y no requieren de restricciones particulares en la definición de problema, el cual puede ser continuo o parcialmente observable [125].

En particular Redes Neuronales Recurrentes (RNNs) han sido concebidas para solucionar esta última clase de problemas, pero sus algoritmos de entrenamiento en la mayoría de los casos son ineficientes e inviables de generalizar; puesto que los GAs son optimizadores universales resultan el recurso ideal para evolucionar los parámetros y topologías de las RNNs [126].

3.2. Redes Neuronales Artificiales

Las ANNs son modelos matemático-computacionales que intentan simular la estructura y funcionalidad de las redes neuronales biológicas. Son sistemas adaptativos no-lineales que aprenden a desarrollar funciones específicas en base a data interna o externa suministrada a la red. Las ANNs consisten en un grupo de neuronas interconectadas, que trabajan en conjunto y con alto grado de paralelismo. En ingeniería son utilizadas para modelar relaciones complejas entre entradas y salidas y como clasificadores de patrones.

Las Redes Neuronales Feedforward son denominadas de esta manera, puesto que la información sólo se transmite en una dirección desde las neuronas de entrada hacia las de salida a través de capas ocultas, por lo tanto no existen ciclos o lazos cerrados dentro de la red. Para esta experiencia se consideran modelos de Redes Perceptron Simple, Perceptron Multicapa y Función Base Radial. En contraste las RNNs permiten formar ciclos o lazos entre las conexiones neuronales, la salida de una neurona podría también representar una entrada para sí misma en una iteración subsiguiente. En este estudio se pretende manejar Redes Recurrentes completamente conectadas, Redes de tiempo continuo y Echo State Networks (ESN).

Mientras tanto las Redes de Impulsos presentan básicamente la misma interconectividad que las RNNs, pero la diferencia radica en que las neuronas se comunican mediante pulsos que se disparan sólo cuando su potencial de activación alcanza un umbral específico.

3.3. Redes Perceptrón

Perceptrón simple o multicapa son los tipos más simples de ANNs, se les considera redes Feedforward que utilizan una o múltiples capas de neuronas con funciones de activación no-lineales generalmente sigmoides.

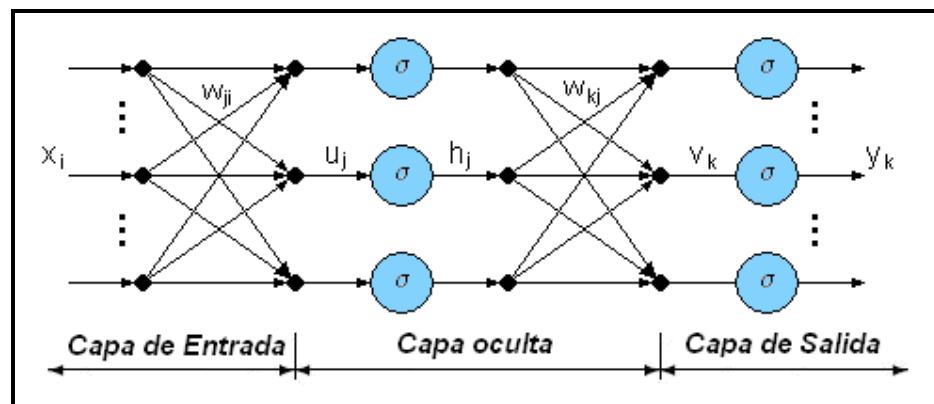


FIGURA 3.1. RED PERCEPTRON MULTICAPA

La figura 3.1 muestra una topología de tres capas, una de entrada otra de salida y una capa oculta. El resultado obtenido en cualquier neurona oculta o de salida se expresa mediante las siguientes ecuaciones 3.1 [127].

$$h_j = \sigma\left(\sum_{i=1}^P w_{ji} \cdot x_i\right) \quad \wedge \quad y_k = \sigma\left(\sum_{j=1}^L w_{kj} \cdot h_j\right) \quad (3.1)$$

3.4. Redes de Función de Base Radial (RBFs)

Radial Basis Function (RBFs) utilizan Funciones Radiales en la capa oculta a manera de activación, una Función de Base Radial se define como cualquier función real cuyo resultado sólo depende de la distancia hacia el origen o hacia un punto central (c); en general se considera $\varphi(x, c) = \varphi(\|x - c\|)$. Por lo general se emplea solamente una capa oculta, ya que la salida siempre debe ser una combinación lineal de funciones radiales.

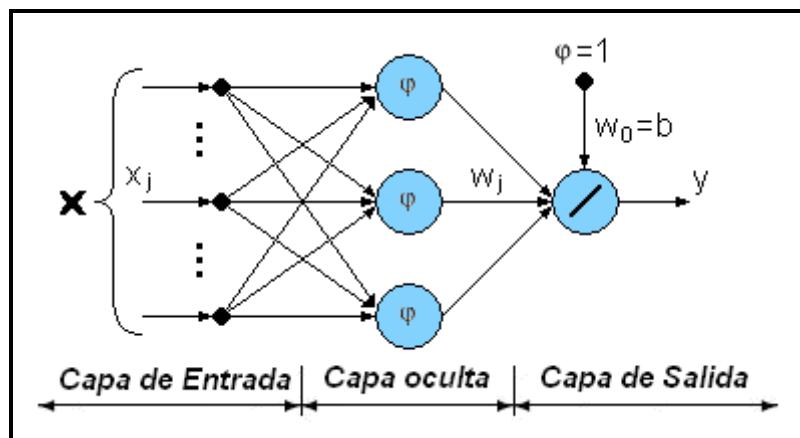


FIGURA 3.2. RED DE FUNCION DE BASE RADIAL

La figura 3.2 evidencia una topología típica de redes RBFs, un vector (\mathbf{X}) es considerado como entrada hacia todas las funciones radiales, asimismo en cada salida se obtiene el siguiente resultado (ecuación 3.2) [128]:

$$\begin{aligned}
 y(\mathbf{x}, \mathbf{c}_0, \mathbf{c}_1, \dots, \mathbf{c}_n) &= w_0 + \sum_{j=1}^N w_j \cdot \varphi(\mathbf{x}, \mathbf{c}_j), \quad \mathbf{c} \rightarrow \text{vector central} \\
 \varphi(\mathbf{x}, \mathbf{c}_j) &= e^{-\beta \cdot \|\mathbf{x} - \mathbf{c}_j\|^2}, \quad \beta > 0
 \end{aligned} \tag{3.2}$$

3.5. Redes Recurrentes Completamente Conectadas (FRNNs)

Fully Recurrent Neural Networks, cada neurona se conecta directamente con las restantes, y cada conexión exhibe un peso modifiable. Algunos nodos son de entradas, otros de salidas y el resto nodos ocultos (figura 3.3) [129].

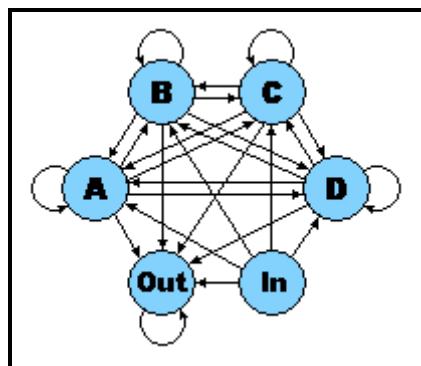


FIGURA 3.3. RED FRNN

3.6. Redes Echo State Network (ESNs)

Presentan una capa oculta llamada Reservorio Dinámico (DR) compuesta por una gran cantidad de neuronas de cualquier tipo (cientos de neuronas), las cuales no se conectan completamente sino presentan una conectividad dispersa, alrededor del 1% del total de conexiones posibles se encuentran activas y estas a la vez son prefijadas en una fase de inicialización al igual que sus pesos. Las ESNs ostentan capas de entrada y salida y pueden manifestar retroalimentación (ver figura 3.4); las conexiones, pesos de retroalimentación y entrada se fijan a priori, permitiendo entrenar sólo los pesos de conexiones hacia las salidas.

La idea básica de las ESNs es usar la RNN fija en la capa oculta como un medio no-lineal aleatorio, cuya alta dimensionalidad posibilita reconstruir cualquier salida deseada mediante una combinación lineal [130].

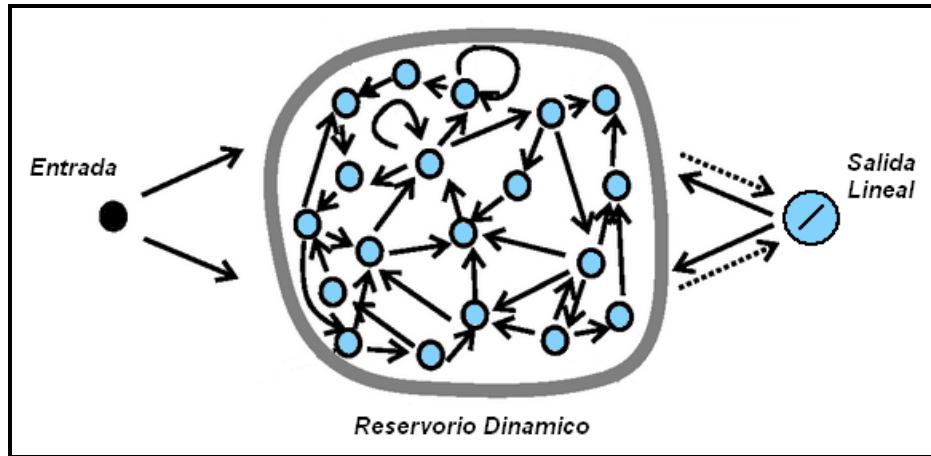


FIGURA 3.4. RED ECHO STATE

3.7. Redes Recurrentes de Tiempo Continuo (CTRNNs)

Continuous-Time Recurrent Neural Networks son redes dinámicas, difieren de las redes clásicas porque cada neurona presenta su propio nivel de activación en función de una constante temporal propia de la neurona. CTRNNs usan ecuaciones diferenciales, permitiéndoles producir una dinámica continua, propiedad de interés para la robótica, ya que admite integrar la variable de tiempo como una entrada más a la red, facilitando tomar decisiones sincronizadas. Una CTRNN exhibe una tendencia a retener el mismo estado ante la ausencia de información externa, acelerando así el aprendizaje de relaciones entre eventos alejados temporalmente. Una de las desventajas de las CTRNNs es la dificultad de entrenarlas mediante algoritmos de aprendizaje supervisado, resultando más simple usar Algoritmos Evolutivos para ello.

$$\frac{dy_i}{dt} = \frac{1}{\tau_i} \left(-y_i + \sum_{j=1}^N w_{ji} \cdot \sigma(y_j + \beta_j) + I_i \right), \quad i = 1, \dots, N, \quad \sigma(x) = \frac{1}{1+e^{-x}} \quad (3.3)$$

La ecuación 3.3 muestra la dinámica interna de una neurona dentro de una red CTRNN, se basa en un modelo no-lineal de primer orden muy utilizado por su simplicidad y su capacidad como aproximador universal dinámico. Aquí y_i representa el potencial de la neurona, τ_i constante de decaimiento, β_j bias, w_{ji} pesos sinápticos e I_i intensidad de entradas sensoriales [131].

3.8. Redes Neuronales de Impulsos (SNNs)

Spiking Neural Networks pertenecen a una nueva generación de redes neuronales Dinámicas, que acrecienta el realismo respecto a la biología, aquí las neuronas no comunican su potencial de activación en cada iteración, en cambio disparan pulsos sólo cuando este alcanza un umbral, generando así un tren de impulsos. Las SNNs prometen reducir el tiempo de ejecución de tareas en neurocontroladores, puesto que la data transmitida entre neuronas no se encuentra codificada en la forma o intensidad de cada pulso, sino en los tiempos de disparo de los pulsos, permitiendo utilizar algoritmos más eficientes para codificar y decodificar la información. Se implementa una SNN al tomar una CTRNN y reemplazar la función de activación por una función Umbral seguida de un generador de Pulsos (función Dirac), además se debe adicionar una conexión negativa de retroalimentación para que la activación disminuya por debajo del umbral inmediatamente después de emitir un pulso, este modelo se conoce como Neurona de Integración y Disparo (figura 3.5).

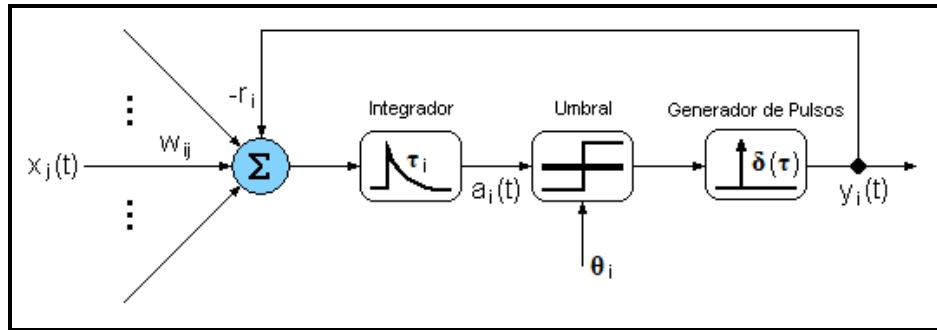


FIGURA 3.5. RED DE INTEGRACION Y DISPARO

Otro modelo es el de Impulso-Respuesta (SRM: Spike Response Model), aquí el potencial de activación se expresa mediante la ecuación 3.4, donde y_j representa el potencial de una neurona (j), ϑ la condición umbral, $t_j^{(f)}$ el tiempo f en que j emite un impulso, F_j un vector que caracteriza la salida de j agrupando los impulsos emitidos, i cualquier neurona presináptica, Γ_j conjunto de neuronas presinápticas respecto de j , $t_i^{(g)}$ tiempo g en que i emite un impulso, F_i vector de impulsos de i , w_{ji} peso sináptico entre j e i , d_{ji} tiempo de retardo de la conexión, $\varepsilon(s)$ función Impulso-Respuesta que describe el cambio de potencial debido a un impulso presináptico, $\eta(s)$ función de Refractoriedad que delinea el cambio de potencial por la propia neurona; $H(s)$ función escalón unitario, y τ_m, τ_s y τ_r constantes de tiempo [132].

$$\begin{aligned}
 y_j(t) &= \sum_{t_j^{(f)} \in F_j} \eta(t - t_j^{(f)}) + \sum_{i \in \Gamma_j} \sum_{t_i^{(g)} \in F_i} w_{ji} \cdot \varepsilon(t - t_i^{(g)} - d_{ji}) \\
 \varepsilon(s) &= \left[\exp\left(-\frac{s}{\tau_m}\right) - \exp\left(-\frac{s}{\tau_s}\right) \right] \cdot H(s), \quad \eta(s) = -\vartheta \cdot \exp\left(-\frac{s}{\tau_r}\right) \cdot H(s)
 \end{aligned} \tag{3.4}$$

CAPITULO 4

GENERACION DE COMPORTAMIENTOS NO-REACTIVOS

4.1. Definición de un Comportamiento No-Reactivivo

Se considera un Comportamiento No-Reactivivo o No-Determinístico aquel donde la acción se ejecuta sin la dependencia de la información sensorial inmediata, para ello un agente debe ser capaz de tomar decisiones basado en su dinámica interna. Efectuar una actividad No-Reactiva requiere de “memoria”, es decir la habilidad de integrar hechos y descubrir patrones a través de largos periodos de tiempo, dicha “memoria” media entre la percepción y la acción permitiendo a los agentes producir comportamientos desfasados de las circunstancias actuales; mientras permanecen aún sensibles a ellas.

Una tarea No-Reactiva se define como aquella solucionable sólo mediante la integración de información a través del tiempo, sin embargo esta interpretación resulta impráctica en aplicaciones reales; ya que no se puede determinar con exactitud para una tarea dada, la inexistencia de posibles soluciones que ejecuten un comportamiento reactivo; la situación se complica aún más en los Algoritmos Evolutivos, creados precisamente para encontrar y depurar estas soluciones triviales y directas.

Estudios realizados por Nolfi y Marocco señalan que en ambientes dinámicos aquellos agentes que dispongan de “memoria” presentarán ventajas sustanciales; y aún cuando inicialmente la evolución exprese preferencia por agentes reactivos, éstos eventualmente darán origen a individuos que manifiesten capacidades No-Determinísticas. En un Neurocontrolador la dinámica interna existe en el estado de activación de sus neuronas y en sus pesos sinápticos, reportándose buen rendimiento en Redes Recurrentes, Redes Dinámicas y Sinapsis Plásticas [133],[134].

4.2. Planteamiento del Problema

Un robot móvil con tracción diferencial deberá atravesar un Laberinto compuesto por intersecciones y bifurcaciones, el laberinto presentará una única zona de Llegada en cada intento o prueba individual, pero ésta variará de ubicación aleatoriamente de una época a otra; el robot realizará dos intentos por época para alcanzar la meta, durante el primer intento se permitirá una exploración total del Laberinto, mientras que en una segunda oportunidad se espera que el robot se dirija directamente desde la posición inicial hasta la posición final, solamente se considerará que un robot ha cumplido satisfactoriamente la tarea cuando atraviese completamente el ambiente en ambas oportunidades, evite transitar regiones prohibidas en el segundo intento y no incurra bajo ninguna circunstancia en colisiones con las paredes del laberinto.

Entretanto para esta experiencia se empleará un Laberinto simplificado en forma de T, es decir que posee una sola bifurcación, éste presenta dos regiones marcadas en el piso, la Partida y la Meta definidas por colores gris y negro respectivamente; mientras tanto el robot dispone de hasta 5 sensores de distancia para detectar las paredes del laberinto, además cuenta con sensores de color que le ayudan a reconocer el color del piso. Este problema puede catalogarse No-Reactiva bajo la definición anterior, puesto que el ambiente no permanece estático entre intentos, igualmente se necesita de agentes capaces de recordar la localización de la posición de Llegada entre intento e intento.

4.3. Características del Hardware y Software

Para las experiencias #1 y #2 se dispuso simular un robot móvil de tracción diferencial, éste presenta 2 ruedas de 47 mm. de diámetro, la distancia entre los centros de sus ruedas comprende 50 mm., el contorno del robot es una circunferencia de 72 mm. de diámetro, cuyo centro es precisamente el punto medio del segmento que une los centros de las ruedas del móvil; el robot consta de 2 servomotores de rotación continua Parallax 900-00008 (Anexo D), cuya velocidad en lazo abierto es controlada mediante una señal PWM (Pulse-Width Modulation) (figura 4.1), estos alcanzan una velocidad límite de 50 RPM, permitiéndole al autómata alcanzar una velocidad lineal máxima de 12.3 cm/s. El robot además se apoya en cinco sensores ópticos-reflexivos TCND5000 (Anexo A) para detectar obstáculos localizados a distancias entre 5 y 45 mm., la ubicación de dichos sensores se considera inicialmente variable a 5mm. hacia el interior de la circunferencia perimetral del móvil, con dicha restricción se consigue eludir el comportamiento no monótono del dispositivo (figura 4.7).

Se encuentran también 4 sensores de color CNY70 (Anexo B) instalados en la superficie inferior del autómata a 0.5 cm. sobre el terreno y 2 cm. del borde exterior, los 4 dispositivos funcionan conjuntamente de forma que eventualmente el controlador principal recibe sólo una única señal triestado, indicando la lectura de la superficie como blanca, negra o gris. Las dimensiones del mencionado robot se pueden observar más claramente en las figuras 4.2, 4.3, 4.4 y 4.5. Para la experiencia #3 se modificaron ciertas dimensiones con el fin de simplificar la complejidad inherente a la acción reactiva y así reducir tiempos en la evolución, las ruedas presentan un diámetro de 50 mm., la distancia entre sus centros es 55 mm. y el contorno del robot es de 60 mm. de diámetro; asimismo los sensores de distancia operan entre 2 y 42 mm., para distancias inmediatamente menores a este rango se observa una discontinuidad tal que el valor de activación en dicha región es similar a la respuesta observada en distancias mayores al rango, gracias a ello estos sensores pueden ser ubicados sobre el perímetro mismo del robot.

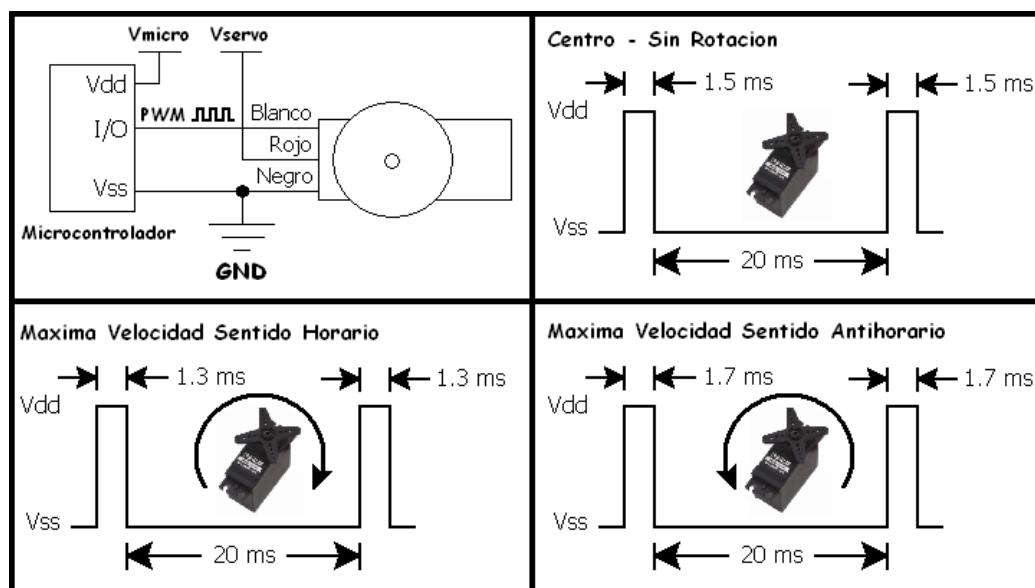


FIGURA 4.1. SEÑAL DE CONTROL PWM

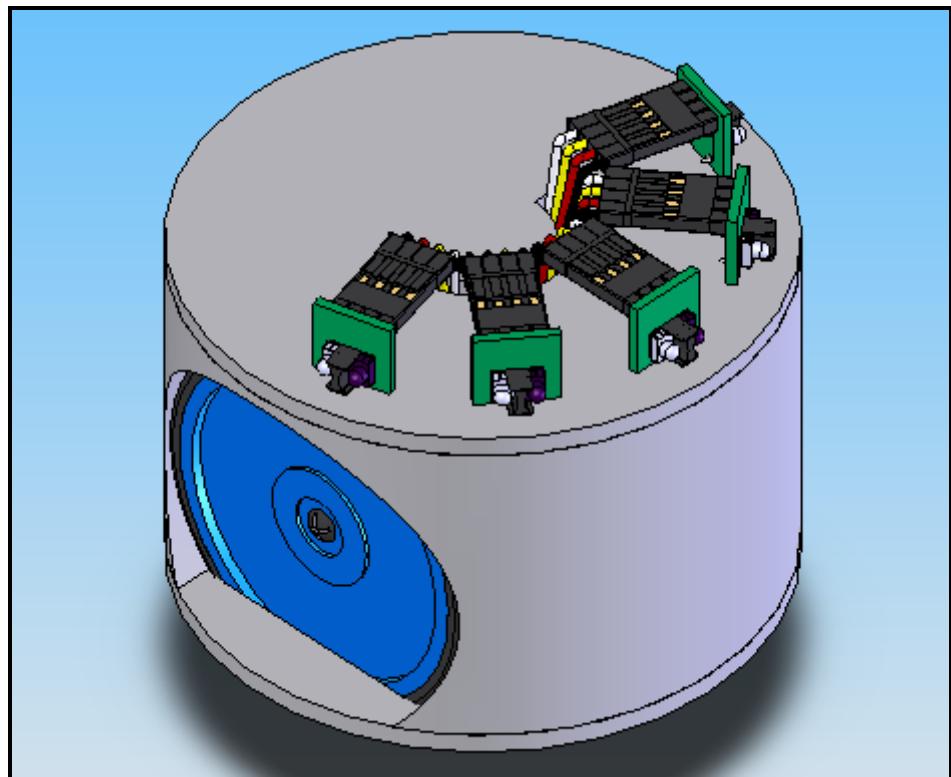


FIGURA 4.2. VISTA ISOMETRICA DEL ROBOT

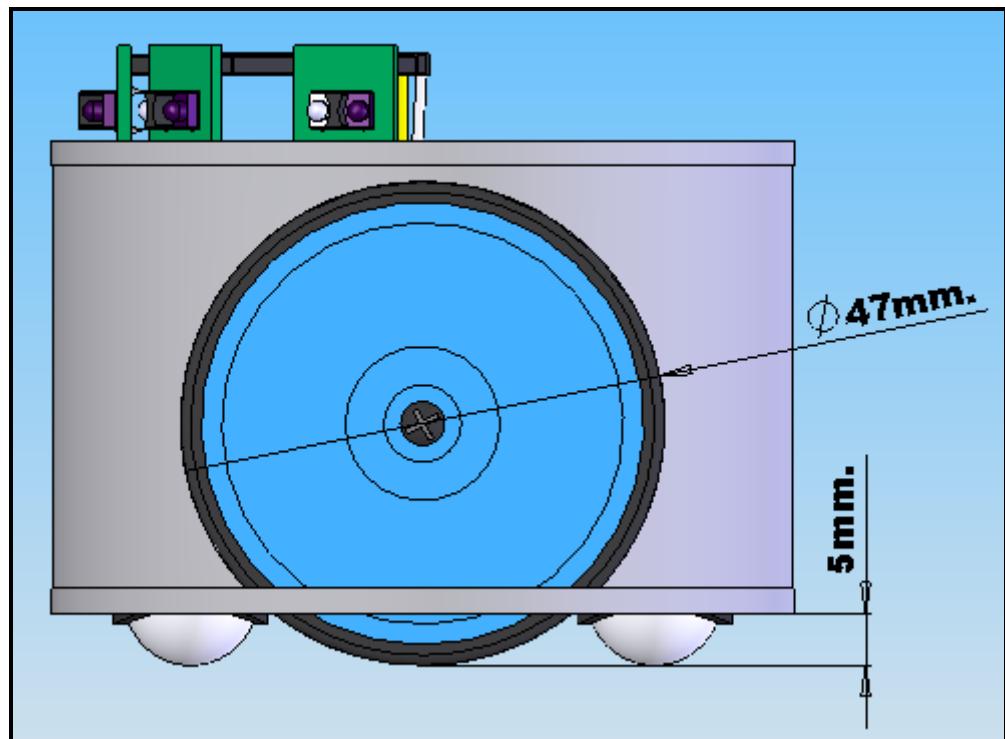


FIGURA 4.3. VISTA FRONTAL DEL ROBOT

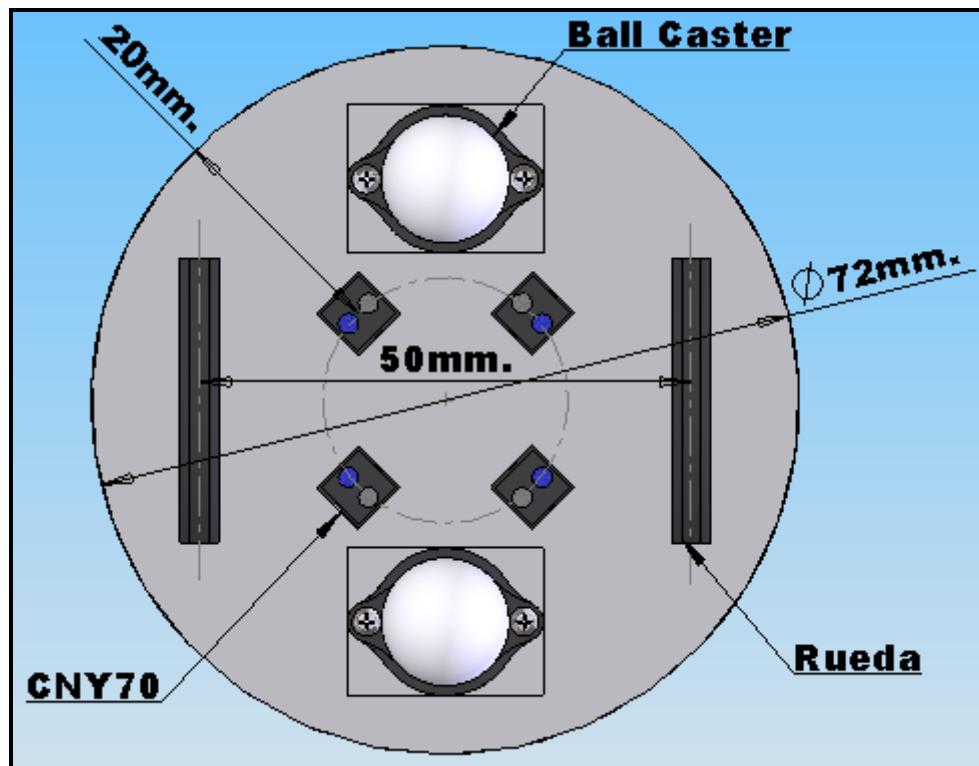


FIGURA 4.4. VISTA INFERIOR DEL ROBOT

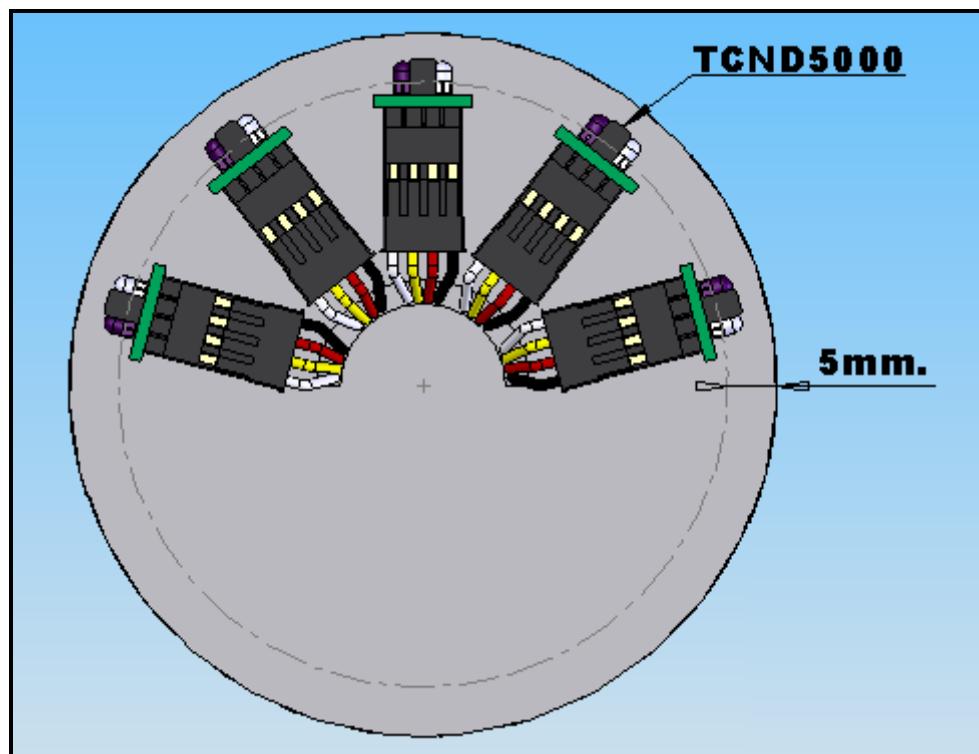


FIGURA 4.5. VISTA SUPERIOR DEL ROBOT

En lo referido al software, los algoritmos se programaron y ejecutaron en su totalidad dentro del entorno Labview 8.6; si bien National Instruments en la actualidad ha adquirido Intelligent Control Toolkit for Labview (ICTL), éste se encontraba en etapa de validación y evaluación al inicio de este informe, por tanto para efectos de este trabajo se implementó de manera propia todo operador y método evolutivo requerido por el mismo y no se utilizaron librerías especializadas (detalles en el Anexo F); entretanto las redes neuronales particulares requeridas en este informe también fueron implementadas de forma exclusiva para este trabajo (detalles en el Anexo G). Toda la experiencia se efectuó en un ordenador Core 4 Quad, de 2.4 GHz y 4 GB RAM.

4.4. Características de la Simulación

Para el ambiente se diseñó una función capaz de crear laberintos de conexión simple aleatoriamente, por conexión simple se entiende que nunca se generan trayectorias cerradas, esta función indica también las posiciones de partida y meta del laberinto. Entrando en detalles, un laberinto cualquiera se representa mediante una matriz, en donde cada celda se interpreta como una región rectangular del espacio real cuya dimensión la define el usuario, cada celda referencia a una geometría del laberinto (pasillo horizontal, esquina, bifurcación, etc), esta lógica es luego usada por funciones de mayor jerarquía. En todas las pruebas acontecidas dentro del contexto de este informe se definieron siempre laberintos cuyas regiones base son cuadrangulares y de dimensión 10 cm. por lado.

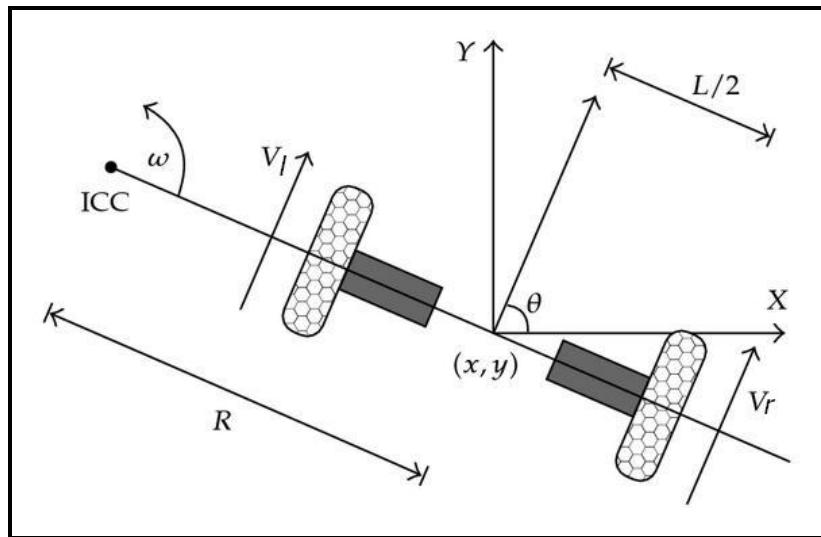


FIGURA 4.6. CINEMATICA DE TRACCION DIFERENCIAL

A su vez el robot, debido a la naturaleza del problema en cuestión, fue simulado basado en cinemática pura, a partir de un modelo destinado para móviles con tracción diferencial [135], este tipo de mecanismo permite variar la velocidad de cada rueda individualmente, por esta razón el Centro Instantáneo de Curvatura (ICC) se localiza en el eje común de ambas tracciones; apoyándose en la figura 4.6 podemos calcular la velocidad rotacional “ ω ” del robot alrededor del ICC y la distancia “ R ” desde ICC hasta el punto medio entre las ruedas (ecuación 4.1).

$$R = \frac{l}{2} \cdot \frac{V_r + V_l}{V_r - V_l}; \omega = \frac{V_r - V_l}{l} \quad (4.1)$$

Considerando al robot en un tiempo “ t ” encontrándose en la posición (x, y) , orientándose en dirección “ θ ” respecto al eje X, entonces para un tiempo “ $t+\delta t$ ” la posición y dirección del robot (x', y', θ') se resuelven a partir de las ecuaciones 4.2 y 4.3.

$$ICC = [x - R \cdot \text{sen}(\theta), y + R \cdot \cos(\theta)] \quad (4.2)$$

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega \cdot \delta t) & -\text{sen}(\omega \cdot \delta t) & 0 \\ \text{sen}(\omega \cdot \delta t) & \cos(\omega \cdot \delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega \cdot \delta t \end{bmatrix} \quad (4.3)$$

En cuanto a los sensores de distancia se empleó la tabla 4.1 que intenta describir someramente las propiedades de los dispositivos reales, dicha tabla se basó en las curvas de “Fotocorriente Relativa Inversa vs Distancia” e “Intensidad Radiante Relativa vs Desplazamiento Angular” fijadas en la datasheet del sensor TCND5000 y mostradas en las gráficas 4.7 y 4.8 respectivamente; puesto que dentro del espectro de este informe no se pretenden realizar experiencias en ambientes reales, así en la creación de esta tabla no se buscó precisión extrema, sino por el contrario se confeccionó a partir de valores solamente tentativos, que no fueron validados de ninguna manera; sin embargo esto no descalifica de ninguna forma los resultados obtenidos, ya que a fin de cuentas la información de la tabla 4.1 no se aleja en demasiado del comportamiento de muchos sensores óptico-reflexivos reales. Las tablas de valores de todos los sensores para las experiencias #1 y #2 se crearon en base a la tabla 4.1 sobre la cual se agregó una aleatoriedad de $\pm 5\%$, dichas tablas permanecen invariables en todas las pruebas, mostrándose en su totalidad dentro del Anexo I (Experiencia #1: Constantes). En forma práctica la realización de una tabla de valores implicaría tomar lecturas del ADC del controlador (por ejemplo en números enteros positivos) para cada sensor a diferentes distancias y ángulos hasta alcanzar el límite de lectura del sensor o un límite autoimpuesto, luego se buscaría el valor de activación máximo de cada dispositivo y con éstos se escalarían las matrices entre [0, 1].

Entretanto en el procesador la entrada sensorial se deberá escalar siempre con respecto a estos valores máximos, siendo los resultados en punto flotante los que ingresen a la red neuronal. Las tablas para la experiencia #3 (Anexo K - Experiencia #3: Constantes) también tomaron como referencia la tabla 4.1, pero con la intención de acelerar la evolución se idealizó un sensor con un rango angular de visión mayor. Dentro del programa de simulación primero se calcula la posición y orientación absoluta de cada sensor de distancia en función de la posición y orientación absoluta del robot y de la posición relativa de cada sensor en el robot de acuerdo a la configuración sensorial predeterminada empleada, luego con el conocimiento de la forma y dimensiones del laberinto se determina la distancia del sensor a la pared más cercana y simultáneamente el ángulo que forma con respecto a la misma, siendo estos dos últimos parámetros las entradas requeridas por la tabla del sensor y que permiten realizar una interpolación lineal doble; en algunas tablas existirá una región estable donde la medición se encuentra fuera de rango, por ejemplo en la tabla 4.1 serían los puntos de valor menor o igual a $I_{RA,REL}=0.07$, dichos puntos no participan en la interpolación bilineal, así si de los 4 puntos que se requieren para dicha interpolación 2 o más estuviesen fuera de rango, entonces se obtendría como resultado el valor estable de la tabla, por otro lado si sólo un punto se encontrase fuera de rango se realizaría una interpolación planar de 3 puntos; asimismo no se ejecuta extrapolación en ningún caso.

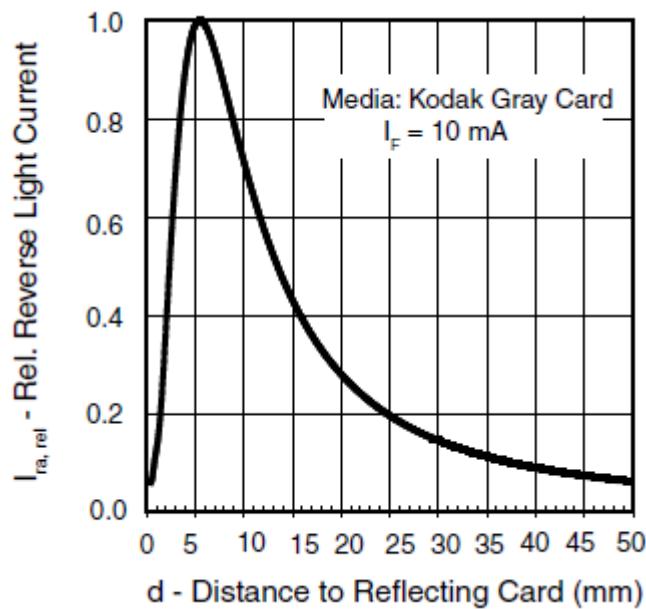


FIGURA 4.7. FOTOCORRIENTE RELATIVA INVERSA VS DISTANCIA

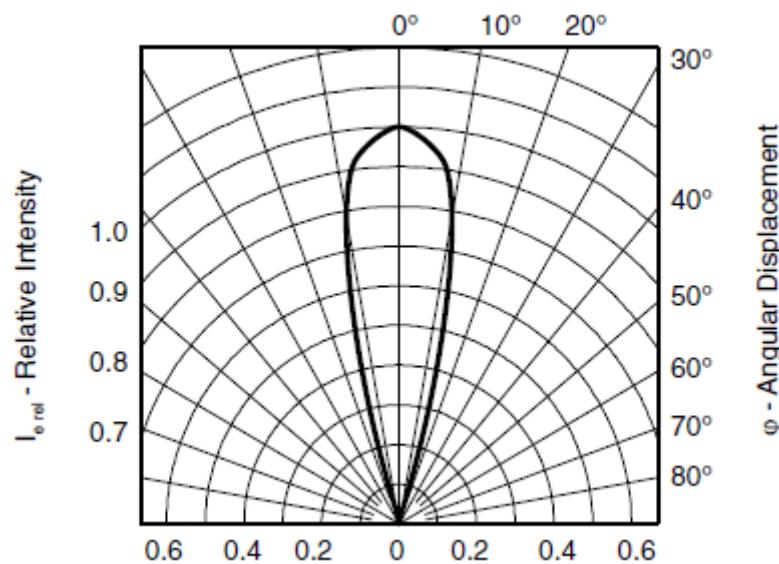


FIGURA 4.8. INTENSIDAD RELATIVA VS DESPLAZAMIENTO ANGULAR

En el caso de los motores se utilizó la tabla 4.2 en las experiencias #1 y #2, que de igual manera trata de describir someramente las características de los componentes reales, para ello se tomó en cuenta la relación de control definida en sus datos técnicos, donde se indica que para un rango de ancho de pulso entre [1300 μ s, 1700 μ s], la velocidad varía en forma directamente proporcional dentro de la región acotada [-50 RPM (horario), 50 RPM (antihorario)], exhibiéndose para un ancho de pulso de 1500 μ s una velocidad de 0 RPM, sobre dicha formulación se agregó un ruido aleatorio uniforme con el fin de confeccionar la tabla 4.2. En la experiencia #3 se emplea una tabla más idealizada mostrada en el Anexo K (Experiencia #3: Constantes). Dentro del programa de simulación se escala linealmente la salida del controlador hacia el rango [1300 μ s, 1700 μ s], que representa el parámetro de entrada para la tabla del motor, y a continuación se efectúa una interpolación lineal.

En caso se desease transferir los resultados a la realidad sería recomendable reemplazar las tablas por mediciones obtenidas a partir de cada dispositivo real y para distintas geometrías (pared infinita, esquina interior, esquina exterior, etc) como lo sugieren Miglino, Lund y Nolfi [115]; u otra opción sería producir tablas de valores aleatorios (incluso con rangos de entrada de distancias, ángulos y anchos de pulso mayores) para cada paso generacional, cada prueba o intento de un robot, como lo describe Seth bajo el concepto “Inter-Trial noise” [136].

TABLA 4.1. TABLA DE VALORES DE LOS SENSORES

		DATA UTILIZADA POR LOS SENSORES ($I_{RA,REL}$)																			
grados mm.		-18	-16	-14	-12	-10	-8	-6	-4	-2	0	2	4	6	8	10	12	14	16	18	
5		0.07	0.20	0.36	0.59	0.80	0.87	0.93	0.96	0.97	1.00	0.98	0.96	0.92	0.88	0.81	0.60	0.35	0.22	0.07	
10		0.07	0.15	0.24	0.44	0.57	0.63	0.66	0.70	0.70	0.72	0.71	0.69	0.66	0.62	0.57	0.42	0.26	0.16	0.07	
15		0.07	0.08	0.15	0.23	0.33	0.36	0.39	0.41	0.41	0.42	0.41	0.40	0.38	0.36	0.34	0.25	0.14	0.11	0.07	
20		0.07	0.07	0.11	0.16	0.22	0.24	0.25	0.26	0.27	0.28	0.28	0.26	0.25	0.25	0.23	0.17	0.09	0.07	0.07	
25		0.07	0.07	0.08	0.13	0.16	0.17	0.18	0.19	0.20	0.20	0.19	0.18	0.18	0.17	0.17	0.13	0.07	0.07	0.07	
30		0.07	0.07	0.07	0.09	0.12	0.12	0.12	0.13	0.14	0.15	0.13	0.13	0.13	0.12	0.12	0.10	0.07	0.07	0.07	
35		0.07	0.07	0.07	0.07	0.09	0.10	0.10	0.10	0.10	0.11	0.11	0.11	0.10	0.09	0.08	0.07	0.07	0.07	0.07	
40		0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.09	0.09	0.09	0.09	0.09	0.08	0.08	0.07	0.07	0.07	0.07	0.07	
45		0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	
50		0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	
$I_{RA,REL}$: Fotocorriente Inversa Relativa																					

TABLA 4.2. TABLA DE VALORES DE LOS ACTUADORES

DATA UTILIZADA POR LOS MOTORES (RPM)							
	1300 μ s.	1320 μ s.	1340 μ s.	1360 μ s.	1380 μ s.	1400 μ s.	1420 μ s.
M1	53.2 SH	52.6 SH	51.7 SH	50.1 SH	42.5 SH	35.8 SH	28.1 SH
M2	49.5 SH	49.1 SH	48.3 SH	46.5 SH	40.8 SH	34.8 SH	26.7 SH
CONTINUACION DE LA TABLA							
	1440 μ s.	1460 μ s.	1480 μ s.	1500 μ s.	1520 μ s.	1540 s.	1560 μ s.
M1	20.7 SH	12.3 SH	6.2 SH	0	6.8 SA	12.8 SA	21.2 SA
M2	19.9 SH	12.8 SH	6.4 SH	0	5.8 SA	11.5 SA	19.2 SA
CONTINUACION DE LA TABLA							
	1580 μ s.	1600 μ s.	1620 μ s.	1640 μ s.	1660 μ s.	1680 μ s.	1700 μ s.
M1	29.3 SA	37.7 SA	44.0 SA	50.4 SA	52.1 SA	53.3 SA	53.7 SA
M2	26.4 SA	33.7 SA	39.8 SA	45.7 SA	47.9 SA	48.3 SA	48.8 SA

*SH/ SA: Sentido Horario/ Sentido Antihorario

En la simulación se representan 4 sensores CNY70 (anexo B), en 2 de ellos se calibra el umbral de detección tal que generen la salida lógica “0” para una superficie blanca y “1” para gris o negra, mientras en el resto la respuesta esperada es “0” para blanca o gris y “1” para negra, en el programa la posición cartesiana absoluta de cada sensor es calculada y se compara con la posición inicial o final del laberinto, determinándose así el estado lógico de cada sensor, luego este array de señales ingresa a una función lógica-numérica que entrega a la red neuronal una única señal triestado (0-blanco, 0.5 gris, 1-negro).

A fin de generar controladores robustos que puedan enfrentar diferentes condiciones y anticipándose a un posible traslado hacia la realidad [137], se aplicó el método de adición de ruido con respecto a la lectura de los sensores de distancia, la posición y orientación inicial del robot, su posición y orientación absoluta dentro del laberinto en cualquier instante; a los sensores de distancia se adicionó ruido a las entradas de sus respectivas tablas de valores, $\pm 2\text{mm}$. a la distancia y $\pm 1.6^\circ$ al ángulo del sensor respecto a la pared más cercana; en lo referente a la posición inicial se agregó $\pm 5\text{ mm}$. sobre las coordenadas “x” y “y” del punto medio del segmento que une los centros de ambas ruedas, mientras que la orientación inicial se halla en el rango de $90^\circ \pm 5^\circ$, asimismo sobre la posición y orientación instantánea del autómata se añadió un ruido de $\pm 5\%$ proporcional a los últimos desplazamientos lineal y angular del robot, en el caso de los sensores de color se les adicionó una función probabilística de ruido proporcional a la proximidad del dispositivo a una región de cambio de color, así la probabilidad de conmutación máxima es de 0.5 sobre la línea de cambio de color y la probabilidad de conmutación mínima es de 0.02 a una distancia mayor a 2 mm. (1 conmutación por cada 50 ciclos de simulación).

Por último en caso la simulación no se detenga cuando ocurran colisiones la posición de reposición del móvil después del choque es el punto central de la cuadrícula donde colisionó más una aleatoriedad de ± 10 mm. en ambas coordenadas “x” y “y”, mientras la orientación se mantiene antes y después del choque pero se agrega un ruido de $\pm 22.5^\circ$. De forma práctica el desarrollo algorítmico completo relacionado con la simulación física del autómata se encuentra documentado dentro del Anexo H.

4.5. Metodología Experimental

En resumen se realizarán 3 experiencias buscando ampliar el espectro de este estudio, en primer lugar se compararán funciones Fitness y determinará la más adecuada para el problema en cuestión; segundo, se decidirá la ubicación de los sensores de distancia en base a una evolución morfológica; finalmente se evaluarán distintos modelos neuronales con la intención de determinar cuales resultan más aptos para el surgimiento de conductas no reactivas.

4.6. Experiencia #1: Evaluación de Funciones Fitness

En este experimento el robot cuenta con cuatro épocas y un intento por cada una para procurar llegar a la meta en un laberinto predefinido de formato constante, el objetivo principal radica en llegar a la meta evitando colisionar con alguna pared. Para este fin se aplicará un Algoritmo Evolutivo, a la vez que se evaluarán distintas funciones Fitness de acuerdo a su eficacia.

4.6.1. Parámetros Experimentales

El ambiente empleado se observa en la figura 4.9, fue diseñado específicamente para esta experiencia, fue creado de manera que incluya múltiples dificultades, en primer lugar es lo suficientemente extenso para así evitar que controladores simples lo atravesen de manera fortuita, obligando por tanto al robot a buscar una estrategia robusta de evasión de obstáculos, adicionalmente presenta múltiples bifurcaciones y caminos sin salida que requieren de un Algoritmo Evolutivo capaz de explorar el ambiente entero, asimismo la posición de meta colinda con regiones fácilmente alcanzables desde la posición de partida, finalmente la distancia recorrida de principio a fin a través del camino directo no es la mayor posible dentro del entorno, estas dos últimas disposiciones tienen la capacidad de engañar a ciertas funciones Fitness relativamente simples.

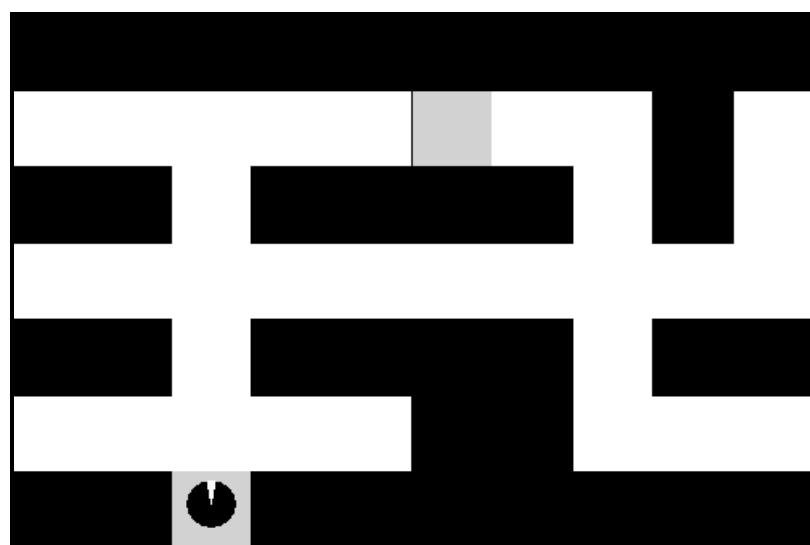


FIGURA 4.9. AMBIENTE PARA LA EXPERIENCIA #1

En esta experiencia se consideraron 6 funciones Fitness, el primer caso FF1 divide el ambiente según una malla formada por secciones cuadrangulares, de dimensiones ídem a las cuadriculas que componen el laberinto, FF1 resulta proporcional a la suma del número de regiones recorridas; la segunda función FF2 es calculada a partir de información directamente accesible al robot en el entorno real, combina 3 factores de navegación en la expresión $Fitness = 10 \cdot \bar{V} \cdot (1 - \sqrt{\Delta V}) \cdot (1 - S_{max})$ donde los términos \bar{V} , ΔV , S_{max} son promedios de la data obtenida en cada paso de simulación, el primero denota la rapidez angular promedio de ambas ruedas del móvil, el segundo describe el valor absoluto de la diferencia algebraica de velocidades y el último documenta el valor de activación del sensor más cercano a un obstáculo, todos estos términos se encuentran normalizados; FF3 cuantifica la distancia euclíadiana (d_e) entre la posición final del robot y la meta según $Fitness = 10 - d_e$; FF4 computa el desplazamiento ejecutado por el autómata, pero sin asignar ningún beneficio adicional por recorrer repetidamente las mismas zonas de manera innecesaria; hasta aquí toda simulación requería detenerse cuando el robot colisionase, en cambio la quinta función FF5 mantiene la ejecución del programa durante un tiempo concreto, además utiliza 2 parámetros de desempeño, uno siendo como en FF4 el desplazamiento total y el otro el número de colisiones cometidas por el móvil; por último FF6 resulta similar a FF5, sólo que adicionalmente al desplazamiento total y número de colisiones promedio de todas las épocas se agrega el valor del mejor resultado obtenido para cada factor, originando de esta manera una función Fitness compuesta en total por 4 parámetros.

En todos los casos cualquier autómata que alcance la meta será recompensado con una Fitness máxima de 10 metros (reemplaza sólo métricas de desplazamiento según corresponda), así todo controlador que complete el laberinto poseerá una medida de desempeño idéntica.

La estructura neuronal utilizada fue Perceptron simple con entradas motoras recurrentes (figura 4.10), modelo escogido por su simplicidad; existen 9 entradas normalizadas en el rango de 0 a 1 (5 sensores de distancia, 1 indicador de color, 2 motores, 1 bias); el controlador por su parte muestra 2 neuronas de salida con funciones sigmoide que luego son escaladas linealmente a fin de entregar señales de control PWM a los motores, en total existen 18 pesos sinápticos limitados todos dentro del rango [-5, 5]. El desarrollo algorítmico en detalle de esta experiencia se encuentra en el Anexo I.

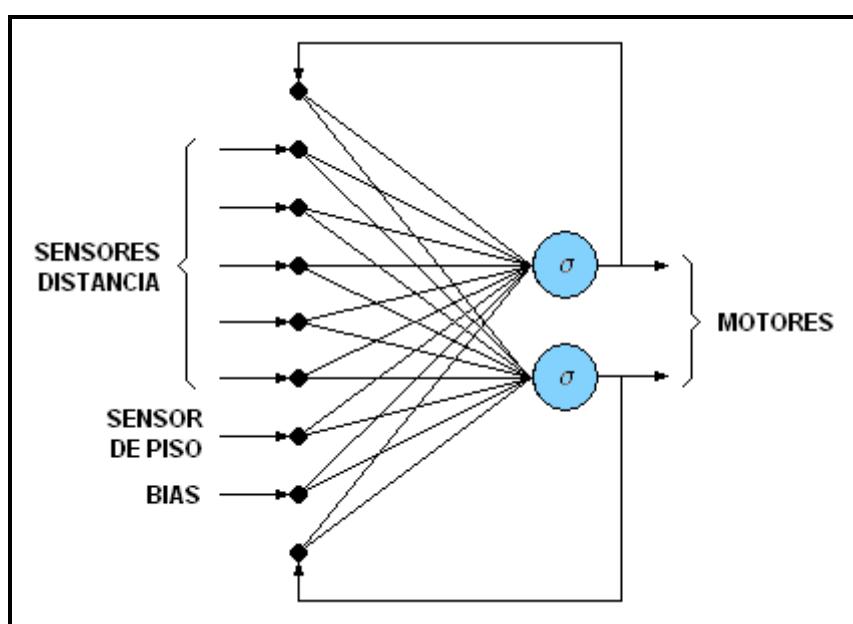


FIGURA 4.10. RED PERCEPTRON SIMPLE

Como Algoritmo Evolutivo fue empleado MOGA NSGA-II ejecutado durante 100 generaciones, la población presentó un total de 100 individuos, el genotipo manifestó una representación binaria de 8-bits, codificando los 18 pesos sinápticos; además se aplicó selección por Torneo Binario, Mating Pool de 20 individuos, Apareamiento sin restricciones, descendencia de 100 individuos, reinserción Elitista basada en Fitness, Crossover Uniforme con probabilidad de 0.7 y probabilidad de intercambio de 0.2, Mutación Estática Binaria mediante flipping con probabilidad de mutar de 0.9 por oportunidad y una tasa de mutación de 0.02 por bit.

4.6.2. Resultados

Cada función fue evaluada en 5 ocasiones, tiempo de simulación máximo de 100 seg. y paso de 0.1 seg., a continuación las poblaciones finales se reevaluaron en un mayor número de pruebas a fin de precisar una media estadística; la tabla 4.3 y figura 4.11 muestran los resultados obtenidos; como ya se explicó la Fitness máxima que entregan todas las funciones es 10 cuando se llega a la meta, caso contrario FF1 otorga un valor de 5.3 por explorar el laberinto en su totalidad, así en la Prueba #2 la cifra 7.3 indica que FF1 logró robots capaces de resolver el problema, ejecutando éstos una estrategia de seguimiento de pared (figura 4.11B), no obstante su efectividad media ronda el 70% demostrando robustez limitada, en las pruebas restantes las soluciones se estancaron en la región superior izquierda del laberinto realizando el mismo recorrido infructífero una y otra vez hasta concluir la simulación (figura 4.11A);

FF2 es una función continua en un rango entre 0 y 10, sin embargo valores elevados de hasta 8.4 no necesariamente denotan una adecuada resolución del problema, de hecho las soluciones de mayor Fitness que fueron obtenidas se mueven únicamente hacia adelante lo más rápidamente posible hasta colisionar con una pared perpendicular a su dirección de movimiento (figura 4.11C), esto se fundamenta en que una acción de giro del autómata frente a un posible choque significa una disminución de su Fitness, tanto así que cualquier robot que logre rotar, evitar la pared y vuelva a desplazarse hacia adelante registra solamente un desempeño máximo de 6.5, esto se debe a la reducción de los tres términos de la ecuación de navegación, \bar{V} disminuye gradualmente al acercarse a la pared para así evitar la colisión, ΔV aumenta durante el giro puesto que las velocidades de los motores se establecen en sentidos opuestos, para así permitir al móvil rotar prontamente, y S_{max} también se incrementa ya que el autómata se detiene y luego gira permaneciendo muy cerca de la pared, dicha rotación es detectada por tres sensores como mínimo; por otra parte la función FF3 fue largamente el peor candidato de entre las 6 funciones de desempeño, ningún controlador evolucionado bajo este método llegó a la meta del laberinto a pesar de la elevada Fitness registrada, esto último se precisa en base a que en prácticamente todos los casos los controladores cometieron el mismo error al colisionar directamente contra una pared colindante a la meta (figura 4.11D), hecho que les significa una ínfima distancia euclíadiana hasta la posición de llegada que a su vez FF3 corresponde con un desempeño sobresaliente y entrega Fitness muy cercanas al máximo posible, consiguiendo magnitudes superiores a 9.0;

las soluciones logradas gracias a la función de desempeño FF4 se asemejan a las generadas por FF1, en dos ocasiones (Pruebas #3 y #4) emergieron controladores poco robustos que atraviesan la totalidad del laberinto basándose de igual manera en una estrategia de seguimiento de pared, entretanto durante las oportunidades restantes la evolución sufre un estancamiento idéntico al ocurrido en FF1, donde el autómata recorre repetitivamente (ida y vuelta) la región superior izquierda del laberinto; las pruebas recolectadas a partir de FF5 presentan todas un patrón similar, en lo positivo en todas la pruebas surgen individuos que exploran el laberinto apropiadamente y como consecuencia arriban a la meta de forma recurrente, en lo negativo estos controladores resultan siempre medianamente robustos en lo concerniente a evitar colisiones, tal que solamente resuelven correctamente el problema de manera ocasional, es decir llegan sólo alrededor del 50% de la veces a la meta sin colisionar; estrictamente se puede afirmar que hasta este momento ninguna experiencia, ninguna función Fitness resultó satisfactoria y que todas terminaron por estancarse, puesto que no se hallaron soluciones suficientemente consistentes; finalmente la función FF6 se desempeñó claramente mejor que las demás alternativas y fue la única función que arribó a la solución deseada del problema, así de acuerdo a los resultados mostrados en la tabla 4.3 en las cinco oportunidades analizadas para FF6 se generaron ya en la generación final controladores muy robustos con más de 85% de recorridos completados en promedio poblacional, entretanto individualmente se observaron cifras superiores al 95% de efectividad.

TABLA 4.3. RESULTADOS DE LA EXPERIENCIA #1

		Fitness Promedio de la Población Final				
		Pruebas				
		#1	#2	#3	#4	#5
FF1	FPP (m.)	1.3744	7.3688	1.3522	1.3685	1.3789
	EPP (%)	0.0	69.3	0.0	0.0	0.0
FF2	FPP (m.)	8.0091	8.0452	7.2306	8.0946	7.9944
	EPP (%)	0.0	0.0	0.0	0.0	0.0
FF3	FPP (m.)	7.6154	8.5697	8.9891	9.0052	8.9124
	EPP (%)	0.0	0.0	0.0	0.0	0.0
FF4	FPP (m.)	1.2551	1.3145	6.1028	5.9825	1.3411
	EPP (%)	0.0	0.0	65.5	62.3	0.0
FF5	DRPP (m.)	7.4553	7.8188	8.2988	7.6525	7.9752
	NCPP	3.9475	0.6315	5.3185	2.4420	3.0664
	EPP (%)	45.7	57.9	53.5	48.6	52.8
FF6	DRPP (m.)	9.6905	9.2569	9.9200	9.6023	9.5442
	RMPP (m.)	9.9296	9.7101	10.000	9.8505	9.7726
	NCPP	1.4240	2.4605	0.1852	1.2512	2.0204
	CMPP	0.8500	0.7300	0.0000	0.6200	0.7100
	EPP (%)	84.7	70.3	85.6	85.1	75.5

-FPP: Fitness Promedio por Población
-EPP*: Efectividad Promedio por Población
-DRPP: Distancia Recorrida Promedio por Población
-NCPP: Número de Colisiones Promedio por Población
-RMPP: Recorrido Máximo Promedio por Población
-CMPP: Colisiones Mínimas Promedio por Población

*EPP: Computa únicamente el número de ocasiones en que los controladores en la generación final resolvieron correctamente el problema, en porcentaje sobre el total de simulaciones realizadas.

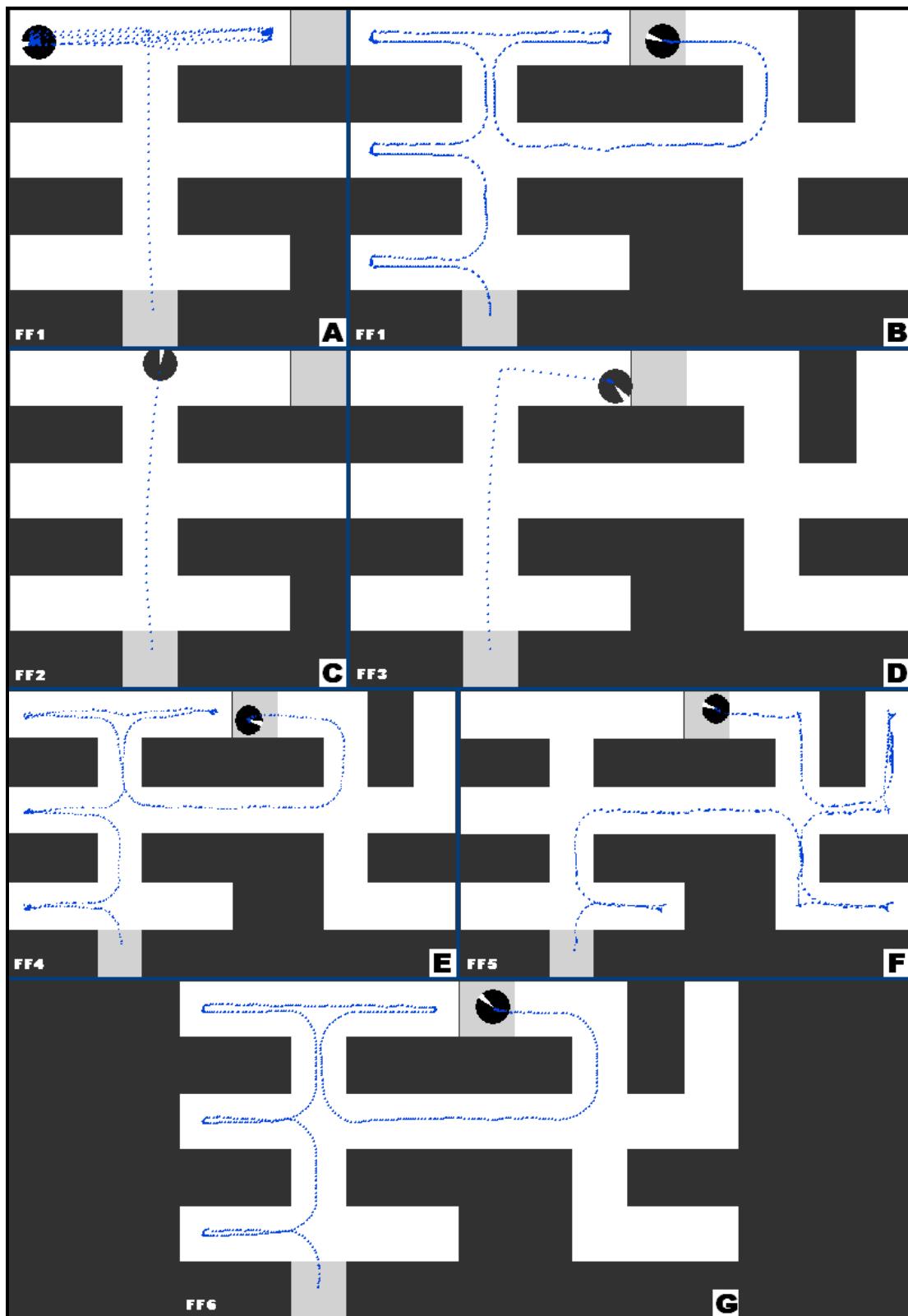


FIGURA 4.11. CONDUCTAS FINALES DE LA EXPERIENCIA #1

4.6.3. Análisis de Resultados

FF1 y FF4 otorgan una Fitness basada en la distancia recorrida por el robot, pero difieren en que FF1 asigna valores discretos mientras FF4 continuos, dada la semejanza de resultados entre ambas funciones se realizó 20 pruebas extras por cada una y se recolectó data de la Fitness media poblacional (figura 4.12); aquí se observa una apreciable similitud entre ambas curvas, al inicio cuando la evolución favorece a soluciones que sólo se desplazan hacia delante, las dos funciones exhiben igual velocidad de convergencia y se estabilizan en la misma Fitness (aprox. 0.5); acto seguido cuando la evolución se detiene hasta descubrir individuos con aptitudes de evasión de obstáculos, FF1 y FF4 escapan del estancamiento en forma simultánea alrededor de la generación #25; existe otra detención cuando el robot permanece atrapado en la región superior izquierda del laberinto, el instante en que la evolución emerge de tal paralización coincide con puntos de inflexión en las gráficas, si bien FF4 se sobrepone antes que FF1, debe advertirse que ambas funciones superaron la detención sólo en 3 de 20 pruebas, por tanto no existe evidencia estadística que refleje una diferencia relevante, en cambio se puede hallar una regularidad comparable en cuanto se refiere a la competencia del GA para resolver el problema en base a FF1 y FF4; en el tramo final las dos curvas presentan también una tasa de convergencia equivalente. De este análisis se extiende que a pesar del mayor esfuerzo requerido en la implementación de una función Fitness continua, ésta no genera los beneficios sustanciales esperados, así en este caso particular su empleo resulta hasta cierto punto ineficiente.

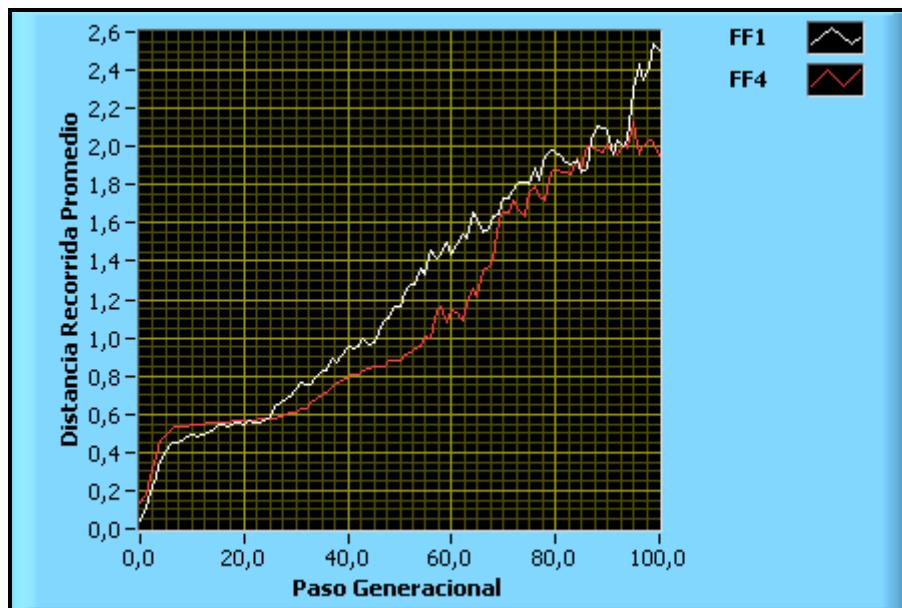


FIGURA 4.12. FF1 VS. FF4: FITNESS MEDIA POBLACIONAL

La idea detrás de agregar parámetros Fitness máximos o mínimos en FF6 parte de la intuición de que cuando una conducta novedosa surge, ésta no se replicará imperiosamente en todo intento ejecutado por el individuo, por tanto si utilizamos únicamente el promedio global como en FF5 corremos el riesgo de que estas nuevas habilidades no se distingan fácilmente entre la población y por ende no logren difundirse, así bajo este supuesto FF6 debería mejorar la rapidez de convergencia vista en FF5; pero si se observa la evolución en las gráficas 4.13 y 4.14 notamos que al inicio no ocurre lo esperado, FF5 converge igual o más rápido que FF6; dicho fenómeno se puede explicar si consideramos que en un comienzo no existen retos evolutivos complejos que hagan uso de conductas innovadoras, perdiendo éstas la importancia antes conferida, además frente a este evento los factores adicionales sólo generan mayor diversidad poblacional y por tanto retrasan la convergencia;

pero esto sólo ocurre en las etapas iniciales hasta un instante en el cual FF6 pasa a liderar sobre FF5, de acuerdo a suposiciones previas dicha situación acontecería cuando los individuos requieran una habilidad nueva e imprescindible, la cual evolucionase más fácilmente a partir de FF6; un análisis de la gráfica 4.15, la cual registra la distancia recorrida por los 20 individuos que colisionan en menor medida, sugiere que se trataría de la evasión activa de obstáculos; al examinar el progreso evolutivo tanto para FF5 como para FF6 se advierte que los individuos manifiestan en el comienzo una de dos tendencias, un comportamiento “explorativo” en el que el robot recorre grandes distancias pero colisiona repetidamente y otro “evasivo” donde los individuos permanecen en una región reducida del espacio pero evitan colisión alguna; con el paso de las generaciones ambas tendencias son optimizadas, no obstante la evolución de aptitudes explorativas debido a su facilidad de emergencia, replicabilidad y/o a la naturaleza de los factores de desempeño muestra una predominancia natural bajo ambas funciones Fitness; la gráfica 4.15 señala además que el desarrollo de conductas evasivas es ampliamente más lento en FF5 que en FF6, estas dos condiciones originan en el caso de FF5 que los individuos exploradores adopten inicialmente estrategias de evasión mediocres; en seguida debido a que el factor discreto del número de colisiones limita la diversidad fenotípica, sucede que a medida que dichas soluciones incrementen su complejidad, completen el laberinto frecuentemente y sobretodo reduzcan levemente la cantidad de colisiones cometidas, elevarán en simultáneo su probabilidad de pertenecer conjuntamente al mismo grupo cero de dominancia Pareto y en definitiva acrecentarán su tasa de reproducción,

entretanto los individuos evasores encuentran una realidad más desfavorable, ya que aún cuando ejecuten comportamientos idénticos las diferencias de décimas o centésimas de milímetros en la distancia recorrida los categorizarán en grupos Pareto distintos, finalmente todo esto conlleva a una merma sustancial en el número de soluciones de características evasivas presentes en la población y consecuentemente su convergencia se estancará, la optimización del número de colisiones queda ahora en cuenta del comportamiento explorativo, situación que presenta baja presión evolutiva, pues las soluciones se desarrollarán hasta lograr culminar satisfactoriamente la prueba de manera ocasional, luego su rendimiento se incrementará paulatinamente hasta copar totalmente la población, en ese momento las soluciones se sustituirán unas a otras sin necesidad de exhibir mejora alguna en cuanto a rendimiento, peor aún una alta eficacia no evita que cualquier individuo sea eliminado arbitrariamente de la población, así la convergencia del algoritmo se desacelera llegado un nivel específico de rendimiento que en la curva de FF5 (figura 4.13) se observa aproximadamente en 75%; la situación de FF6 es similar a FF5 en cuanto a la tendencia de comportamientos explorativos a imponerse sobre los comportamientos evasivos, no obstante la incorporación del parámetro de mínimo número de colisiones por intento evita la rápida extinción de la conducta evasiva al intensificar su velocidad de convergencia inicial, tal y como permite inferirse de la gráfica 4.15; asimismo ya desde el paso #5 se contempla como FF6 empieza a marcar diferencias respecto a FF5, los individuos con conductas evasivas exploran más activamente el ambiente, esto les ayuda a desarrollar estrategias de evasión más robustas prontamente;

dichas soluciones originan posteriormente híbridos que combinan en su código genético los mejores rasgos exploratorios y evasivos, en la misma gráfica la pendiente pronunciada que se forma pasada la generación #20 revela el inicio de dicho proceso; mientras que, por el contrario la evolución mediante FF5 evidencia un estancamiento a partir de ese mismo instante; gradualmente tales híbridos monopolizan la población, prosiguiendo de este modo el proceso de optimización comenzado por las dos tendencias, y gracias a los operadores de mutación y crossover eventualmente descubren la mejor manera de vincular provechosamente ambas características.

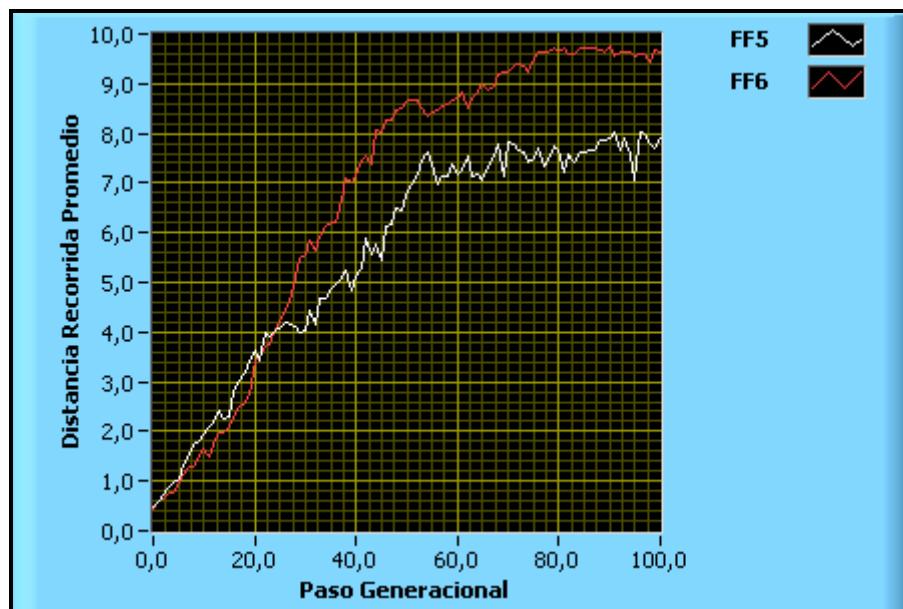


FIGURA 4.13. FF5 VS. FF6: DISTANCIA RECORRIDA

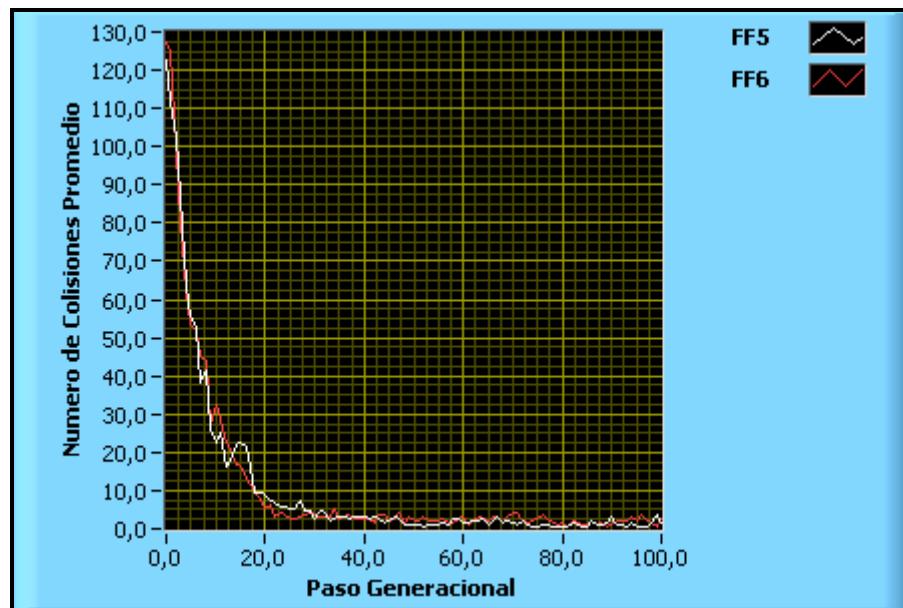


FIGURA 4.14. FF5 VS. FF6: NUMERO DE COLISIONES

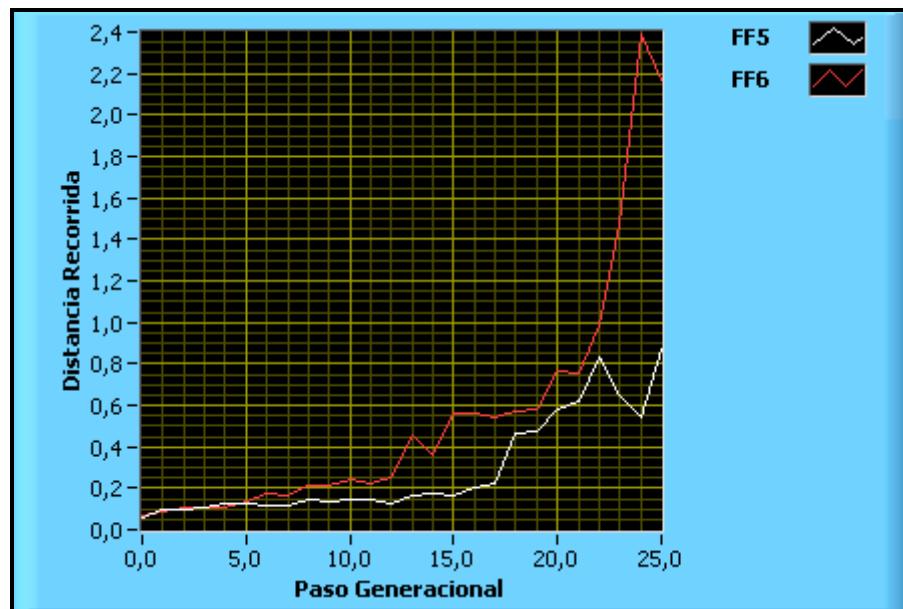


FIGURA 4.15. FF5 VS. FF6: DISTANCIA RECORRIDA POR LOS
20 INDIVIDUOS CON MENOR NÚMERO DE COLISIONES

El propósito de esta primera etapa fue demostrar que el rendimiento de un Algoritmo Genético varía respecto de la Función Fitness empleada por el mismo, todos los resultados conseguidos constatan fehacientemente esta realidad, al tanto que señalan que una mala elección de la Función de Desempeño conduce a la no resolución del problema o en el mejor de los casos a mayores tiempos de simulación. Asimismo se evidenció la subjetividad inherente al momento de definir una Función Fitness; para un problema específico, incluso uno sencillo como el de esta experiencia, se pueden considerar múltiples alternativas y resulta muy complicado determinar de antemano la más óptima entre ellas; los análisis previos precisaron al estancamiento en mínimos locales como el mayor inconveniente en contra de la evolución, dicha situación ocurre cuando un comportamiento particular es lo suficientemente eficiente durante una instancia de la evolución, al punto que termina acaparando la población, estos comportamientos en general no aceptan mayores avances evolutivos y tampoco permiten a otras conductas desarrollarse, deteniéndose así el proceso evolutivo; de hecho todas las funciones evaluadas a excepción de FF6 revelan dicho impasse de una u otra forma, FF1, FF3 y FF4 ya informaron de esta condición, FF2 también se detiene debido a un comportamiento particular que no obstante maximiza su ecuación de navegación, e inclusive FF5 se estanca antes de descubrir controladores con mayor robustez; actualmente los Algoritmos Evolutivos intentan prever esta circunstancia y proponen la adición de parámetros de desempeño relativos a la diversidad o novedad genotípica, fenotípica o conductual, obteniéndose resultados diversos.

4.7. Experiencia #2: Evolución Morfológica

En este experimento el robot cuenta con dos épocas y un único intento por cada una para procurar llegar a la meta en un laberinto tipo T, dicha posición final varía aleatoriamente entre épocas, el autómata es permitido de atravesar el laberinto durante la totalidad del tiempo de simulación, aún si chocase con algún obstáculo la prueba no se detiene a excepción que se concluya satisfactoriamente el intento, el objetivo principal es simplemente alcanzar la meta sin colisionar con alguna pared. Asimismo se implementará la evolución de la configuración posicional de los sensores de distancia alrededor del robot, la noción subyacente es que distribuciones idóneas no mostrarán dificultades para detectar y evitar paredes, mientras que configuraciones inadecuadas presentarán puntos ciegos y demás complicaciones imposibles de remediar bajo ninguna estrategia de navegación.

4.7.1. Parámetros Experimentales

La función Fitness ofrece dos parámetros de evaluación al igual que FF5 en la Experiencia #1, el primer componente calcula la longitud del recorrido ejecutado por el robot; mientras el segundo factor computa los choques sucedidos durante la evaluación. Se usó Perceptron con entradas recurrentes (figura 4.10); la red neuronal exhibe 9 entradas normalizadas en el rango [0,1] (5 sensores de distancia, 1 indicador de color, 2 motores, 1 bias); el controlador muestra 2 neuronas de salida con función sigmoide, en total existen 18 pesos sinápticos limitados todos en un rango de [-5,5].

Como EA se empleó NSGA-II ejecutado durante 250 generaciones, una población de 100 individuos, genotipo binario de 8-bits; se crean 2 cadenas cromosómáticas independientes (figura 4.17), una codifica los 18 pesos sinápticos y la otra 5 ángulos comprendidos entre $[-\pi, +\pi]$ (medidos en relación a un eje perpendicular al eje común de las ruedas), que revelan la ubicación de los sensores de distancia (figura 4.16); se aplicó selección por Torneo Binario, Mating Pool de 20 individuos, Apareamiento sin restricciones, Offspring de 100 individuos, reinserción Elitista basada en Fitness, Crossover Uniforme con probabilidad de 0.7 y probabilidad de intercambio de 0.2, Mutación Binaria por flipping con probabilidad de 0.9 y tasa de mutación de 0.02 por bit. El desarrollo algorítmico en detalle de esta experiencia se encuentra en el Anexo J.

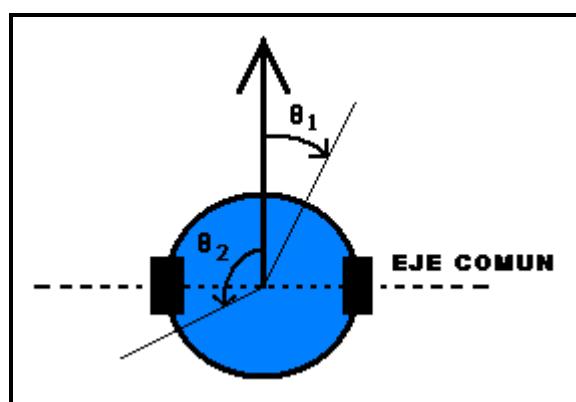


FIGURA 4.16. DISPOSICION SENSORIAL

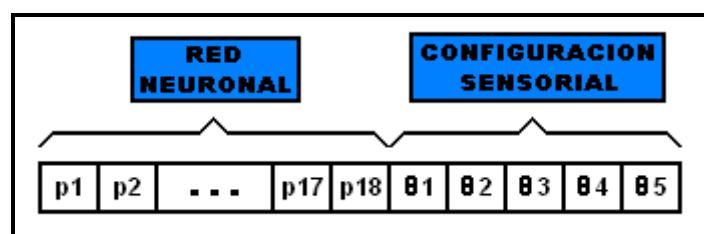


FIGURA 4.17. ORGANIZACIÓN CROMOSOMATICA

4.7.2. Resultados

Se realizaron 5 pruebas con un tiempo de simulación de 40 s. y paso de 0.1 s., los individuos de las generaciones finales de todas las pruebas se reevaluaron en 1000 recorridos extras a fin de precisar su robustez, la tabla 4.4 muestra las mejores soluciones de cada evaluación; se observa al individuo más robusto en la Prueba #4, éste incurre en sólo una colisión por cada 100 recorridos (alrededor de 99% de efectividad), presenta una distribución que reúne los 5 sensores en un cuadrante, con un sensor apuntando hacia el frente; Pruebas #1, #2 y #3 describen una configuración similar excepto por un sensor ubicado a 30° de la posición frontal en el cuadrante opuesto; en la Prueba #5 la disposición hallada exhibe 2 sensores frontales, otros 2 en el cuadrante derecho y el restante en el cuadrante izquierdo de la zona posterior (ver gráfica 4.18). Suplementariamente la gráfica 4.19 visualiza la trayectoria habitual que sigue el controlador mejor adaptado (Prueba #4).

TABLA 4.4. MEJORES CONFIGURACIONES SENSORIALES

		Pruebas				
		#1	#2	#3	#4	#5
SD1	rad.	1.219	-0.012	1.293	0.110	-0.874
	Grados°	69.88	-0.705	74.11	6.35	-50.11
SD2	rad.	1.490	0.406	1.219	-1.170	-2.500
	Grados°	85.41	23.29	69.88	-67.06	143.29
SD3	rad.	-0.012	-1.318	-0.554	-0.283	-1.268
	Grados°	-0.705	-75.53	-31.76	-16.23	-72.70
SD4	rad.	1.268	-0.726	0.455	-0.628	0.135
	Grados°	72.70	-41.64	26.11	-35.71	7.76
SD5	rad.	-0.579	-0.923	-0.086	-0.948	-0.110
	Grados°	-32.71	-52.94	-4.94	-54.35	-6.35
Tasa de Colisiones		0.025	0.225	0.075	0.011	0.062

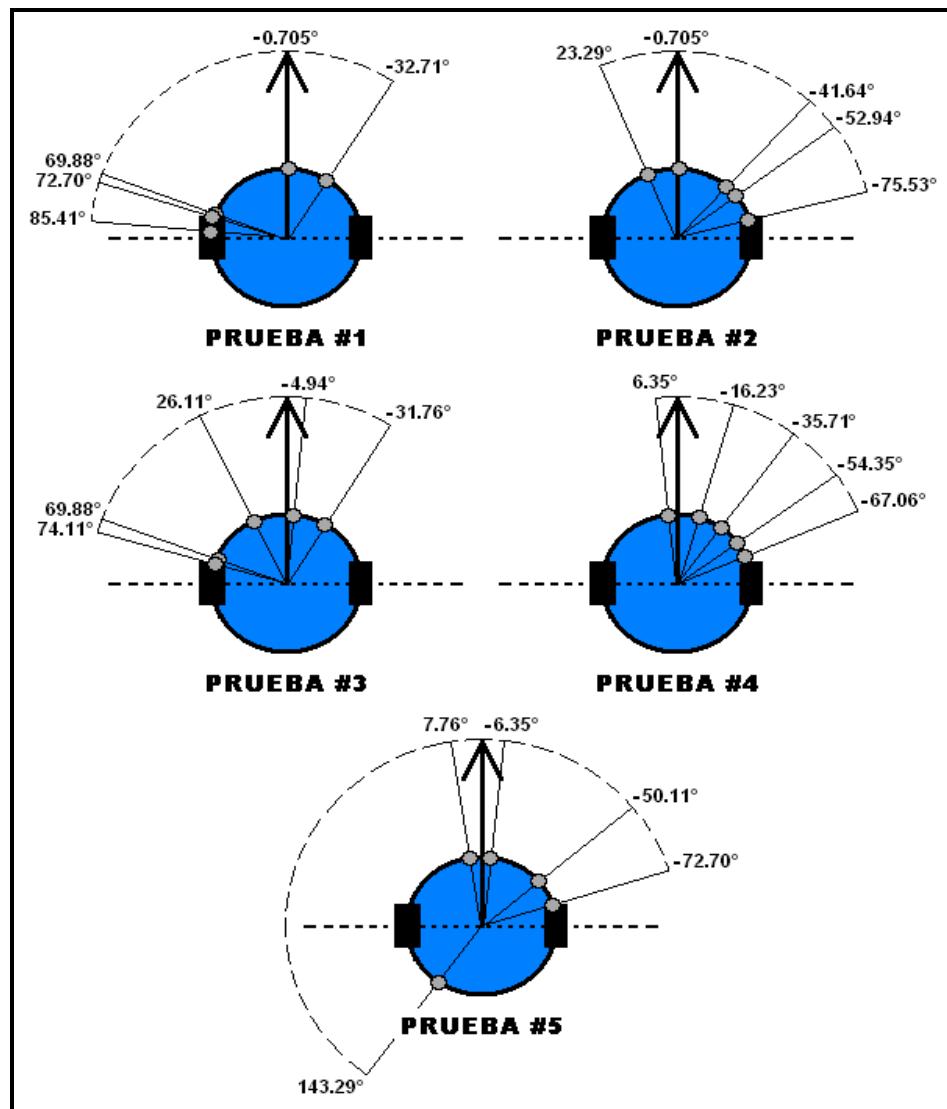


FIGURA 4.18. CONFIGURACIONES SENSORIALES RESULTANTES

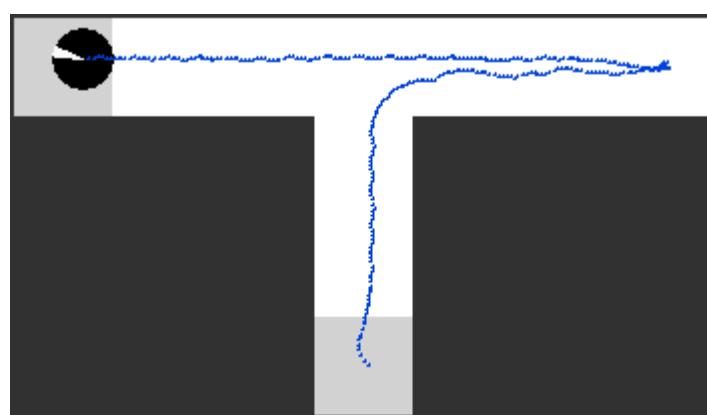


FIGURA 4.19. RECORRIDO DEL CONTROLADOR #4

A continuación se procedió a comparar la configuración sensorial más prometedora (Prueba #4) frente a otros arreglos predeterminados (ver tabla 4.5), los cuales probablemente hubieran sido manejados en caso de obviar la evolución morfológica; el algoritmo aplicado en esta etapa conservó los mismos parámetros evolutivos citados a priori, salvo dos excepciones, en primer lugar el genotipo obviamente se modificó eliminando los componentes referidos a las posiciones sensoriales, adicionalmente se redujo el límite de generaciones a 100 pasos. Cada morfología fue evaluada en un total de 5 oportunidades, registrándose la distancia recorrida y el número de colisiones promedio por población en cada paso generacional; el cuadro 4.6 revela los promedios obtenidos en la generación final de cada ensayo; Conf_Elegida (seleccionada a partir de la evolución morfológica) cumplió un desempeño remarcable, logró resultados de Distancia Recorrida Promedio (DRPP) de 9.4 sobre un límite de 10 y Número de Colisiones Promedio (NCPP) de 0.8 por recorrido, cifras muy superiores a las alcanzadas por las distribuciones Conf_A, Conf_B y Conf_D; entretanto el arreglo predeterminado más sobresaliente fue Conf_C, cuyo diseño demuestra una distribución sensorial compacta con un amplio rango de detección, Conf_C revela factores de desempeño DRPP = 9.2 y NCPP = 1.47 que se asemejan a aquellos conseguidos por Conf_Elegida; tal es la suficiencia de Conf_Elegida que a partir de ella (Prueba #2) también evolucionó el controlador más robusto cuya efectividad bordea el 93%, le sigue de cerca Conf_C (Prueba #5) con el 90.5%, mientras que la máxima efectividad en las configuraciones restantes fue de 49.9% obtenida con Conf_B (Prueba #2).

TABLA 4.5. CONFIGURACIONES SENSORIALES PREDETERMINADAS

	Configuración Sensorial (grados°)				
	SD1	SD2	SD3	SD4	SD5
Conf_Elegida	6.35	-67.06	-16.23	-35.71	-54.35
Conf_A	-90	-45	0	45	90
Conf_B	-90	-30	0	30	90
Conf_C	-75	-37.5	0	37.5	75
Conf_D	-30	-15	0	15	30

TABLA 4.6. COMPARACION DE ARREGLOS SENSORIALES

		Resultados de la Generación Final					Prom.	
		Pruebas						
		#1	#2	#3	#4	#5		
Conf_Eleg.	DRPP(m.)	9.90	9.95	9.53	9.44	8.32	9.428	
	NCPP	0.24	0.19	1.17	1.13	1.49	0.844	
	ECMR(%)	89.4	92.8	59.1	58.5	46.2	69.2	
Conf_A	DRPP(m.)	7.62	3.49	5.95	4.46	7.57	5.818	
	NCPP	2.29	4.60	4.07	3.51	1.79	3.252	
	ECMR(%)	40.3	23.5	42.0	35.7	44.4	37.20	
Conf_B	DRPP(m.)	6.93	5.95	4.38	3.83	3.90	4.998	
	NCPP	1.95	4.98	6.58	2.97	3.10	3.916	
	ECMR(%)	39.8	49.9	24.2	19.1	10.1	28.62	
Conf_C	DRPP(m.)	9.13	9.38	9.00	8.75	9.76	9.204	
	NCPP	1.83	0.41	2.05	2.86	0.20	1.470	
	ECMR(%)	49.9	85.3	55.6	50.0	90.5	66.26	
Conf_D	DRPP(m.)	3.43	6.07	4.51	2.26	5.12	4.278	
	NCPP	2.66	8.42	8.65	2.72	4.55	5.400	
	ECMR(%)	16.1	0.0	5.8	23.8	4.2	9.980	

4.7.3. Análisis de Resultados

Las gráficas 4.20 y 4.21 se elaboraron en base al promedio de los resultados registrados en cada prueba, contradictoriamente demuestran una leve superioridad de Conf_C sobre Conf_Elegida; la primera figura expone el desarrollo de la Distancia Recorrida Promedio a través del tiempo, aquí tanto Conf_Elegida como Conf_C sobrepasan apreciablemente a las distribuciones restantes; inicialmente ambas curvas son idénticas hasta llegado un punto donde Conf_Elegida detiene su evolución mientras Conf_C mantiene una convergencia estable, eventualmente Conf_Elegida escapa del estancamiento y equipara a Conf_C sobre el término de la experiencia; es importante mencionar que las pruebas de Conf_C son consistentes entre sí, mientras que las de Conf_Elegida fluctúan y solamente una de las cinco sufrió de estancamiento, en tanto el resto de pruebas evidenció un comportamiento ideal libre de este inconveniente; la siguiente gráfica devela la evolución progresiva del Número de Colisiones, en este caso resulta nuevamente obvio que Conf_Elegida y Conf_C detentan un desempeño apropiado, convergiendo rápida y establemente hasta una instancia cercana a la ideal, mientras las demás configuraciones muestran una convergencia más lenta, además aún hacia el final de la experiencia incurren en una elevada cantidad de colisiones, como consecuencia de ello se generan pocos individuos capaces de resolver el problema satisfactoriamente.

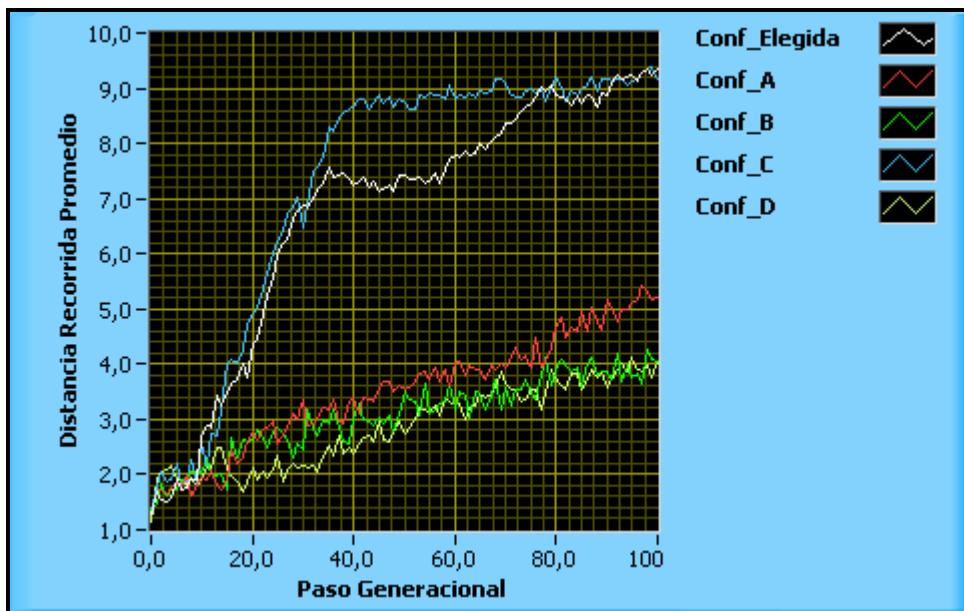


FIGURA 4.20. DRPP EN DISTINTOS ARREGLOS SENSORIALES

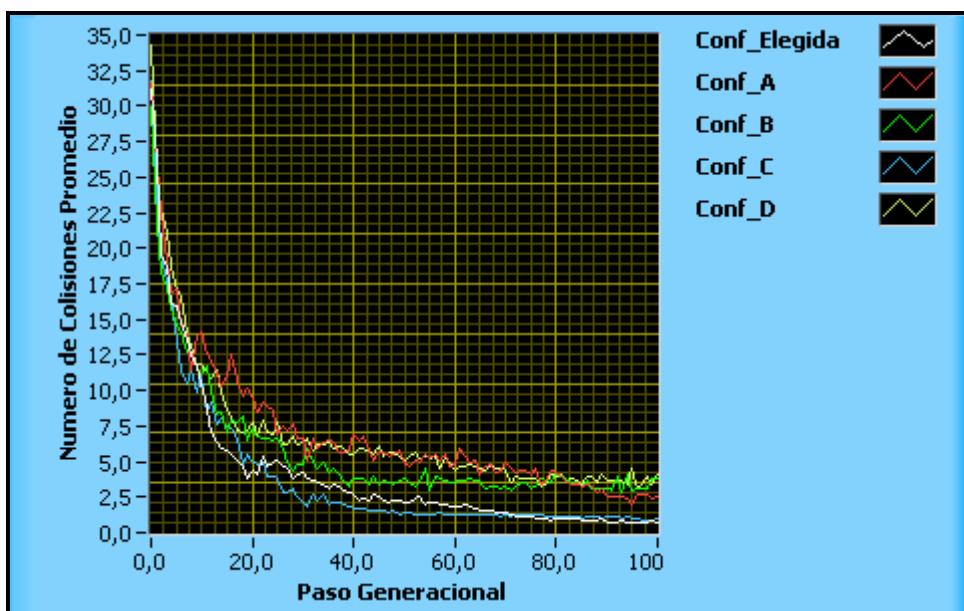


FIGURA 4.21. NCPP EN DISTINTOS ARREGLOS SENSORIALES

Los resultados obtenidos brindan gran cantidad de información presta a mayor análisis, primero demostraron que ciertas distribuciones sensoriales son más adecuadas que otras en este problema particular, sea por la rapidez de convergencia, consistencia de las evaluaciones o en general por su capacidad de descubrir una solución; la primera parte de esta experiencia probó la suficiencia de una evolución morfológica para hallar configuraciones sensoriales altamente útiles al problema específico, alcanzando incluso un impresionante 99% de efectividad; entretanto aún siendo el problema relativamente simple y familiar se consiguieron soluciones diversas y algunas difícilmente pronosticables, esta noción trasladada a situaciones de mayor complejidad y con menor disponibilidad de información hace de la evolución una herramienta muy deseable frente a la pérdida de tiempo que suele ocurrir al experimentar con múltiples alternativas; finalmente el mejor controlador obtenido resultó también el más pragmático, ya que conservó todos sus sensores repartidos en un sólo cuadrante, siendo de vital importancia el sensor al frente del robot, que evita colisiones con obstáculos frontales; por otro lado el mejor controlador de la Prueba #1 presentó dos sensores prácticamente en una misma ubicación (69.8° y 72.7°), infiriendo la posibilidad de prescindir de un componente, dicha realidad en proyectos de mayor envergadura permitiría reducir costos sustanciales. La segunda fase del experimento confirmó claramente que la configuración seleccionada a partir de la evolución morfológica se encuentra a la altura de las expectativas, manifestó una convergencia rápida y siempre se hallaron controladores que resolvieron el problema, al tiempo que ninguna distribución sensorial alternativa logró resultados superiores.

4.8. Experiencia #3: Evaluación de Redes Neuronales

En este experimento el robot cuenta con 16 épocas y 2 intentos por cada una para procurar llegar a la meta en un laberinto tipo T, dicha posición final varía aleatoriamente por época; el autómata es permitido de atravesar el laberinto durante la totalidad del tiempo de simulación, la prueba no se detiene a menos que se concluya satisfactoriamente el intento; el objetivo final es alcanzar la meta sin colisionar con pared alguna, pero con el detalle adicional que solamente durante el primer intento de cada época puede el robot explorar completamente el laberinto, mientras que en los intentos subsiguientes se espera que recorra únicamente el “camino correcto”. De esta manera el experimento requiere en general de controladores que manifiesten habilidades No-Reactivas, en particular el reconocimiento de patrones temporales; con este propósito se examinará la evolución de distintas estructuras neuronales a fin de verificar su idoneidad para cumplir con los requerimientos del problema.

4.8.1. Parámetros Experimentales

El entorno consiste en un laberinto T de dimensión reducida (figura 4.22), empleado a fin de acortar el tiempo de ejecución del algoritmo.



FIGURA 4.22. LABERINTO T REDUCIDO

La función Fitness empleada ofrece 5 parámetros de evaluación: distancia recorrida promedio (32 intentos), distancia recorrida máxima (entre intentos), número de colisiones promedio (32 intentos), número de colisiones mínimo (entre intentos) y cantidad de “caminos correctos” elegidos (16 intentos); al igual que en experiencias anteriores los dos primeros componentes calculan el recorrido global ejecutado por el robot, el primero asume el promedio de todos los intentos realizados y el segundo considera sólo el valor máximo, igualmente sucede con los parámetros tercero y cuarto que computan los choques acontecidos; por último el quinto factor precisa las veces que encontrándose el robot ante una bifurcación escoge el “camino correcto”, no obstante se obvia el primer intento de cada época, concediéndole así al robot la posibilidad de explorar el laberinto sin menoscabar su desempeño global.

Las estructuras usadas son Perceptron simple con entradas recurrentes (PS), Perceptron Multicapa con entradas recurrentes (PM), Redes de Funciones Radiales con entradas recurrentes (RBF), redes FRNNs, CTRNNs, ESNs, ESNs con leak rate (α), SNNs (Modelo SRM) y SNNs con STDP (spike-timing-dependant plasticity), dichos modelos fueron escogidos por su preponderancia y competencia en el campo de la Robótica Evolutiva; entretanto todas las entradas se normalizaron en el rango [0, 1], excepto para las redes ESNs y SNNs, la realimentación en las ESNs opera entre [-1, 1], mientras que en las SNNs las señales inicialmente normalizadas se transforman en secuencias de impulsos antes de ingresar al control; el modelo PS muestra 9 señales de entrada (5 sensores de distancia, 1 indicador de color, 2 motores y 1 bias),

además muestra 2 neuronas de salida con función sigmoide que luego serán escaladas linealmente a fin de accionar los motores, de esta manera se despliegan 18 pesos sinápticos limitados dentro de un rango [-5, 5]; la red PM ostenta igual configuración de entradas y salidas, mas cuenta adicionalmente con una capa oculta de 5 neuronas (función sigmoide), asimismo la entrada bias se conecta completamente tanto a la capa oculta como a la capa de salida, resultando en un total de 57 conexiones sinápticas todas ellas acotadas entre [-5, 5]; análogamente las redes RBF conservan las 9 señales de entrada mencionadas con anterioridad, adicionalmente esta arquitectura manifiesta 5 neuronas dentro de su capa oculta, cada neurona oculta a su vez exhibe 9 parámetros variables relativos a una función gaussiana estándar

$$y = e^{-\frac{(\|x - c_i\|)^2}{b_i^2}}$$

, de los cuales 8 factores componen el vector centro (c_i) de la i-ésima neurona, estos 8 elementos se circunscriben dentro del rango [0, 1] en cuenta que los valores manejados por las entradas se encuentran normalizados, el parámetro restante (b_i) corresponde al factor de amplitud de la función gaussiana y presenta sólo valores comprendidos entre [0.05, 2.5] de acuerdo al estándar comúnmente empleado, finalmente la capa de salida contiene dos neuronas quienes reciben la activación de la capa oculta y del bias a través de pesos sinápticos acotados dentro de la región [-5, 5], las neuronas de salida se encuentran provistas de funciones de transferencia lineales que además normalizan el resultado de la combinación lineal, de este modo la red evidencia un agregado final de 57 parámetros partícipes de la evolución;

a diferencia de los modelos previos las redes FRNNs se concibieron completamente conectadas, reciben 6 entradas relativas solamente a los sensores, además manifiestan 5 neuronas en la capa oculta y 2 en la capa de salida, todas las neuronas poseen una función de activación sigmoide, finalmente son obtenidas 91 conexiones sinápticas; las redes CTRNNs también se conectan completamente, manejan únicamente 6 entradas pertenecientes a sensores, asimismo cuentan con 5 neuronas en la capa oculta y 2 en la capa de salida, toda neurona usa la misma

$$\text{ecuación dinámica } \frac{dy_i}{dt} = \frac{1}{\tau_i} \left(-y_i + \sum_{j=1}^N w_{ji} \cdot \sigma(y_j + \beta_j) + \sum_{k=1}^S w_{ik} \cdot I_k \right),$$

que se resuelve aplicando el método de Euler con paso de 0.1s., cada neurona presenta una constante de tiempo τ_i en el rango [1,50]s., un bias β_i comprendido entre [-1,1] y 13 pesos sinápticos circunscritos dentro de [-5,5], agrupando así 105 parámetros factibles de evolucionar, la salida de la red es dirigida hacia una función sigmoide a fin de acotar la misma entre [0, 1], posteriormente este valor se escala linealmente; las ESNs adquieren 9 entradas (6 de sensores, 2 realimentadas desde los motores y bias), muestran un reservorio compuesto por 100 neuronas y su salida consta de 2 neuronas, todas ellas cuentan con activación tangente hiperbólica, la creación de la red estima un radio espectral (spectral radius) de 0.9, la conectividad del reservorio es de 0.1 con pesos de valor 1, mientras la conectividad Entradas-Reservorio y Entradas-Salida es de 0.3 con pesos fijos de 0.25, en tanto el reservorio se encuentra completamente conectado a las salidas y son estas 200 conexiones las únicas viables de optimizar en el rango [-5, 5],

la red ESN sigue las siguientes ecuaciones de actualización de estado

$$\mathbf{x}(t+1) = \tanh(\mathbf{W}_{res}^{res} \cdot \mathbf{x}(t) + \mathbf{W}_{in}^{res} \cdot \mathbf{u}(t)) \text{ donde } \mathbf{x}(t) \text{ describe el estado}$$

del reservorio y $\mathbf{u}(t)$ denota las entradas, mientras \mathbf{W}^{res} son matrices

$$\mathbf{y}(t+1) = \tanh(\mathbf{W}_{res}^{out} \cdot \mathbf{x}(t+1) + \mathbf{W}_{in}^{out} \cdot \mathbf{u}(t)) \text{ en donde } \mathbf{y}(t)$$

referencia a las salidas y \mathbf{W}^{out} los pesos de las conexiones hacia las mismas; las ESNs con leak rate (α) demuestran exactamente la misma configuración que las ESNs normales, encontrándose la única diferencia en su ecuación de estado del reservorio, en donde es adicionado un parámetro también evolucionable (leak rate = α) acotado en la región [0, 1], presentando en definitiva la siguiente expresión de estado

$$\mathbf{x}(t+1) = \tanh((1 - \alpha) \cdot \mathbf{x}(t) + \alpha \cdot (\mathbf{W}_{res}^{res} \cdot \mathbf{x}(t) + \mathbf{W}_{in}^{res} \cdot \mathbf{u}(t))); \text{ las redes}$$

SNNs también fueron diseñadas completamente conectadas, el controlador es actualizado cada milisegundo, mientras que las señales de sensores y actuadores se renuevan cada 100 ms., la capa de entrada dispone de los 6 sensores más una entrada bias, sus lecturas son previamente transformadas en impulsos con una probabilidad de ocurrencia proporcional a la magnitud normalizada percibida por los sensores en cada actualización temporal del controlador, esta estructura adicionalmente manifiesta 6 neuronas en la capa oculta y 4 en la capa de salida, de estas últimas dos neuronas gobiernan las velocidades positivas de cada motor y las dos neuronas restantes controlan las velocidades negativas, el valor real obtenido a partir de estas neuronas resulta proporcional al número de disparos efectuados por la neurona respectiva durante los 20 ms. anteriores a la actualización del actuador,

la representación genética de esta estructura dispone por cada neurona de un bit que señala su carácter excitatorio o inhibitorio (una neurona excitatoria origina conexiones postsinápticas con pesos positivos, mientras que una inhibitoria implanta pesos negativos, asimismo todas las entradas se estiman con carácter excitatorio), adicionalmente se acondiciona un bit por cada sinapsis dentro de la red que indica si dicha posible conexión existe o permanece inactiva, en total se enumeran 180 bits en el cromosoma de cada individuo; la fórmula del potencial de membrana en cada neurona es: $v_i(t) = \sum_j w_{ij} \sum_f \varepsilon_j(s_j) + \sum_f \eta_i(s_i)$, donde $\varepsilon(s) = e^{-[(s-\Delta)/\tau_m]} \cdot (1 - e^{-[(s-\Delta)/\tau_s]})$ y $\eta(s) = -r \cdot e^{-(s/\tau_m)}$, w_{ij} manifiesta el peso de la conexión cuya magnitud es $w=1$ en todos los casos, entretanto su signo es dependiente del carácter excitatorio o inhibitorio de la neurona presináptica, s_n indica la diferencia temporal entre el instante actual y el tiempo del último disparo efectuado por la neurona “n”, en referencia a $\varepsilon(s)$ si su argumento se encontrase fuera del rango $\Delta \leq s \leq 20 \text{ ms}$, entonces el resultado de la función se asume como nulo, situación similar sucede con $\eta(s)$ al transponer los límites de la región acotada entre $0 < s \leq 20 \text{ ms}$, $\Delta = 2 \text{ ms}$ se considera como un retraso ocasionado durante la transmisión del impulso, $\tau_s = 10 \text{ ms}$ es una constante de tiempo sináptico, $\tau_m = 4 \text{ ms}$ constante de tiempo de membrana, y el factor r es un valor uniformemente aleatorio en el rango $[0, 1]$; todas las neuronas presentan un umbral $\theta = 0.1$, si dicho umbral fuese excedido por el potencial de membrana, entonces dicha neurona emitirá un impulso individual en aquel instante específico;

redes SNNs y el método STDP parten del modelo neuronal previamente descrito, sin embargo sus pesos no conservan el valor unitario, tal que todo w_{ij} pertenece al rango $[0, 1]$, los pesos no son evolucionables y se generan aleatoriamente al inicio de cada simulación, asimismo dichos valores no permanecen fijos durante la simulación sino que pueden ser modificados en cada paso temporal según una regla de aprendizaje, la misma que se define mediante cinco parámetros diferentes para cada sinapsis, dicha regla de aprendizaje se rige conforme a las siguientes ecuaciones $w_{ij}(t+1) = w_{ij}(t) + \Delta_d w_{ij}(t)$, donde $\Delta_d w_{ij}$ responde a la variación del peso sináptico con un amortiguamiento o damping direccional expresado por $\Delta_d w_{ij}(t) = (1 - w_{ij}(t)) \cdot \Delta w_{ij}(t)$, $\forall \Delta w_{ij} \geq 0$ y $\Delta_d w_{ij}(t) = w_{ij}(t) \cdot \Delta w_{ij}(t)$, $\forall \Delta w_{ij} < 0$, además $\Delta w_{ij}(t) = \gamma \cdot \zeta_{ij}(t)$, aquí γ es el factor de aprendizaje comprendido entre $[0.001, 0.5]$, mientras ζ_{ij} es una notación definida como $\zeta_{ij}(t) = P_{ij}^+(t) \cdot f_i(t) + P_{ij}^-(t) \cdot f_j(t)$, donde P_{ij}^+ y P_{ij}^- son variables que registran la influencia histórica de los impulsos presinápticos f_i y postsinápticos f_j respectivamente, a su vez su formulación se denota por $P_{ij}^+(t) = P_{ij}^+(t-1) \cdot e^{-\frac{\delta t}{\tau^+}} + A^+ \cdot f_j(t)$ y $P_{ij}^-(t) = P_{ij}^-(t-1) \cdot e^{-\frac{\delta t}{\tau^-}} - A^- \cdot f_i(t)$, en dichas ecuaciones $\delta t = 1 \text{ ms}$, $\tau^+ \wedge \tau^-$ están acotados entre $[1, 40] \text{ ms}$. y $A^+ \wedge A^-$ son parámetros adimensionales en el rango $[0.001, 2]$, completándose finalmente un total de 850 parámetros reales y 180 binarios que requieren ser optimizados.

Como Algoritmo Evolutivo fue empleado un MOGA NSGA-II ejecutado durante 300 generaciones, la población presentó un total de 100 individuos, sus genotipos manifestaron una representación binaria de 8-bits, excepto en el caso de las redes SNNs donde ciertos parámetros son intrínsecamente binarios y no requieren de encriptación adicional; además fue aplicada selección por Torneo Binario, Mating Pool de 20 individuos, Apareamiento sin restricciones, Crossover Uniforme con probabilidad de 0.7 y probabilidad de intercambio de 0.2, Mutación Estática Binaria mediante flipping con probabilidad de 0.9 y tasa de mutación de 0.02 por bit. El desarrollo algorítmico en detalle de esta experiencia se encuentra en el Anexo K.

4.8.2. Resultados

El algoritmo se evaluó 20 veces para cada arquitectura, con un tiempo de simulación máximo por individuo de 10 seg. en cada intento, un paso de 0.1 segundos igual para todos los casos excepto para las redes SNNs cuyo controlador se simuló cada 1 ms., posteriormente la generación final fue reevaluada en 100 pruebas adicionales a fin de determinar la eficacia de las mejores soluciones alcanzadas, la tabla 4.7 muestra los resultados obtenidos por cada estructura neuronal, dicha información señala que sólo los modelos basados en FRNN, CTRNN, ESN+ α y STDP consiguieron evolucionar capacidades No-Reactivas o memorísticas; según se ve en dicha tabla únicamente los modelos CTRNN y STDP lograron 100% de efectividad en lo que respecta a la suficiencia para evolucionar neurocontroladores que exhiban memoria,

sin embargo STDP no concretó una optimización satisfactoria, tal que su Fitness Promedio no llegó nunca a converger y el mejor controlador presentó sólo 87% de efectividad, muy por debajo del 100% expuesto por los otros modelos, esto se debe en gran medida a que la evolución del método STDP es más lenta, ya que incluso después de 300 generaciones resulta difícil encontrar una estrategia de navegación aceptable bajo dicho modelo; FRNN y ESN+ α obtienen excelentes resultados finales provisto que no se estanquen en una evolución carente de controladores No-Reactivos, adicionalmente en ambos casos los individuos finales son aproximadamente 5 veces más veloces que las soluciones generadas por el modelo CTRNN.

TABLA 4.7. COMPARACION DE ESTRUCTURAS NEURONALES

	Resultado de la Evolución de distintas Arquitecturas		
	Fitness Promedio	ECMR(%)	Eficacia de Decisión
PS	-	50	0
PM	-	50	0
RBF	-	50	0
FRNN	15.693	100	15
CTRNN	15.866	100	20
ESN	-	50	0
ESN+α	15.813	100	18
SNN	-	50	0
STDP	7.73	87	20

-Fitness Promedio: Media poblacional del factor de “caminos correctos” obtenida en la generación final, promediándose las 20 ejecuciones.
 -ECMR(%): Porcentaje de efectividad para resolver el problema completa y satisfactoriamente, extraído luego de efectuar 100 pruebas a la mejor solución hallada a través de las 20 evaluaciones.
 -Eficacia de Decisión: Número de pruebas que desarrollaron soluciones con aptitudes memorísticas sobre la base de las 20 evaluaciones.

4.8.3. Análisis de Resultados

La gráfica 4.23 exhibe la evolución de la Memoria Promedio Poblacional en un ensayo representativo de las arquitecturas que manifestaron sólo comportamientos Reactivos, estos individuos toman el mismo camino siempre frente a una bifurcación, por ello en la mitad de las ocasiones resolverán el problema correctamente, obteniendo así un valor medio de Memoria de 8 respecto del máximo de 16; entretanto se observa que PS, el cual posee el cromosoma de tamaño más reducido, converge más rápidamente alrededor del paso #20, a su vez la red ESN con un cromosoma considerablemente mayor demora evidentemente más que el resto, alcanzando la convergencia cerca de la generación #150, mientras tanto PM, RBF y SNN muestran una característica similar convergiendo entre el 50vo y 70vo paso evolutivo.

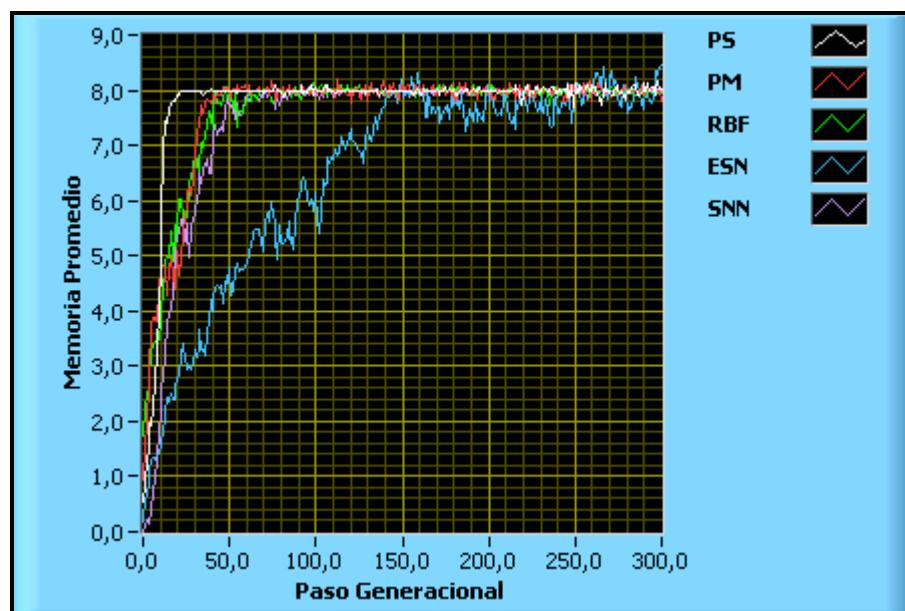


FIGURA 4.23. EVOLUCIÓN DE CONDUCTAS REACTIVAS

La gráfica 4.24 provee información similar al caso previo, pero ésta refiere a los modelos que obtuvieron comportamientos No-Reactivos; la Red FRNN evolucionó la conducta Reactiva de manera rápida, luego permaneció estancada por más de 100 generaciones, para finalmente desarrollar capacidades memorísticas a partir del 150vo paso evolutivo, en general este modelo al inicio no demuestra signos de evolucionar memoria, por lo cual la conducta Reactiva copa la población, se puede conjeturar que si bien existe en su esencia la capacidad de desarrollar memoria, ésta no ocurre frecuentemente o se encuentra en regiones muy limitadas o difíciles de acceder en su espacio genotípico, de esta forma el algoritmo se detiene hasta que en algún momento de manera fortuita una mutación conduzca a la solución ideal, dicha situación se lograría contrarrestar modificando el Algoritmo Evolutivo de modo que promueva la diversidad de especies o que intensifique la exploración del espacio de búsqueda de soluciones, tal y como sucede en los métodos NEAT, CMA-ES o EANT2; la red CTRNN evidenció controladores con memoria desde muy temprano, sin embargo sufrió de un estancamiento inicial notable debido principalmente a la competencia entre 2 grupos de individuos disímiles, el primero formado por soluciones con prominentes habilidades de navegación a la vez carentes de memoria, al contrario el segundo compuesto por individuos con destacable aptitud de memoria pero con una estrategia de navegación inadecuada, ni los unos eran capaces de evolucionar memoria, ni los otros lograban optimizar su navegación, por ello no fue sino hasta la generación #100 cuando emergieron híbridos quienes reiniciaron la convergencia del algoritmo;

la red ESN con leak rate manifestó un desempeño ideal, evolucionó de forma prácticamente lineal convergiendo velozmente en el paso #120, en este caso a medida que la población perfeccionaba la navegación algunos individuos además iban incorporando habilidades de memoria, de esta manera confluyeron gradualmente en un mismo controlador ambas características; finalmente el modelo STDP demostró el peor resultado en este grupo, inclusive no llegó a converger siquiera al nivel de una conducta Reactiva, en sí padece del mismo inconveniente que la red CTRNN, pero con el agravante que su cromosoma contiene 8 veces el número de parámetros que la red antes mencionada, con lo cual la evolución tiende a retardarse proporcionalmente, en una prueba complementaria se mantuvo ejecutando el algoritmo por mayor tiempo, descubriendo convergencia en el paso #736 tras un estancamiento previo de aproximadamente 500 generaciones.

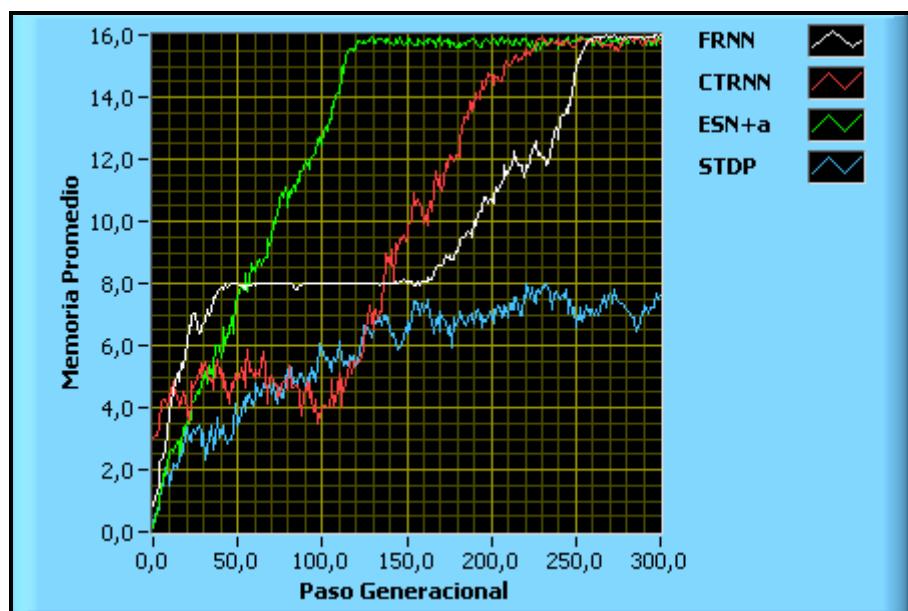


FIGURA 4.24. EVOLUCIÓN DE CONDUCTAS NO-REACTIVAS

El experimento realizado pretendió incentivar la evolución de controladores que sensen el tiempo transcurrido; de manera que si alcanzasen la meta rápidamente, debieran mantener la ruta elegida a través del segundo recorrido, en cambio si demorasen pasado un tiempo predefinido correspondería tomar el camino opuesto; a fin de precisar si los controladores hicieron uso de esta estrategia, fueron testeadas las soluciones más eficaces obtenidas bajo las distintas estructuras neuronales ante tiempos de simulación y ambientes modificados, la figura 4.25 retrata el comportamiento de un controlador de arquitectura CTRNN frente a un tiempo máximo de simulación variable durante el primer recorrido (las redes y métodos FRNN, ESN+ α , STDP expusieron resultados similares), dicha figura revela que a pesar de realizar comportamientos diferentes y por ende recibir lecturas de sensores y actuadores discordantes con lo habitual, los individuos preservan una resolución apropiada del problema vinculada estrechamente al paso de simulación vigente; por ejemplo para este controlador particular cuando es permitido menos de 110 pasos de simulación durante el primer recorrido, luego en un segundo intento el robot tiende a tomar el camino derecho, en cambio si el tiempo supera los 130 pasos, entonces el autómata elige el camino izquierdo; esto ocurre además al margen de cualquier información proveniente de alguna marca ambiental puesto que los recorridos seguidos en el intento inicial son completamente distintos entre sí, de esta forma se demuestra que los controladores evolucionados son capaces de tomar decisiones en base a pistas temporales y no solamente ambientales.

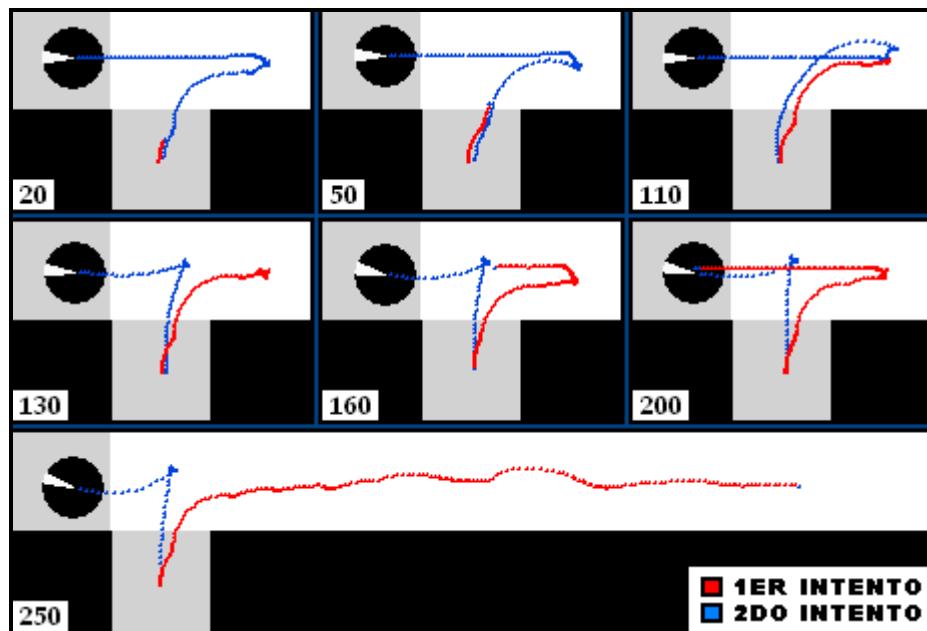


FIGURA 4.25. TIEMPO DE SIMULACION Y AMBIENTE VARIABLES

Adicionalmente para un controlador de red CTRNN evolucionado satisfactoriamente se elaboraron las gráficas 4.27, 4.28, 4.29, 4.30, que referencian al estado de activación de toda neurona y sensor cuando el robot ejecuta una conducta “alternativa” o “regular” (figura 4.26), y las tablas 4.8, 4.9 que registran los pesos sinápticos de la red; esta data señala que en ciertos casos los autómatas actúan independientemente de la influencia sensorial, ya que las curvas de estado de las neuronas H3 y H5 prácticamente no se ven afectadas por variaciones sensoriales, mientras que H1 sólo inicialmente decrementa de forma exponencial en respuesta al sensor de piso en la partida, incluso H2 y H4 que muestran las mayores alteraciones poseen constantes de tiempo muy pequeñas, reduciendo la influencia a largo plazo de los sensores, así se tiene una dinámica interna que actúa aún ante la ausencia de estimulación sensorial y facilita la consecución de comportamientos No-Reactivos;

para descifrar como la dinámica interna reconoce patrones temporales y hace uso de ellos, se debe notar primero que ante la ausencia de estímulos externos (sin considerar al indicador de color, puesto que la sinapsis entre M2 y SF resulta ínfima), la activación de M2, relativa al motor izquierdo, converge abruptamente desde cualquier valor en que se encontrase hasta establecerse alrededor de 0.7, valor que depende más precisamente de la diferencia entre H2 y H4 en el instante que desapareció la actividad sensorial, así en razón de la explícita inferencia de la dinámica interna sobre el comportamiento ilustrado se deduce la naturaleza no-reactiva de éste; otro hecho a considerar sucede ya en el segundo recorrido del robot, desde que éste recién ingresa a la partida, ya la actividad sensorial decae notablemente durante un periodo de tiempo, lo que posibilita a M2 alcanzar su estado estable; un tercer punto a observar es la conducta de la señal M1 vinculada al motor derecho, ésta mantiene un lento crecimiento monótono exponencial a través del tiempo que converge aproximadamente en 0.75 más allá del ciclo generacional #200, de igual forma no presenta alteraciones significativas frente a las variaciones sensoriales o en todo caso los sensores no son artífices de aquel comportamiento estable, ya que únicamente SF muestra una sinapsis positiva de magnitud importante y sin embargo SF no justifica el aumento permanente visualizado en la curva de estado de la neurona M1, en debida cuenta que sólo actúa en instantes de tiempo específicos y no de forma continua, por tanto la activación de M1 depende fundamentalmente de la dinámica interna, particularmente del incremento en las señales H2 y H3, el cual a su vez se mantiene autónomo respecto de cualquier posible influencia externa,

el funcionamiento de la red se explica en base a los conceptos referidos previamente, valorando M1 como una señal de control que en instantes iniciales ofrece una activación baja, luego en tiempos ulteriores entrega un nivel alto, entretanto M2 se asume como una señal umbral; así en el primer intento del robot se observa una diferencia considerable entre M2 y M1 cuya magnitud promedia 0.5, ello origina una rotación horaria que conduce al autómata hacia el camino derecho; a continuación si el robot alcanza la meta rápidamente M1 conservará aún un nivel bajo y la desigualdad se mantendrá, generando así la “conducta regular”; sobre la “conducta alternativa”, el mayor tiempo sucedido permite acrecentar en magnitud a M1 hasta bordear el umbral propuesto por M2, frente a dicha condición el robot revela un desplazamiento casi rectilíneo que se detiene en las cercanías de la pared superior del laberinto cuando SD3 y SD4 emiten pulsos de detección, lo cual basado en el comportamiento reactivo del controlador suscita una inversión en la velocidad del motor izquierdo, seguida de un giro antihorario del móvil (la figura 4.26 permite apreciar mejor ambas conductas). En definitiva, en este caso paradigmático, la evolución produjo una señal sincronizada con el tiempo de simulación, la cual fue utilizada luego como parámetro de una función umbral para así determinar la ejecución de una u otra conducta.

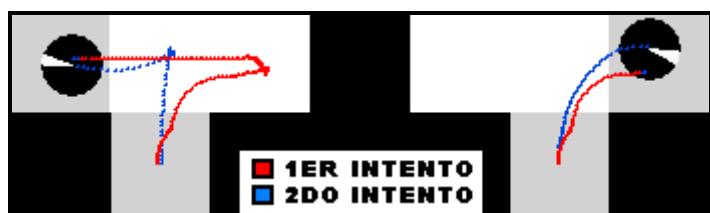


FIGURA 4.26. CONDUCTAS ALTERNATIVA Y REGULAR

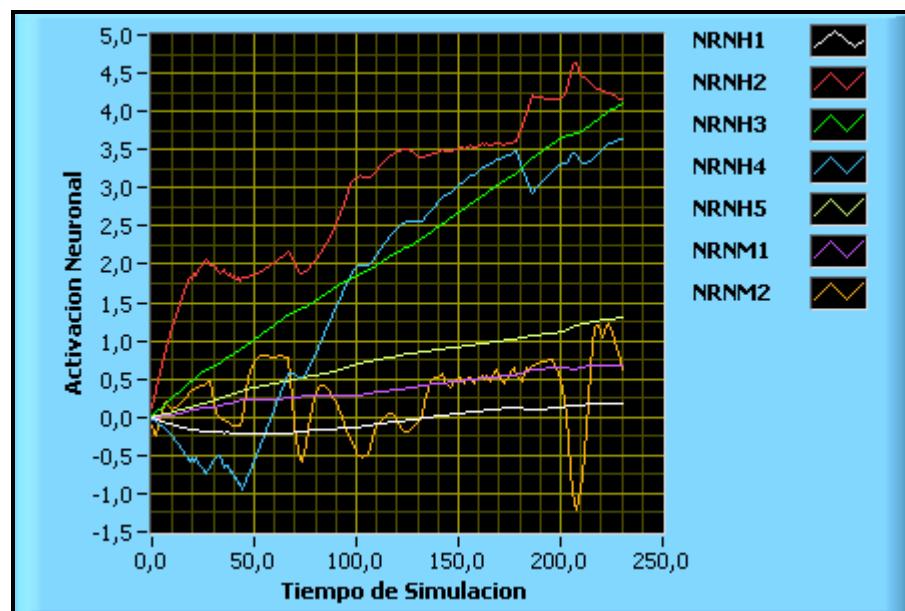


FIGURA 4.27. CONDUCTA ALTERNATIVA: ACTIVACION NEURONAL

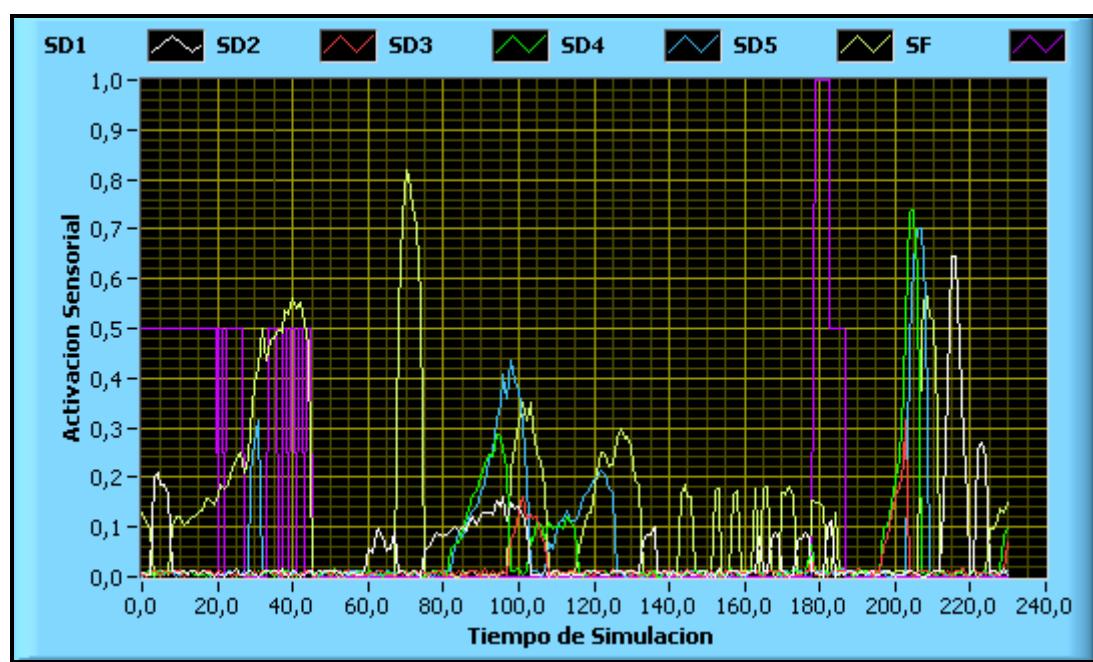


FIGURA 4.28. CONDUCTA ALTERNATIVA: ACTIVACION SENSORIAL

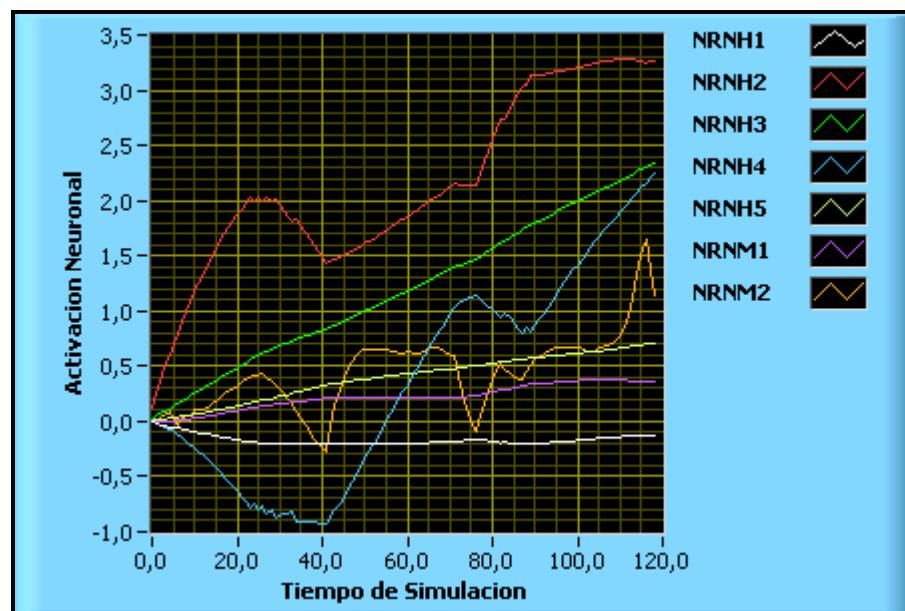


FIGURA 4.29. CONDUCTA REGULAR: ACTIVACION NEURONAL

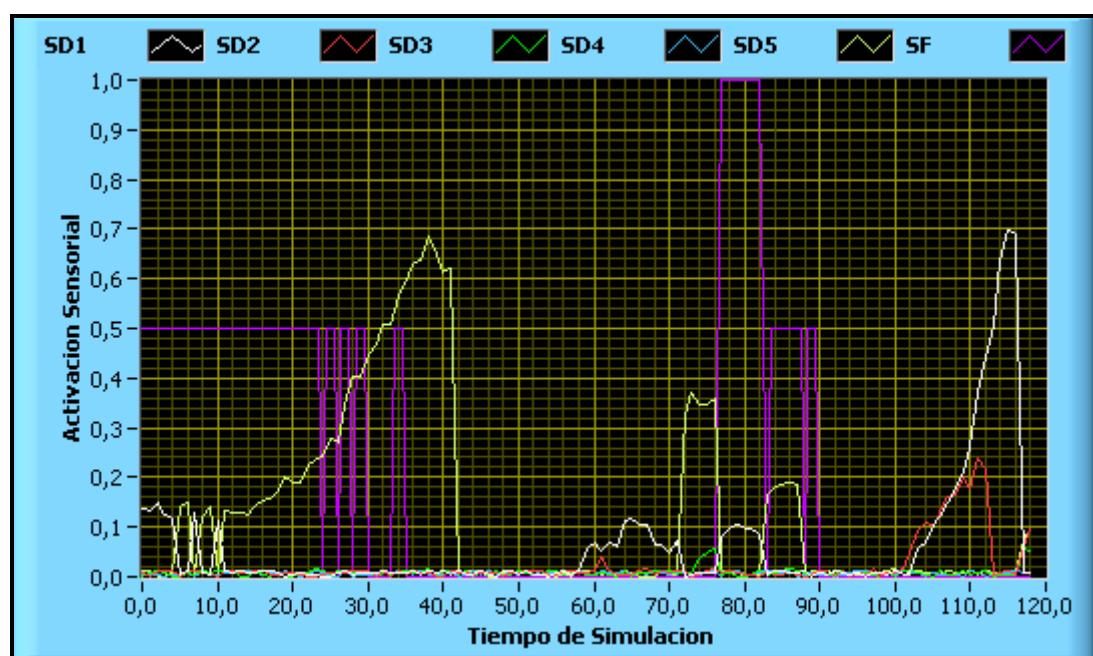


FIGURA 4.30. CONDUCTA REGULAR: ACTIVACION SENSORIAL

TABLA 4.8. MATRIZ DE PESOS

	Conexiones Sinápticas						
	H1	H2	H3	H4	H5	M1	M2
H1	-2,92157	-3,27451	3,58824	0,41176	4,13725	-0,72549	0,01960
H2	2,80392	-1,19608	2,33333	-2,92157	2,52941	5,00000	3,62745
H3	1,23529	4,13725	3,39216	1,23529	-0,76470	2,84314	0,33333
H4	3,15686	2,13725	-0,52941	4,60784	-3,70588	-1,54902	-4,05882
H5	-3,78431	1,47059	4,49020	-0,25490	3,03922	-2,25490	0,64705
M1	-1,74510	-1,70588	1,74510	0,49019	1,43137	-1,03922	4,41176
M2	-1,50980	-0,21568	3,27451	2,80392	-0,33333	-3,94118	-3,86275
SD1	-1,70588	-1,19608	2,21569	-0,45098	2,72549	-0,64705	4,76471
SD2	0,52941	0,25490	-4,52941	-0,68627	0,09803	-4,80392	-2,68627
SD3	1,11765	2,84314	-3,62745	-1,19608	4,88235	-0,92156	-4,56863
SD4	-3,03922	4,52941	-3,90196	3,86275	2,45098	-3,31373	-3,7451
SD5	1,00000	-3,82353	-3,82353	-4,68627	1,74510	0,52941	-4,13725
SF	-3,23529	2,92157	3,39216	-4,33333	0,41176	1,70588	0,05882

H.n: n-ésima neurona oculta.
M.n: n-ésima neurona motora.
SD.n: n-ésimo sensor de distancia.
SF: Indicador de color de piso.

TABLA 4.9. MATRIZ DE PARAMETROS DINAMICOS

	Parámetros de la Función de Activación Dinámica	
	Constante de Tiempo (seg.)	Bias
H1	42,5059	-0,898039
H2	3,69020	-0,780392
H3	47,6941	-0,372549
H4	5,03529	0,772549
H5	43,6588	0,364706
M1	28,4784	-0,341176
M2	1,00000	-0,623529

Para concluir, en base a la información develada es posible afirmar que ciertos modelos neuronales resultan más convenientes que otros cuando se requiere recabar pistas temporales y utilizarlas en la toma de decisiones, los resultados señalaron como las estructuras más prometedoras a las redes ESN+ α , CTRNN y FRNN esta última provista de un Algoritmo Evolutivo mejorado; el método STDP manifestó una convergencia muy lenta, peor aún los tiempos de ejecución del algoritmo fueron excesivos llegando a consumir muchos días y hasta semanas, por ello actualmente las aplicaciones más productivas de SDTP se dan en casos de evolución online en el mismo hardware.

4.9. Validación de los Resultados

En esta sección se comprobó el funcionamiento de los neurocontroladores obtenidos mediante evolución artificial bajo un ambiente de simulación distinto al modelo propio implementado en Labview. Para ello se utilizó el software reconocido y comercial Simulink/Simscape/SimMechanics/ SimElectronics desarrollado por Mathworks, Inc.

4.9.1. Simulación del ambiente virtual y agente robótico

Todas las consideraciones mecánicas se programaron directa y exclusivamente en SimMechanics, donde a partir de un sistema de coordenadas fijo, se procedió a agregar los componentes que forman el robot mediante articulaciones y restricciones físicas, además por cada componente se definió una masa y por ende un momento de inercia

particular. En el caso particular del robot, se definió además su centro de masa (CM) como una geometría sin masa, así respecto a este se referenciaron las demás partes del móvil; los diagramas de bloques obtenidos dentro del entorno Simulink se muestran a continuación tanto para el ambiente virtual como para el robot propiamente dicho.

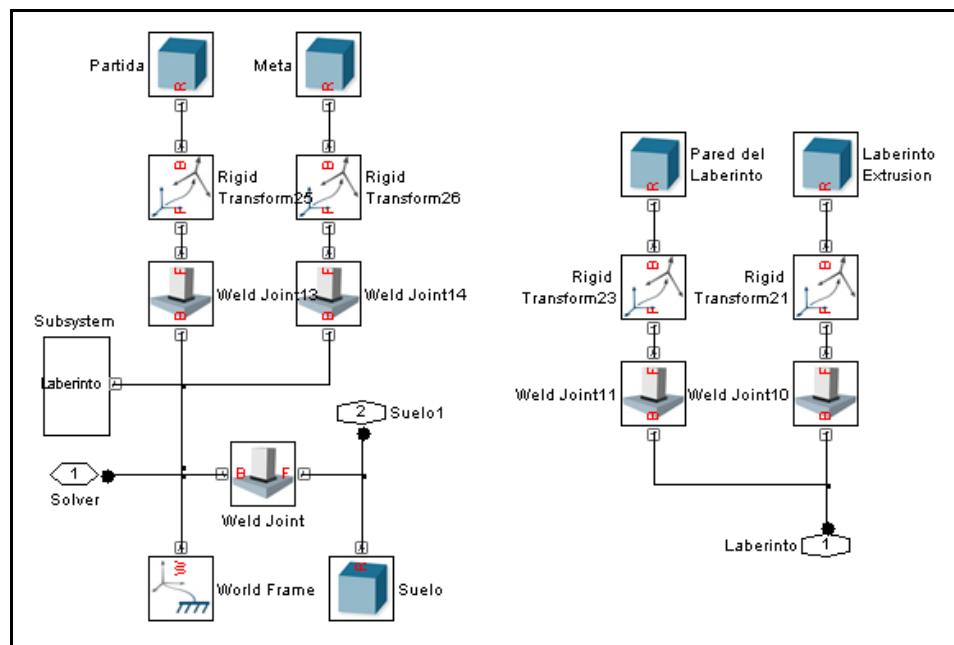


FIGURA 4.31. AMBIENTE DE LA SIMULACIÓN

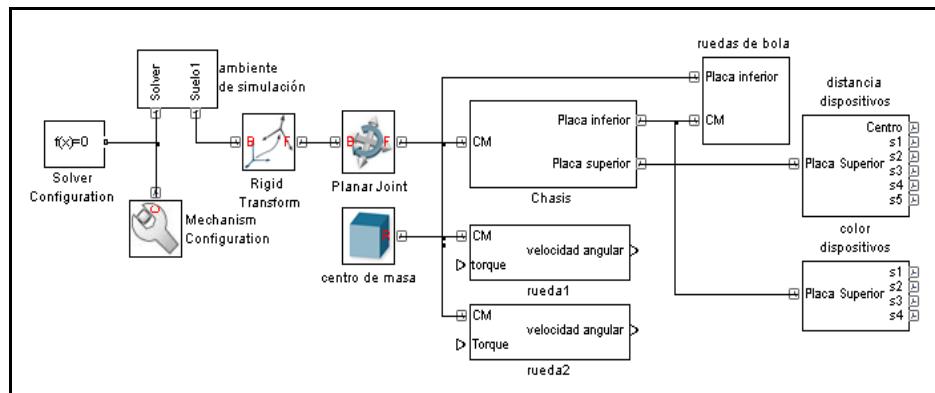


FIGURA 4.32. DIAGRAMA GENERAL DEL MÓVIL

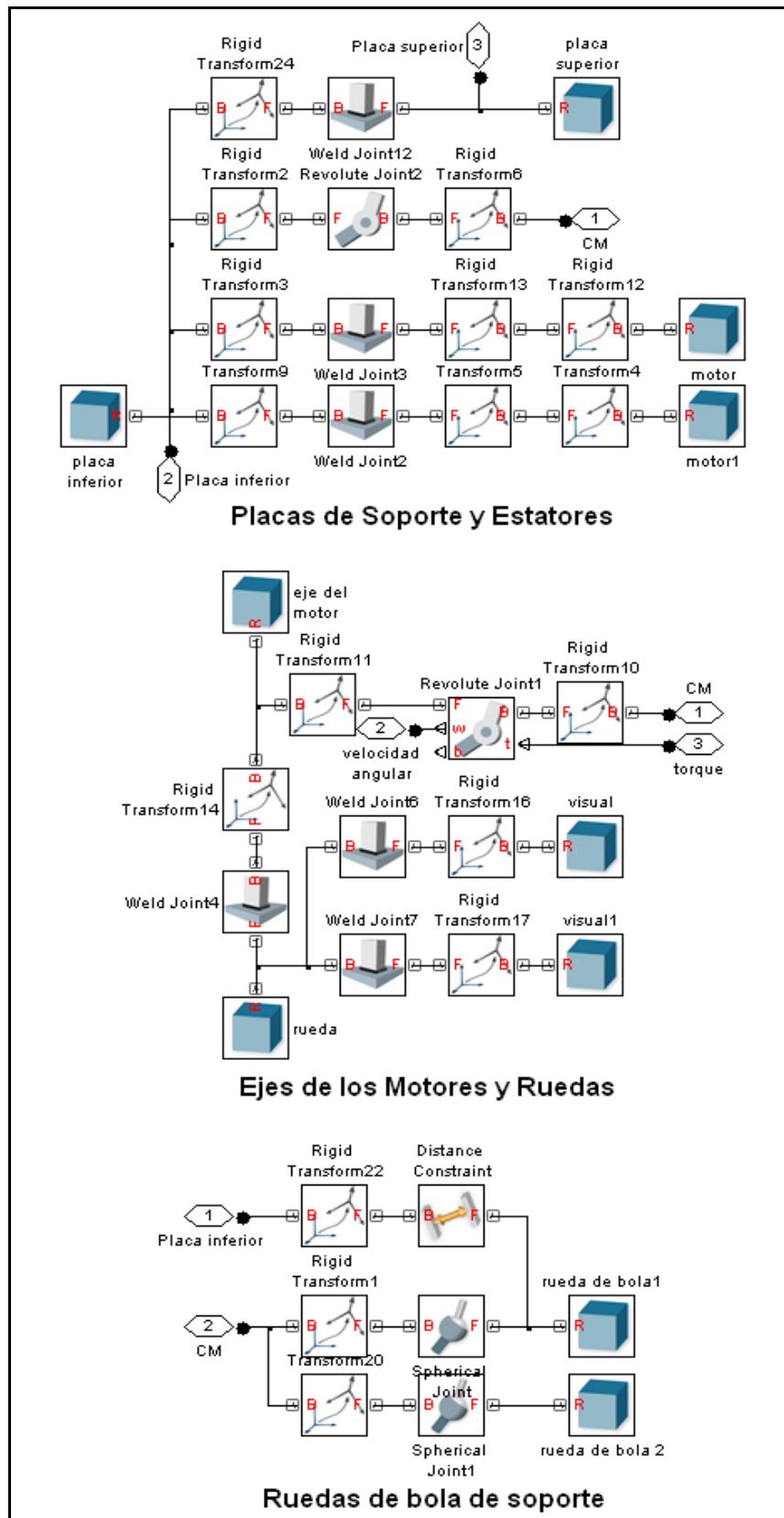


FIGURA 4.33. PARTES MECÁNICAS DEL MÓVIL

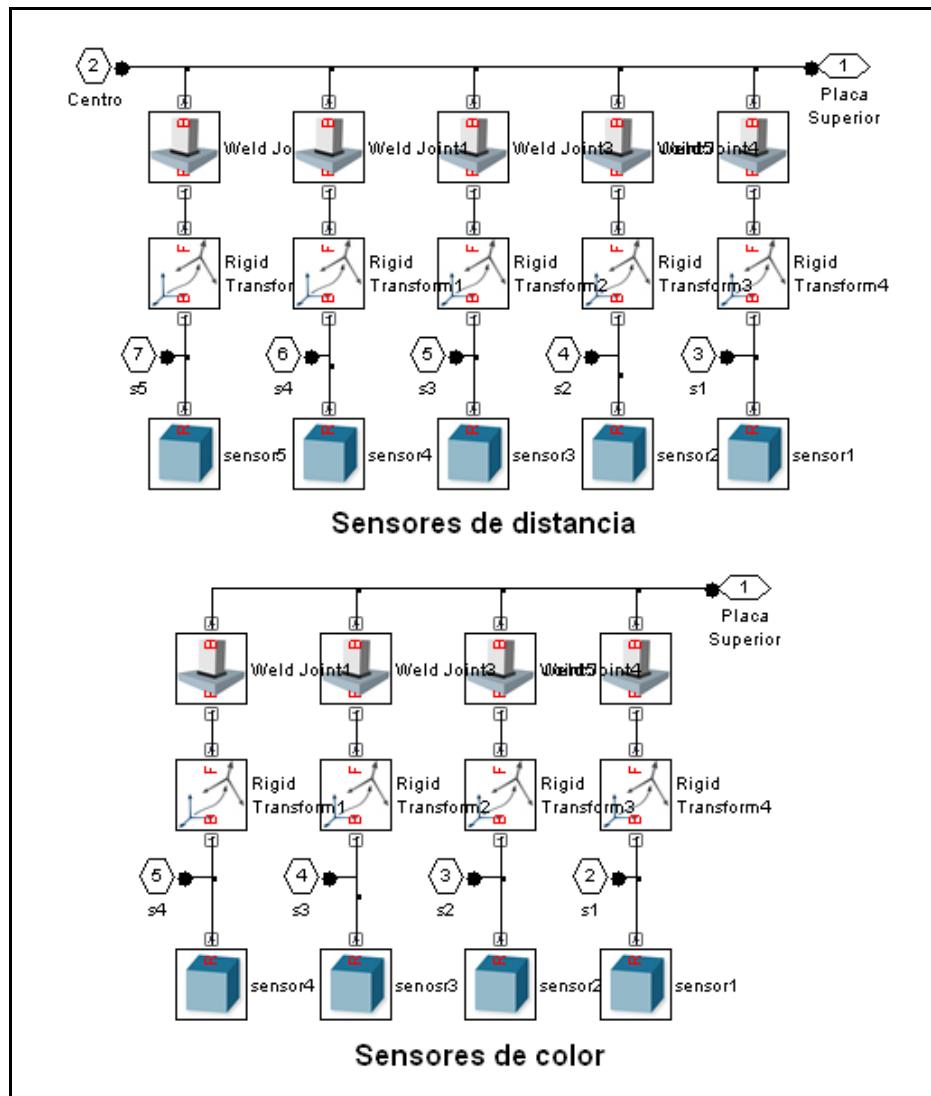


FIGURA 4.34. SENSORES DEL ROBOT

SimMechanics permite acrecentar el nivel de realismo respecto de la simulación propia llevada a cabo en Labview, ya que incorpora interacciones dinámicas, como la fuerza de gravedad, la fuerza centrípeta, las inercias translacional y rotacional de los objetos, la fricción en articulaciones o juntas cilíndricas. Para esta validación se tomaron ciertas consideraciones adicionales:

- Todos los cuerpos individuales simulados son completamente rígidos.
- De la consideración anterior también se deriva que los centros de todas las ruedas se encontrarán siempre a una distancia igual al radio de dicha rueda respecto del plano XY donde se produce el movimiento, esto representa una restricción física esencial para la simulación.
- Debido a que SimMechanics no permite simular de ninguna forma práctica la interacción por fricción entre objetos rodantes y el piso correspondiente, entonces para visualizar el desplazamiento del robot se empleó el mismo modelo matemático de cinemática pura para móviles con tracción diferencial al cual se recurrió en la simulación en Labview [135], dicho modelo posibilita fijar la posición y orientación del centro de masa del robot en cualquier instante, en función únicamente de la velocidad angular existente en motores y ruedas anexas, y de acuerdo a la ecuación de estado siguiente:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega \cdot \delta t) & -\sin(\omega \cdot \delta t) & 0 \\ \sin(\omega \cdot \delta t) & \cos(\omega \cdot \delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega \cdot \delta t \end{bmatrix} \quad (4.4)$$

Donde $R = \frac{l}{2} \cdot \frac{V_r + V_l}{V_r - V_l}; \quad \omega = \frac{V_r - V_l}{l}; \quad Icc_x = x - R \cdot \sin(\theta); y$

$$Icc_y = y + R \cdot \cos(\theta).$$

Respecto a los parámetros ingresados a la simulación, las piezas se modelaron como sólidos geométricos simples con masa con las mismas dimensiones básicas indicadas en el punto 4.4 de este informe (ver figura 4.36.), asimismo se incorporó un coeficiente de amortiguamiento de 4×10^{-6} N-m/(grados/s) entre los ejes de ambos motores y sus respectivos rotores.

4.9.2. Simulación de los actuadores

Simscape y SimElectronics hicieron posible la simulación de un motor DC, cuya velocidad es controlada en lazo abierto mediante una señal PWM, el cambio de giro del motor es implementado a través de un puente H, además se acondicionó un reductor de velocidad de ratio 80:1 para conseguir velocidades similares a los servomotores comúnmente empleados en este tipo de aplicaciones. La salida del motor DC se transmite en forma de torque al robot diseñado en SimMechanics gracias al uso de bloque que representa a un sensor ideal de torque; generando velocidades de 50 rpm en los sentidos horario y antihorario. El diagrama de bloques de este sistema electromecánico se visualiza a continuación, asimismo la tabla 4.9 describe los parámetros del motor DC que fueron manejados.

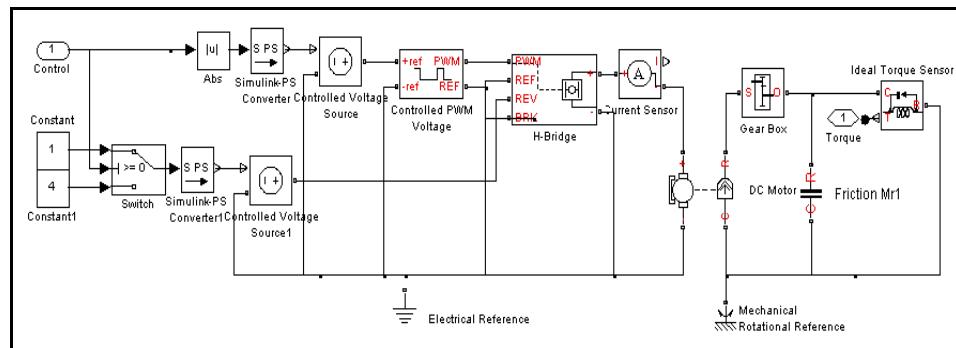


FIGURA 4.35. ACTUADORES DEL AGENTE ROBOTICO

TABLA 4.10. PARAMETROS DE UN MOTOR DC

Parámetro	Valor	Unidad
Inductancia de Armadura	0.01	H
Velocidad sin carga	4000	Rpm
Velocidad nominal	2500	Rpm
Carga nominal	10	W
Voltaje nominal	12	V
Inercia del rotor	2000	g.cm^2
Amortiguamiento de rotor	1e-6	N.m(rad/s)

4.9.3. Simulación de los sensores de distancia

La simulación de los sensores de distancia y los sensores de color fue la que presentó mayores inconvenientes desde el punto de vista de encontrar bloques dedicados para ello en Simulink/Simscape/SimMechanics/SimElectronics, no encontrándose resultados satisfactorios, tampoco existen bloques que imiten las curvas características de los sensores de distancia.

Por ello esta simulación se valió de la lógica implementada anteriormente en el entorno Labview, donde la estrategia de detección consiste simplemente en determinar las distancias y los ángulos formados entre cada sensor y la pared más cercana, ello gracias a sensores virtuales de SimMechanics que se pueden agregar a la simulación; luego estos dos parámetros calculados ingresan a una tabla o matriz de datos, y finalmente mediante una interpolación lineal doble es posible definir la lectura instantánea del sensor. Asimismo, las tablas empleadas para cada sensor se mantuvieron idénticas a los parámetros usados dentro de Labview.

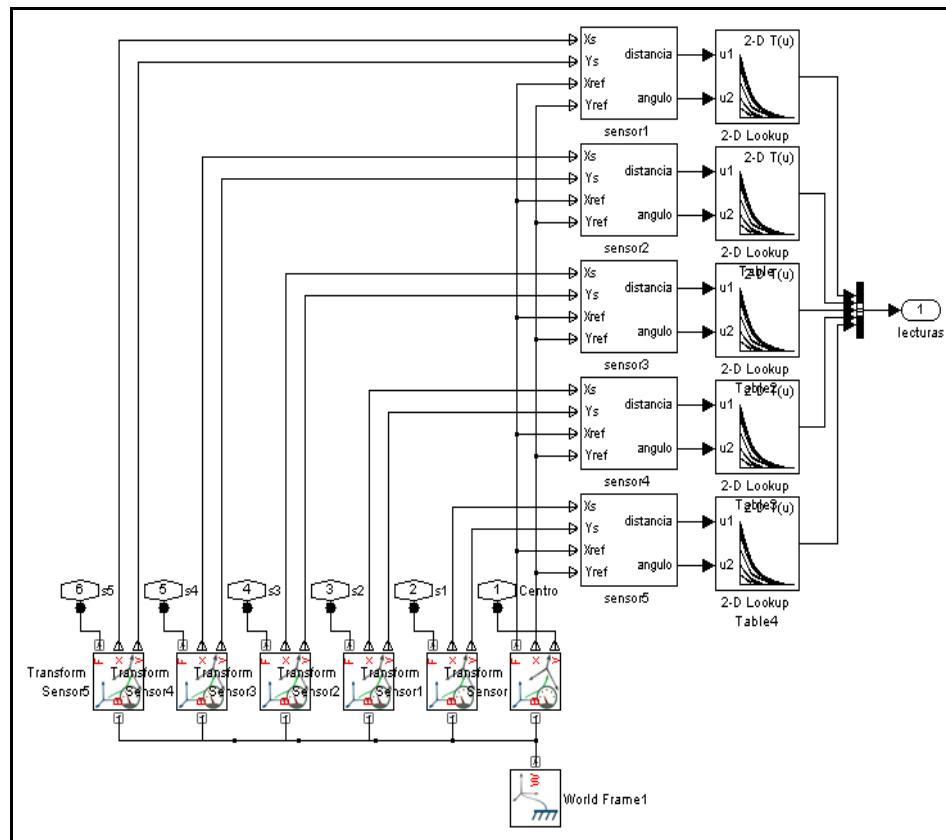


FIGURA 4.36. SENsoRES DE DISTANCIA

TABLA 4.11. TABLA DE VALORES DEL SENSOR1

TABLA 4.12. TABLA DE VALORES DEL SENSOR2

TABLA 4.13. TABLA DE VALORES DEL SENSOR3

		DATA UTILIZADA POR LOS SENSORES ($I_{RA,REL}$)																		
mm.	grados	-18	-16	-14	-12	-10	-8	-6	-4	-2	0	2	4	6	8	10	12	14	16	18
5	0.07	0.20	0.36	0.59	0.80	0.87	0.93	0.96	0.97	1.00	0.98	0.96	0.92	0.88	0.81	0.60	0.35	0.22	0.07	
10	0.07	0.15	0.24	0.44	0.57	0.63	0.66	0.70	0.70	0.72	0.71	0.69	0.66	0.62	0.57	0.42	0.26	0.16	0.07	
15	0.07	0.08	0.15	0.23	0.33	0.36	0.39	0.41	0.41	0.42	0.41	0.40	0.38	0.36	0.34	0.25	0.14	0.11	0.07	
20	0.07	0.07	0.11	0.16	0.22	0.24	0.25	0.26	0.27	0.28	0.28	0.26	0.25	0.25	0.23	0.17	0.09	0.07	0.07	
25	0.07	0.07	0.08	0.13	0.16	0.17	0.18	0.19	0.20	0.20	0.19	0.18	0.18	0.17	0.17	0.13	0.07	0.07	0.07	
30	0.07	0.07	0.07	0.09	0.12	0.12	0.12	0.13	0.14	0.15	0.13	0.13	0.13	0.12	0.12	0.10	0.07	0.07	0.07	
35	0.07	0.07	0.07	0.07	0.09	0.10	0.10	0.10	0.10	0.11	0.11	0.11	0.10	0.09	0.08	0.07	0.07	0.07	0.07	
40	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.09	0.09	0.09	0.09	0.08	0.08	0.07	0.07	0.07	0.07	0.07	
45	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	
50	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	

$I_{RA,REL}$: Fotocorriente Inversa Relativa

TABLA 4.14. TABLA DE VALORES DEL SENSOR4

		DATA UTILIZADA POR LOS SENSORES ($I_{RA,REL}$)																		
mm.	grados	-18	-16	-14	-12	-10	-8	-6	-4	-2	0	2	4	6	8	10	12	14	16	18
5	0.07	0.20	0.36	0.59	0.80	0.87	0.93	0.96	0.97	1.00	0.98	0.96	0.92	0.88	0.81	0.60	0.35	0.22	0.07	
10	0.07	0.15	0.24	0.44	0.57	0.63	0.66	0.70	0.70	0.72	0.71	0.69	0.66	0.62	0.57	0.42	0.26	0.16	0.07	
15	0.07	0.08	0.15	0.23	0.33	0.36	0.39	0.41	0.41	0.42	0.41	0.40	0.38	0.36	0.34	0.25	0.14	0.11	0.07	
20	0.07	0.07	0.11	0.16	0.22	0.24	0.25	0.26	0.27	0.28	0.28	0.26	0.25	0.25	0.23	0.17	0.09	0.07	0.07	
25	0.07	0.07	0.08	0.13	0.16	0.17	0.18	0.19	0.20	0.20	0.19	0.18	0.18	0.17	0.17	0.13	0.07	0.07	0.07	
30	0.07	0.07	0.07	0.09	0.12	0.12	0.12	0.13	0.14	0.15	0.13	0.13	0.13	0.12	0.12	0.10	0.07	0.07	0.07	
35	0.07	0.07	0.07	0.07	0.09	0.10	0.10	0.10	0.10	0.11	0.11	0.11	0.10	0.09	0.08	0.07	0.07	0.07	0.07	
40	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.09	0.09	0.09	0.09	0.08	0.08	0.07	0.07	0.07	0.07	0.07	
45	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	
50	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	

$I_{RA,REL}$: Fotocorriente Inversa Relativa

TABLA 4.15. TABLA DE VALORES DEL SENSOR5

		DATA UTILIZADA POR LOS SENSORES ($I_{RA,REL}$)																		
mm.	grados	-18	-16	-14	-12	-10	-8	-6	-4	-2	0	2	4	6	8	10	12	14	16	18
5	0	0.07	0.20	0.36	0.59	0.80	0.87	0.93	0.96	0.97	1.00	0.98	0.96	0.92	0.88	0.81	0.60	0.35	0.22	0.07
10	0	0.07	0.15	0.24	0.44	0.57	0.63	0.66	0.70	0.70	0.72	0.71	0.69	0.66	0.62	0.57	0.42	0.26	0.16	0.07
15	0	0.07	0.08	0.15	0.23	0.33	0.36	0.39	0.41	0.41	0.42	0.41	0.40	0.38	0.36	0.34	0.25	0.14	0.11	0.07
20	0	0.07	0.07	0.11	0.16	0.22	0.24	0.25	0.26	0.27	0.28	0.28	0.26	0.25	0.25	0.23	0.17	0.09	0.07	0.07
25	0	0.07	0.07	0.08	0.13	0.16	0.17	0.18	0.19	0.20	0.20	0.19	0.18	0.18	0.17	0.17	0.13	0.07	0.07	0.07
30	0	0.07	0.07	0.07	0.09	0.12	0.12	0.12	0.13	0.14	0.15	0.13	0.13	0.13	0.12	0.12	0.10	0.07	0.07	0.07
35	0	0.07	0.07	0.07	0.07	0.09	0.10	0.10	0.10	0.10	0.11	0.11	0.11	0.10	0.09	0.08	0.07	0.07	0.07	0.07
40	0	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.09	0.09	0.09	0.09	0.08	0.08	0.07	0.07	0.07	0.07	0.07
45	0	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
50	0	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
$I_{RA,REL}$: Fotocorriente Inversa Relativa																				

4.9.4. Simulación de los sensores de color

En cuanto a la simulación de los sensores de color, está igualmente se planteó tal y como se realizó en Labview, por tanto un par de sensores entrega una salida lógica de “1” cuando se ubican en la partida o meta y un valor lógico de “0” en cualquier otro lugar, mientras que el otro par proporciona una señal de “1” sólo cuando se localizan en la meta, y “0” en cualquier otro caso. Estas señales ingresan después a una función cuyo resultado es “1” cuando el robot se ubica en la meta, “0.5” cuando se encuentra en la partida y “0” en otra zona del ambiente. Para determinar la ubicación del robot se hace uso únicamente de sensores de posición virtuales facilitados por SimMechanics relativos al centro de masa del móvil.

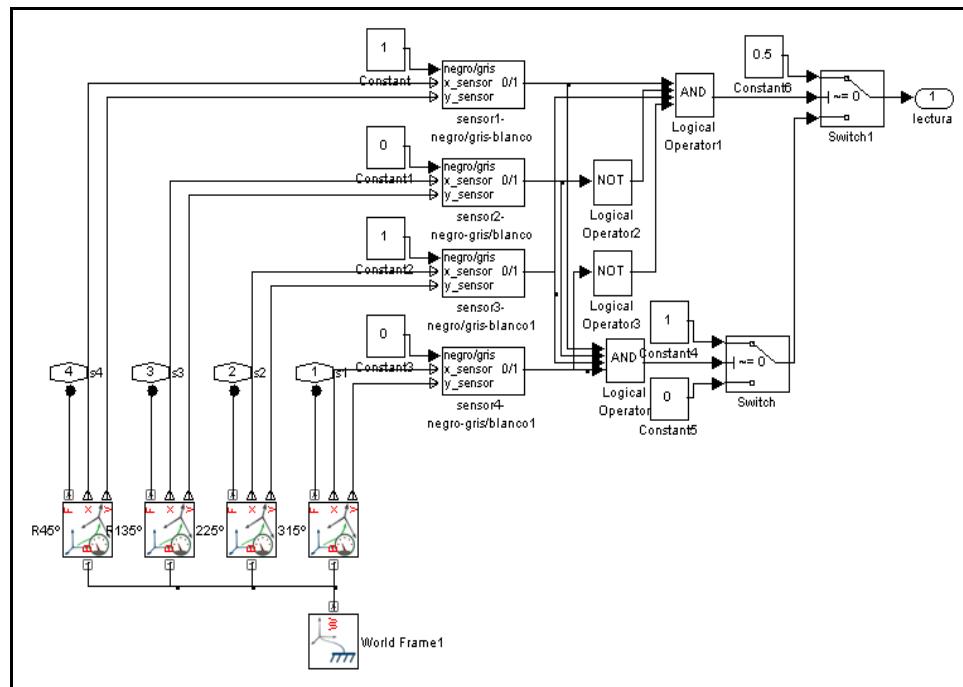


FIGURA 4.37. SENSORES DE COLOR

4.9.5. Simulación del Neurocontrolador

Los arquitecturas resultantes fueron programados directamente en archivos M, y se incorporaron a Simulink a través del bloque “MATLAB Function”. Los pesos y parámetros neuronales son definidos manualmente en dichas funciones, extraídos de los clusters y/o estructuras obtenidas en Labview. El código fuente presentan las líneas siguientes para una red Perceptron Simple.

```

function motores_out =
perceptron_simple(ss_distancia,s_color,motores_in,controlador,dt)
%#codegen
bias=1;
if size(motores_in,2) ~=2
    s_array=[zeros(1,2),transpose(ss_distancia),s_color,bias];
else
    s_array=[motores_in,transpose(ss_distancia),s_color,bias];
end
[rn,cn]=size(controlador);
motores_out=zeros(1,rn);

```

```

for ii=1:rn
    pesos=controlador(ii,:)
    combinacion=sum(s_array.*pesos,2);
    activacion=1/(1+exp(-combinacion)) %%funcion sigmoide
    motores_out(ii)=activacion;
end

```

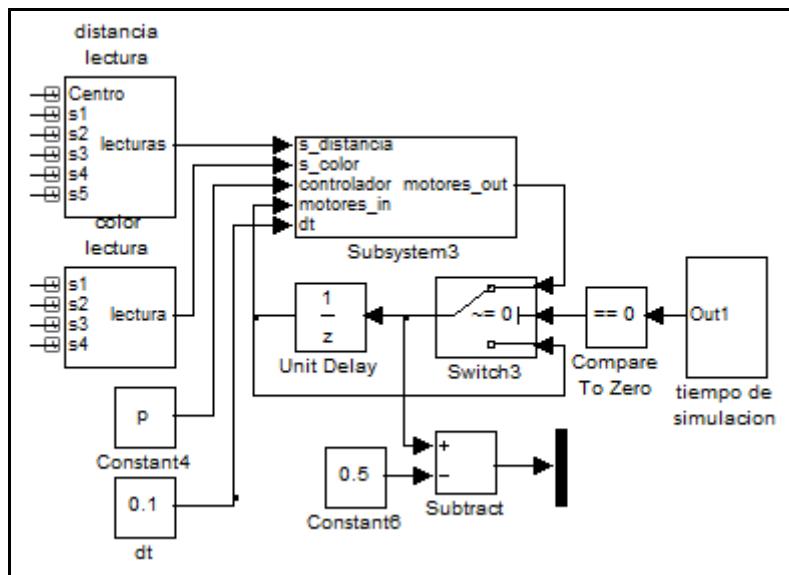


FIGURA 4.38. NEUROCONTROLADOR SIMULADO

Por otra parte, la entrada “controlador”, indica los pesos sinápticos y otros parámetros relativos a cada neurona, para facilitar su traspaso desde Labview al Workspace de Matlab, es recomendable emplear funciones del ActiveX Server en Matlab, de la manera mostrada en seguida.

```

e=actxserver('LabVIEW.Application');
vipath='C:\DATA\Labview\test.vi';
vi=invoke(e,'GetVIReference',vipath);
vi.Run;
Controlador=vi.GetControlValue("name");

```

4.9.6. Simulación de Colisiones

Ya que SimMechanics no cuenta con un método propio para implementar la simulación de choques y fuerzas normales, y recordando

que a futuro se espera que los controladores altamente evolucionados no incurran en colisiones con las paredes del ambiente, por tanto no se requiere un modelamiento preciso de las fuerzas involucradas en una colisión, únicamente se desea que el robot nunca deje los límites del ambiente y que ante un choque sea reposicionado en una ubicación donde ya no esté en contacto con el obstáculo respectivo. En vista de esto, se programó un algoritmo de detección dentro de un bloque “Matlab Function”, el cual puede determinar el momento en que ocurre un choque en base a las previamente conocidas geometrías del laberinto y el autómata y mediante el sensado de la posición y orientación instantánea del robot, luego a partir de ello otros algoritmos pueden mover al robot hacia una posición aleatoria de reposición.

4.9.7. DESARROLLO Y RESULTADOS DE LA SIMULACION

Para esta Validación, se decidió simular los resultados previos obtenidos de la experiencia #2, en particular se realizaron pruebas de las dos mejores configuraciones sensoriales resultantes de la evolución morfológica.

Respecto a la simulación propiamente dicha, se tomaron algunas consideraciones que valen la pena mencionar. Primero, como se explicó anteriormente, se ha considerado un modelamiento cinemático, lo que implica que se deben definir la posición y orientación del robot en cada actualización, sin embargo en SimMechanics Segunda Generación no existe una forma directa de lograr ello.

Con ello en mente se recurrió a implementar 3 controladores PID que permitan movilizar al autómata a la ubicación deseada (x, y, θ). Estos controladores no fueron diseñados, sino se ejecutó un autotunning para descubrir las ganancias apropiadas, igualmente los mismos PIDs se utilizan para reposicionar el robot después de una colisión. Siguiendo con los controladores, estos presentan un tiempo de asentamiento de alrededor de 0.25 segundos, puesto que las simulaciones en Labview se desarrollaron con un paso de tiempo de 0.1 segundo, entonces se debe agregar al modelamiento cinemático un reloj o “clock” que permita una actualización sólo cada 0.25 segundos (figura 4.39).

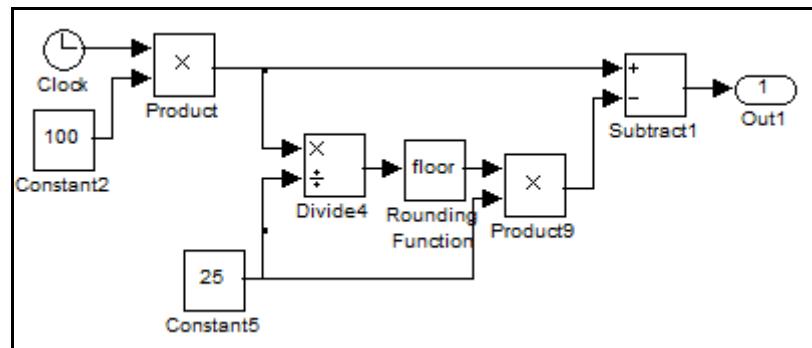


FIGURA 4.39. RELOJ DE SINCRONIZACION

Por otro lado el algoritmo de simulación aplicado fue “ode14x” con paso de simulación fijo de 0.01 segundos, igualmente aquí el controlador debe sincronizarse para actuar sólo cada 0.1 segundos.

El diagrama de bloque general se muestra a continuación.

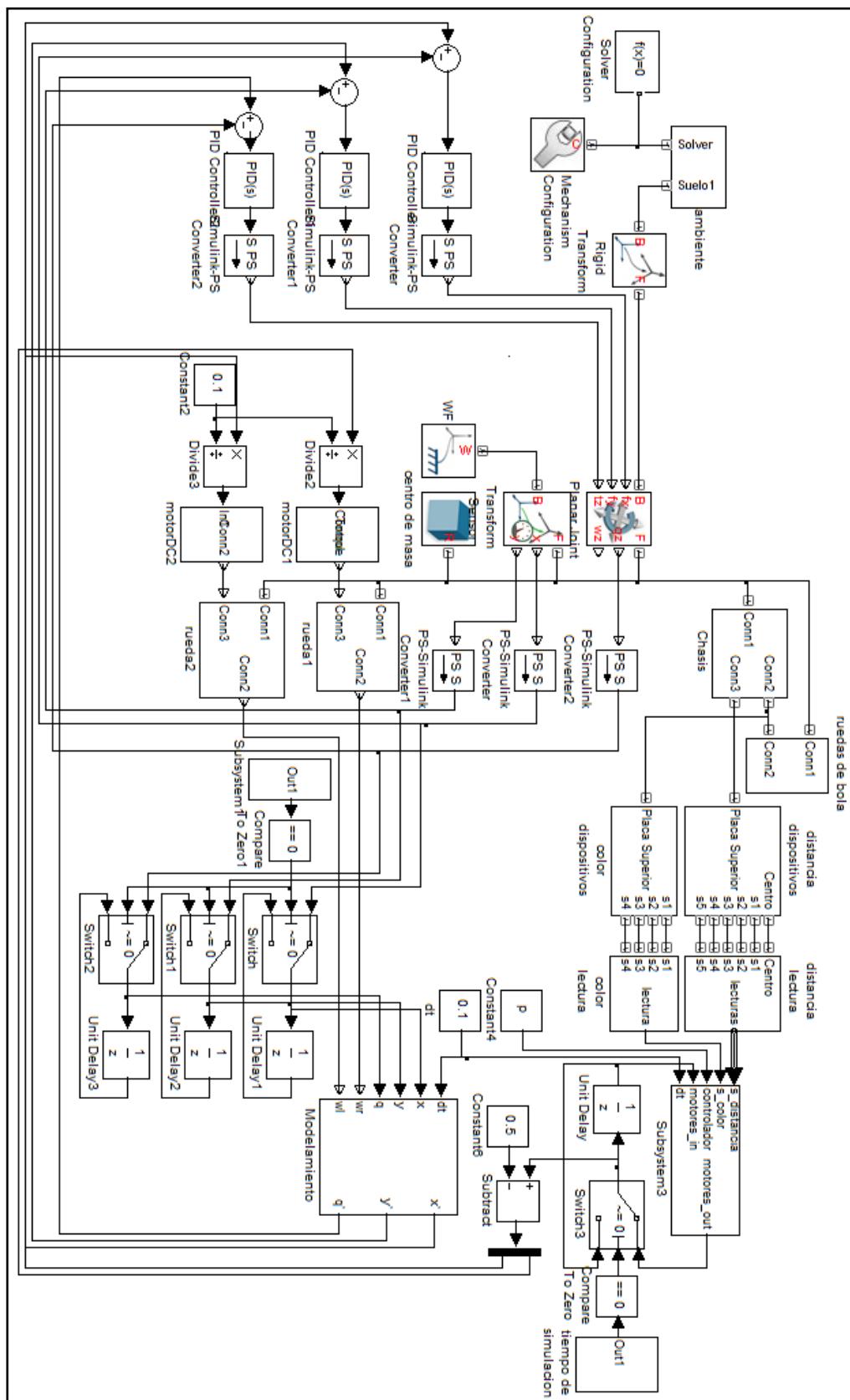


FIGURA 4.40. DIAGRAMA DE BLOQUES DE LA SIMULACION

En cuanto a la experiencia #2: Evolución Morfología, los arreglos de sensores empleados fueron: Prueba 4: [sensor1: 6.35°, sensor2: -67.06°, sensor3: -16.23°, sensor4: -35.71°, sensor5: -54.35°], y Prueba 1: [sensor1: 69.88°, sensor2: 85.41°, sensor3: -0.705°, sensor4: 72.70°, sensor5: -32.71°], asimismo se realizó 50 simulaciones y se calculó el número de colisiones promedio recorrido ejecutado.

Controlador	Recorridos	Colisiones	Colisiones Promedio
Prueba #4	50	4	0.08
Prueba #1	50	7	0.14

En general el promedio de colisiones aumentó considerablemente debido al cambio de ambiente, sin embargo los controladores aún mostraron un comportamiento similar al encontrado en su mundo virtual de origen, por lo que la evolución artificial no fue un despropósito total, aunque la eficiencia si sufrió la modificación.

CONCLUSIONES Y RECOMENDACIONES

- 1.** Es factible implementar mediante Algoritmos Evolutivos controladores robóticos capaces de desarrollar comportamientos reactivos tales como la coordinación visomotora para evadir obstáculos, e incluso comportamientos no-reactivos como es el caso de la navegación autónoma inteligente.
- 2.** Las funciones de desempeño empleadas en este informe fueron creadas bajo intervención humana de manera específica para el problema en cuestión, obteniéndose resultados insatisfactorios en la mayoría de los casos; por tanto es recomendable el uso de metodologías que permitan definir funciones de desempeño con menor inferencia del programador y mayor generalidad, posibles alternativas son la Evolución Competitiva y métricas de novedad basadas en el análisis del comportamiento de los individuos.
- 3.** Es factible implementar la evolución morfológica de un sistema robótico, lo cual permitiría evitar complicaciones debido a un mal diseño.
- 4.** Se comprobó la habilidad innata de redes dinámicas, reservorios de neuronas y sinapsis plásticas para almacenar información a través del tiempo, ésta a su vez puede ser utilizada en algún instante posterior.

5. En la primera experiencia se observó como la diferencia entre un factor fenotípico discreto y uno continuo es aprovechada negativamente por la evolución, por tanto se recomienda agregar un ruido aleatorio respecto a la función Fitness que se pretende utilizar.
6. En las experiencias realizadas para este informe se reportó en múltiples ocasiones que la evolución sufrió de un estancamiento, para contrarrestar dicha situación es recomendable investigar métodos evolutivos con tasas de mutación y recombinación variables a través de las generaciones; asimismo se observó en muchas oportunidades que previo al mencionado estancamiento aparecieron controladores con un desempeño ocasional mejor al apreciable en generaciones posteriores, por tanto sería aconsejable experimentar con otros métodos de asignación de Fitness, sobre todo aquellos que otorguen mayor importancia al Elitismo.
7. Es sugerible la implementación e investigación de Representaciones con números reales y los Operadores Evolutivos correspondientes, ya que así se evitaría una encriptación adicional de variables propias del problema, como sucedió en la segunda experiencia para la optimización de la posición de los sensores de distancia.
8. Resulta recomendable el empleo y estudio de Representaciones Genéticas y Estructuras Neuronales que presenten una mayor similitud con la Biología, ya que años recientes el intercambio de información e ideas entre los campos de Inteligencia Artificial y Neurociencias ha sido beneficioso para ambos.

9. Con la finalidad de reducir los tiempos de simulación se aconseja desarrollar una programación más eficiente en el lenguaje G de Labview, por ejemplo National Instruments recomienda evitar la asignación dinámica de memoria, no colocar constantes dentro de bucles FOR o WHILE, usar variables globales en lugar de variables locales, usar operaciones de enteros y no punto flotante, no usar clusters, evitar la estructura CASE para códigos simples.
10. Con el objetivo de disminuir el tiempo de simulación de una Población, se sugiere implementar un Algoritmo Evolutivo que aproveche las capacidades multi-core de procesadores actuales como el AMD Opteron 6200 (16 núcleos); para ello es adecuado utilizar Labview como lenguaje de programación y se requiere distribuir equitativamente la Simulación de los individuos de una Población entre el número total de núcleos mediante estructuras WHILE que trabajen en paralelo y comuniquen los resultados alcanzados en cada iteración gracias a variables locales, globales o compartidas.

BIBLIOGRAFIA

- [1] V. Braitenberg, "Vehicles, Experiments in Synthetic Psychology", Cambridge, MA: MIT Press, 1984.
- [2] D. Floreano, F. Mondada, "Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot", 1994.
- [3] I. Harvey, D. Cliff, P. Husbands, "Seeing the Light: Artificial Evolution, Real Vision", 1994.
- [4] O. Miglino, K. Nafasi, C.E. Taylor, "Selection for Wandering Behaviour in a small Robot", en *Artificial Life*, vol. 2, pp. 101-116, The MIT Press, 1994.
- [5] M.A. Lewis, A.H. Faag, G.A. Bekey, "Genetic Algorithms for Gait Synthesis in a Hexapod Robot", 1994.
- [6] K. Sims, "Evolving 3D Morphology and Behaviour by Competition", 1994.
- [7] A. Thompson, "Evolving Electronic Robot Controllers that exploit Hardware Resources", 1995.

- [8] R.A. Watson, S.G. Ficici, J.B. Pollack, "Embodied Evolution: Embodying an Evolutionary Algorithm in a Population of Robots", 1999.
- [9] T. Reil, P. Husbands, "Evolution of Central Pattern Generators for Bipedal Walking in a Real-Time Physics Environment", 2002.
- [10] D. Floreano, D. Marocco, T. Kato, E. Sauser, "Coevolution of Active Vision and Feature Selection", 2003.
- [11] G.J. Barlow, C.K. Oh, "Autonomous Controller Desing for Unmanned Aerial Vehicles using Multi-Objective Genetic Programming", 2004.
- [12] R.A. Tellez, C. Angulo, D.E. Pardo, "Evolving the Walking Behaviour of a 12 DOF Quadruped using a Distributed Neural Architecture", 2006.
- [13] M. Eaton, T.J. Davitt, "Evolutionary Control of Bipedal Locomotion in a High Degree-Of-Freedom Humanoid Robot: First Steps", en *Artificial Life and Robotics*, vol. 11, No. 1, Springer, pp. 112-115, 2007.
- [14] A.J. Ijspeert, A. Crespi, D. Ryczko, J.M. Cabelguen, "From Swimming to Walking with a Salamander Robot Driven by a Spinal Cord Model", en *Science*, vol. 315, No. 5817, pp. 1416-1420, 2007.
- [15] T. Tohge, H. Iba, "Evolutionary Morphology for Polycube Robots", en *Frontiers In Evolutionary Robotics*, ed. H. Iba, I-Tech Education and Publishing, pp. 221-232, 2008.

- [16] www.swarm-bots.org
- [17] <http://mobots.epfl.ch/s-bot.html>
- [18] M. Suzuki, T. Gritti, D. Floreano, "Active Vision for Goal-Oriented Humanoid Robot Walking", en *Creating Brain-Like Intelligence*, Springer, pp. 303-313, 2009.
- [19] F. Ruini, A. Cangelosi, "Distributed Control in Multi-Agent Systems: A Preliminary Model of Autonomous MAV Swarms", 2008.
- [20] M. Peniak, D. Marocco, A. Cangelosi, "Coevolving Controller and Sensing Abilities in a Simulated Mars Rover Explorer", 2009.
- [21] www.swarmanoid.org
- [22] www.symbriion.eu
- [23] www.replicators.eu
- [24] K.A. De Jong, Simple EAs as Parallel Adaptive Search, en *Evolutionary Computation: A Unified Approach*, MIT Press, pp. 71, 2006.
- [25] M. Mitchell, The Appeal of Evolution, en *An Introduction to Genetic Algorithms*, MIT Press, pp. 4, 1998.

- [26] M. Mitchell, A Brief History of Evolutionary Computation, en *An Introduction to Genetic Algorithms*, MIT Press, pp. 2-3, 1998.
- [27] S.N. Sivanandam, S.N. Deepa, The Historical Development of EC, en *Introduction to Genetic Algorithms*, Springer, pp. 2-5, 2007.
- [28] M. Mitchell, Modeling Interactions between Learning and Evolution, en *An Introduction to Genetic Algorithms*, MIT Press, pp. 66-75, 1998.
- [29] A.E. Eiben, J.E. Smith, The Inspiration from Biology, en *Introduction to Evolutionary Computing*, 1ra edición, Springer, pp. 2-7, 2003.
- [30] R.L. Haupt, S.E. Haupt, Biological Optimization: Natural Selection, en *Practical Genetic Algorithms*, 2da edición, John Wiley & Sons, pp. 19-22, 2004.
- [31] J.S. Huxley, “Evolution: The Modern Synthesis”, The Definitive Edition, MIT Press, 2010.
- [32] S.N. Sivanandam, S.N. Deepa, Genetics, en *Introduction to Genetic Algorithms*, Springer, pp. 17, 2007.
- [33] K.A. De Jong, EV: A Simple Evolutionary System, en *Evolutionary Computation: A Unified Approach*, MIT Press, pp. 3-6, 2006.

- [34] A.E. Eiben, J.E. Smith, What is an Evolutionary Algorithm?, en *Introduction to Evolutionary Computing*, 1ra edición, Springer, pp. 15-18, 2003.
- [35] M. Mitchell, How do Genetic Algorithms work?, en *An Introduction to Genetic Algorithms*, MIT Press, pp. 21-23, 1998.
- [36] S.N. Sivanandam, S.N. Deepa, Evolution and Genetic Algorithms, en *Introduction to Genetic Algorithms*, Springer, pp. 23, 2007.
- [37] T. Weise, Genomes in Genetic Algorhitms, en *Global Optimization Algorithms – Theory and Application*, pp. 145-147, 2008.
- [38] D. Floreano, P. Dürr, C. Mattiusi, “Neuroevolution: From Architectures to Learning” - Evolution of Neural Architectures, en *Evolutionary Intelligence*, vol. 1, No 1, Springer, pp. 48-52, 2008.
- [39] P.J. Fleming, A.M. Zalzala, Encoding Neural Networks in Chromosomes, en *Genetic Algorithms In Engineering Systems*, vol. 55, IEE Control Engineering Series, pp. 100-103, 1997.
- [40] P. Köhn, Direct Encoding, en *Combining Genetic Algorithms and Neural Networks: The Encoding Problem*, Tesis para optar por el grado de Master en Ciencias, Universidad de Tennessee, Knoxville, pp. 17-19, 1994.
- [41] D.E. Moriarty, R. Miikkulainen, “Efficient Reinforcement Learning through Symbiotic Evolution”, 1994.

- [42] F.J. Gomez, "Robust Non-Linear Control through Neuroevolution", Disertación para optar por el grado de Doctor en Filosofía, Universidad de Texas en Austin, 2003.
- [43] K.O. Stanley, R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies", 2002.
- [44] F. Gruau, "Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm", Tesis para optar por el diploma de Doctor en Ciencias de la Computación, Universidad Claude Bernard-Lyon I, 1994.
- [45] D. Floreano, P. Dürr, C. Mattiussi, "Neuroevolution: From Architectures to Learning" - Implicit Encoding, en *Evolutionary Intelligence*, vol. 1, No 1, Springer, pp. 52-54, 2008.
- [46] C. Mattiussi, D. Floreano, "Analog Genetic Encoding for the Evolution of Circuits and Networks", en *IEEE Transactions on Evolutionary Computation*, vol. 11, No 5, pp. 596-607, 2007.
- [47] C.R. Reeves, J.E. Rowe, Initial Population, en *Genetic Algorithms - Principles and Perspectives: A Guide to GA Theory*, Kluwer Academic Publishers, pp. 25-30, 2003.
- [48] R.L. Haupt, S.E. Haupt, Population, en *Practical Genetic Algorithms*, 2da edición, John Wiley & Sons, pp. 117-121, 2004.

- [49] S.N. Sivanandam, S.N. Deepa, Populations, en *Introduction to Genetic Algorithms*, Springer, pp. 41, 2007.
- [50] S. Sumathi, T. Hamsapriya, P. Surekha , Evaluation/Fitness Function, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 37, 2008.
- [51] A.E. Eiben, J.E. Smith, Evaluation Function (Fitness Function), en *Introduction to Evolutionary Computing*, 1ra edición, Springer, pp. 19, 2003.
- [52] S. Sumathi, T. Hamsapriya, P. Surekha , Selection, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 38, 2008.
- [53] T. Weise, Fitness Proportionate Selection, en *Global Optimization Algorithms – Theory and Application*, pp. 124-127, 2008.
- [54] S. Sumathi, T. Hamsapriya, P. Surekha , Rank Based Fitness Assignment, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 39-41, 2008.
- [55] M. Gen, R. Cheng, Fitness Sharing and Population Diversity, en *Genetic Algorithms & Engineering Optimization*, 1ra edición, John Wiley & Sons, pp. 111-113, 2000.

- [56] C.R. Reeves, J.E. Rowe, New Population: Crowding and Niching, en *Genetic Algorithms - Principles and Perspectives: A Guide to GA Theory*, Kluwer Academic Publishers, pp. 46-48, 2003.
- [57] T. Weise, Clearing and Simple Convergence Prevention (SCP), en *Global Optimization Algorithms – Theory and Application*, pp. 134-137, 2008.
- [58] D. Ashlock, Niche Specialization, en *Evolutionary Computation for Modeling and Optimization*, Springer, pp. 78-82, 2006.
- [59] S.N. Sivanandam, S.N. Deepa, Niche and Speciation, en *Introduction to Genetic Algorithms*, Springer, pp. 89-97, 2007.
- [60] L.C. Jain, N.M. Martin, Multi-Objective Evolutionary Algorithms in Gas Turbine Aero-Engine Control: Decision Strategies, en *Fusion of Neural Networks, Fuzzy Systems and Genetic Algorithms: Industrial Applications*, CRC Press, pp. 250-251, 1998.
- [61] M. Gen, R. Cheng, Pareto Ranking Method, en *Genetic Algorithms & Engineering Optimization*, 1ra edición, John Wiley & Sons, pp. 118-122, 2000.
- [62] R.L. Haupt, S.E. Haupt, Pareto Optimization, en *Practical Genetic Algorithms*, 2da edición, John Wiley & Sons, pp. 99-101, 2004.

- [63] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II”, 2002.
- [64] E. Zitzler, M. Laumanns, L. Thiele, “SPEA2: Improving the Strength Pareto Evolutionary Algorithm”, 2001.
- [65] R.L. Haupt, S.E. Haupt, Sum of Weighted Cost Functions, en *Practical Genetic Algorithms*, 2da edición, John Wiley & Sons, pp. 99, 2004.
- [66] M. Gen, R. Cheng, Vector Evaluated Genetic Algorithms, en *Genetic Algorithms & Engineering Optimization*, 1ra edición, John Wiley & Sons, pp. 115-118, 2000.
- [67] A. Abraham, Evolution Computation: Selection and Reproduction, vol. 3, section 3, en *Handbook of Measuring System Desing*, ed. P.H. Sydenham, R. Thorne, John Wiley & Sons, pp. 922-923, 2005.
- [68] S. Sumathi, T. Hamsapriya, P. Surekha , Roulette Wheel Selection, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 43, 2008.
- [69] S.N. Sivanandam, S.N. Deepa, Stochastic Universal Sampling, en *Introduction to Genetic Algorithms*, Springer, pp. 50, 2007.
- [70] T. Weise, Truncation Selection, en *Global Optimization Algorithms – Theory and Application*, pp. 122-124, 2008.

- [71] A.E. Eiben, J.E. Smith, Tournament Selection, en *Introduction to Evolutionary Computing*, 1ra edición, Springer, pp. 63, 2003.
- [72] M. Mitchell, Selection Methods: Boltzman Selection, en *An Introduction to Genetic Algorithms*, MIT Press, pp. 126, 1998.
- [73] S. Sumathi, T. Hamsapriya, P. Surekha , Global Population Models, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 226, 2008.
- [74] S. Sumathi, T. Hamsapriya, P. Surekha , Local Population Models, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 228-230, 2008.
- [75] S. Sumathi, T. Hamsapriya, P. Surekha , Local Selection, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 45-47, 2008.
- [76] S. Sumathi, T. Hamsapriya, P. Surekha , Regional Population Models, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 226-228, 2008.
- [77] A.E. Eiben, J.E. Smith, Variation Operators – Recombination, en *Introduction to Evolutionary Computing*, 1ra edición, Springer, pp. 21, 2003.

- [78] S. Sumathi, T. Hamsapriya, P. Surekha , Discrete Recombination, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 52, 2008.
- [79] S. Sumathi, T. Hamsapriya, P. Surekha , Real Valued Recombination, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 53-57, 2008.
- [80] S.N. Sivanandam, S.N. Deepa, Crossover (Recombination), en *Introduction to Genetic Algorithms*, Springer, pp. 50-56, 2007.
- [81] S.N. Sivanandam, S.N. Deepa, Crossover Probability, en *Introduction to Genetic Algorithms*, Springer, pp. 56, 2007.
- [82] S.N. Sivanandam, S.N. Deepa, Mutation, en *Introduction to Genetic Algorithms*, Springer, pp. 56-57, 2007.
- [83] S. Sumathi, T. Hamsapriya, P. Surekha , Real Valued Mutation, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 62, 2008.
- [84] S. Sumathi, T. Hamsapriya, P. Surekha , Real Valued Mutation with Adaptation of Step Sizes, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 64, 2008.

- [85] S. Sumathi, T. Hamsapriya, P. Surekha , Other Types of Mutation, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 66, 2008.
- [86] K.A. De Jong, Survival Selection: A Special Case, en *Evolutionary Computation: A Unified Approach*, MIT Press, pp. 59, 2006.
- [87] T. Weise, Classification of Evolutionary Algorithms: Populations in Evolutionary Algorithms, en *Global Optimization Algorithms – Theory and Application*, pp. 102-103, 2008.
- [88] C.R. Reeves, J.E. Rowe, New Population, en *Genetic Algorithms - Principles and Perspectives: A Guide to GA Theory*, Kluwer Academic Publishers, pp. 45, 2003.
- [89] D. Ashlock, Models of Evolution, en *Evolutionary Computation for Modeling and Optimization*, Springer, pp. 33-36, 2006.
- [90] S. Sumathi, T. Hamsapriya, P. Surekha , Reinsertion, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 67-68, 2008.
- [91] A.E. Eiben, J.E. Smith, Termination Condition, en *Introduction to Evolutionary Computing*, 1ra edición, Springer, pp. 23, 2003.

- [92] S.N. Sivanandam, S.N. Deepa, Search Termination (Convergence Criteria), en *Introduction to Genetic Algorithms*, Springer, pp. 59-60, 2007.
- [93] K.A. De Jong, Convergence and Stopping Criteria, en *Evolutionary Computation: A Unified Approach*, MIT Press, pp. 78, 2006.
- [94] S. Sumathi, T. Hamsapriya, P. Surekha , Types of Genetic Algorithm, en *Evolutionary Intelligence: An Introduction to Theory and Applications with Matlab*, Springer, pp. 109-121, 2008.
- [95] S.N. Sivanandam, S.N. Deepa, Classification of Genetic Algorithm, en *Introduction to Genetic Algorithms*, Springer, pp. 105-126, 2007.
- [96] <http://shark-project.sourceforge.net/2.1.1/doc/EALib/index.html>
- [97] <http://www.aforgenet.com/framework/>
- [98] <http://jenetics.sourceforge.net/>
- [99] <http://deap.gel.ulaval.ca/doc/0.7/index.html>
- [100] <http://www.geatbx.com/>
- [101] <http://www.ra.cs.uni-tuebingen.de/software/EvA2/introduction.html>
- [102] <http://cs.gmu.edu/~eclab/projects/ecj/>

[103] <http://jclec.sourceforge.net/>

[104] <http://jgap.sourceforge.net/>

[105] <http://gp.zemris.fer.hr/ecf/>

[106] <http://eodev.sourceforge.net/>

[107] <http://dev.heuristiclab.com/trac/hl/core>

[108] <http://www.keel.es/>

[109] <http://www.palisade.com/evolver/>

[110] http://opende.sourceforge.net/wiki/index.php/Main_Page

[111] <http://www.vxsim.com/>

[112] <http://www.cyberbotics.com/overview>

[113] <http://playerstage.sourceforge.net/>

[114] <http://simbad.sourceforge.net/>

[115] O. Miglino, H.H. Lund, S. Nolfi, “Evolving Mobile Robots in Simulated and Real Environments”, 1995.

- [116] N. Jakobi, "Evolutionary Robotics and the Radical Envelope of Noise Hypothesis", 1997.
- [117] C. Hartland, N. Bredeche, "Evolutionary Robotics: From Simulation to the Real World using Anticipation", 2006.
- [118] N. Jakobi, P. Husbands, I. Harvey, "Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics", 1995.
- [119] J.J Grefenstette, C.L. Ramsey, A.C. Schultz, "Learning Sequential Decision Rules using Simulation Models and Competition", 1990.
- [120] N. Jakobi, "Half Baked, Ad Hoc and Noisy: Minimal Simulations for Evolutionary Robotics", 1997.
- [121] J.C. Bongard, H. Lipson, "Nonlinear System Identification using Coevolution of Models and Tests", 2005.
- [122] J. Urzelai, D. Floreano, "Evolution of Adaptive Synapses: Robots with Fast Adaptive Behaviour in New Environments", 2001.
- [123] J.J. Grefenstette, C.L. Ramsey, "An Approach to Anytime Learning", 1992.
- [124] M. Mitchell, Evolving Neural Networks, en *An Introduction to Genetic Algorithms*, MIT Press, pp. 49-60, 1998.

- [125] L. Wang, K.C. Tan, C.M. Chew, Neural Networks, en *Evolutionary Robotics: from Algorithms to Implementations*, World Scientific Publishing Company, pp. 12, 2006.
- [126] D.A. Sofge, M.A. Potter, M.D. Bugajska, A.C. Schultz, "Challenges and Opportunities of Evolutionary Robotics" - Evolved Neural Network Based Control, en *Proceedings of the Second International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS)*, IEEE, 2003.
- [127] D. Graupe, The Perceptron, en *Principles of Artificial Neural Networks*, 2da edición, World Scientific Publishing Company, pp. 17-37, 2007.
- [128] V. Kecman, Radial Basis Function Networks, en *Learning and Soft Computing: Support Vector Machines, Neural Networks and Fuzzy Logic Models*, MIT Press, pp. 313-365, 2001.
- [129] D. Graupe, Fully Recurrent Networks, en *Principles of Artificial Neural Networks*, 2da edición, World Scientific Publishing Company, pp. 234-235, 2007.
- [130] H. Jaeger, "The Echo State Approach to Analysing and Training Recurrent Neural Networks", 2010.
- [131] R.D. Beer, "On the Dynamics of Small Continuous-Time Recurrent Neural Networks", 1995.

- [132] W. Maass, "Networks of Spiking Neurons: The Third Generation of Neural Network Models", 1997.
- [133] S. Nolfi, D. Marocco, "Evolving Robots able to integrate Sensory-Motor Information over Time", en *Theory in Biosciences*, vol. 120, Urban & Fischer Verlag, pp. 287-310, 2001.
- [134] E. Izquierdo, E.A. Di Paolo, "Is an Embodied System ever purely Reactive?", en Proceedings of the 8th European Conference of Artificial Life, ed. M.S. Capcarrere, Springer, pp. 252-261, 2005.
- [135] G. Dudek, M. Jenkin, Locomotion: Differential Drive, en *Computational Principles of Mobile Robotics*, 2da edición, Cambridge University Press, pp. 39-41, 2010.
- [136] A.K. Seth, "Noise and the Pursuit of Complexity: A Study in Evolutionary Robotics", 1998.
- [137] E.A. Di Paolo, I. Harvey, "Decisions and Noise: The Scope of Evolutionary Synthesis and Dynamical Analysis", 2003.

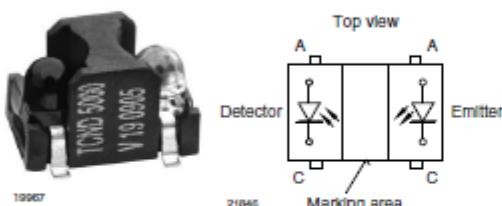
ANEXO A
DATASHEET DEL SENSOR TCND5000



TCND5000

Vishay Semiconductors

Reflective Optical Sensor with PIN Photodiode Output



DESCRIPTION

The TCND5000 is a reflective sensor that includes an infrared emitter and pin photodiode in a surface mount package which blocks visible light.

FEATURES

- Package type: surface mount
- Detector type: pin photodiode
- Dimensions (L x W x H in mm): 6 x 4.3 x 3.75
- Peak operating distance: 6 mm
- Operating range within > 20 % relative collector current: 2 mm to 25 mm
- Typical output current under test: $I_{out} > 0.11 \mu\text{A}$
- Daylight blocking filter
- High linearity
- Emitter wavelength: 940 nm
- Lead (Pb)-free soldering released
- Moisture sensitivity level (MSL): 4
- Compliant to RoHS Directive 2002/95/EC and in accordance to WEEE 2002/96/EC



APPLICATIONS

- Proximity sensor
- Object sensor
- Motion sensor
- Touch key

PRODUCT SUMMARY

PART NUMBER	DISTANCE FOR MAXIMUM CTR _{rel} ⁽¹⁾ (mm)	DISTANCE RANGE FOR RELATIVE I _{out} > 20 % (mm)	TYPICAL OUTPUT CURRENT UNDER TEST ⁽²⁾ (mA)	DAYLIGHT BLOCKING FILTER INTEGRATED
TCND5000	6	2 to 25	0.15	Yes

Notes

(1) CTR: current transfer ratio, I_{out}/I_{in}

(2) Conditions like in table basic characteristics/sensors

ORDERING INFORMATION

ORDERING CODE	PACKAGING	VOLUME ⁽¹⁾	REMARKS
TCND5000	Tape and reel	MOQ: 2000 pcs, 2000 pcs/reel	Drypack

Note

(1) MOQ: minimum order quantity

ABSOLUTE MAXIMUM RATINGS ($T_{amb} = 25^\circ\text{C}$, unless otherwise specified)

PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
INPUT (EMITTER)				
Reverse voltage		V_R	5	V
Forward current		I_F	100	mA
Peak forward current	$t_p = 50 \mu\text{s}, t = 2 \text{ ms}, T_{amb} \leq 25^\circ\text{C}$	I_{FM}	500	mA
Power dissipation		P_V	190	mW
Junction temperature		T_J	100	°C

TCND5000

Vishay Semiconductors Reflective Optical Sensor with PIN
Photodiode Output



ABSOLUTE MAXIMUM RATINGS ($T_{amb} = 25^{\circ}\text{C}$, unless otherwise specified)				
PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
OUTPUT (DETECTOR)				
Reverse voltage		V_R	60	V
Power dissipation		P_V	75	mW
Junction temperature		T_J	100	$^{\circ}\text{C}$
SENSOR				
Ambient temperature range		T_{amb}	- 40 to + 85	$^{\circ}\text{C}$
Storage temperature range		T_{stg}	- 40 to + 100	$^{\circ}\text{C}$
Soldering temperature	acc. fig. 14	T_{sd}	260	$^{\circ}\text{C}$

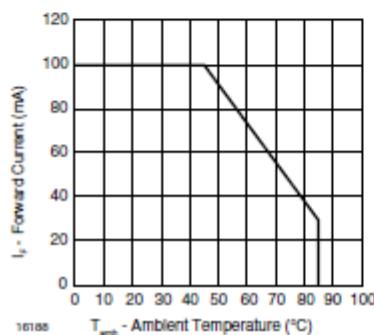
ABSOLUTE MAXIMUM RATINGS

Fig. 1 - Forward Current Limit vs. Ambient Temperature

BASIC CHARACTERISTICS ($T_{amb} = 25^{\circ}\text{C}$, unless otherwise specified)						
PARAMETER	TEST CONDITION	SYMBOL	MIN.	TYP.	MAX.	UNIT
INPUT (EMITTER) (*)						
Forward voltage	$I_F = 50 \text{ mA}, t_p = 20 \text{ ms}$	V_F		1.2	1.5	V
Temperature coefficient of V_F	$I_F = 1 \text{ mA}$	TK_{VF}		- 1.3		mV/K
Reverse current	$V_R = 5 \text{ V}$	I_R		10		μA
Junction capacitance	$V_R = 0 \text{ V}, f = 1 \text{ MHz}, E = 0 \text{ lx}$	C_J	25			pF
Radiant intensity	$I_F = 20 \text{ mA}, t_p = 20 \text{ ms}$	I_g	7	75		mW/sr
Angle of half intensity		φ		± 12		deg
Peak wavelength	$I_F = 100 \text{ mA}$	λ_P	930	940		nm
Spectral bandwidth	$I_F = 100 \text{ mA}$	$\Delta\lambda$		50		nm
Temperature coefficient of λ_P	$I_F = 100 \text{ mA}$	$\text{TK}_{\lambda P}$		0.2		nm/K
Rise time	$I_F = 100 \text{ mA}$	t_r		800		ns
Fall time	$I_F = 100 \text{ mA}$	t_f		800		ns
Virtual source diameter	Method: 63 % encircled energy	d		1.2		mm



TCND5000

Reflective Optical Sensor with PIN Vishay Semiconductors
Photodiode Output

BASIC CHARACTERISTICS ($T_{amb} = 25^\circ C$, unless otherwise specified)						
PARAMETER	TEST CONDITION	SYMBOL	MIN.	TYP.	MAX.	UNIT
OUTPUT (DETECTOR) [2]						
Forward voltage	$I_F = 50 \text{ mA}$	V_F		1	1.3	V
Breakdown voltage	$I_H = 100 \mu\text{A}$	V_{BR}	60			V
Reverse dark current	$V_R = 10 \text{ V}, E = 0 \text{ lx}$	I_{RD}		1	10	nA
Diode capacitance	$V_R = 5 \text{ V}, f = 1 \text{ MHz}, E = 0 \text{ lx}$	C_D		1.8		pF
Reverse light current	$E_g = 1 \text{ mW/cm}^2, \lambda = 950 \text{ nm}, V_R = 5 \text{ V}$	I_{RL}		12		μA
Temperature coefficient of I_{RL}	$\lambda = 870 \text{ nm}, V_R = 5 \text{ V}$	$TK_{I_{RL}}$		0.2		%/K
Angle of half intensity		φ		± 15		deg
Wavelength of peak sensitivity		λ_P		930		nm
Range of spectral bandwidth		$\lambda_{0.5}$		840 to 1050		nm
SENSOR						
Reverse Light Current	$V_R = 2.5 \text{ V}, I_F = 20 \text{ mA}, D = 30 \text{ mm}, \text{reflective mode: see figure 2}$	I_{RL}	110			nA

Note

[1] See figures 2 to 8 accordingly

[2] See figures 9 to 12 accordingly

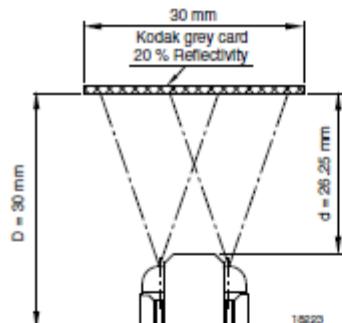


Fig. 2 - Test Circuit

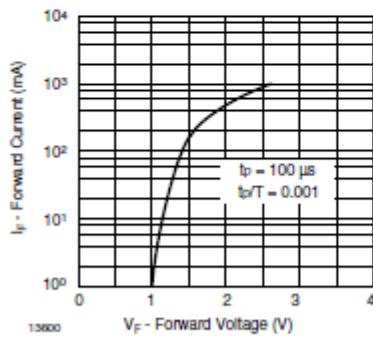
BASIC CHARACTERISTICS ($T_{amb} = 25^\circ C$, unless otherwise specified)

Fig. 3 - Forward Current vs. Forward Voltage

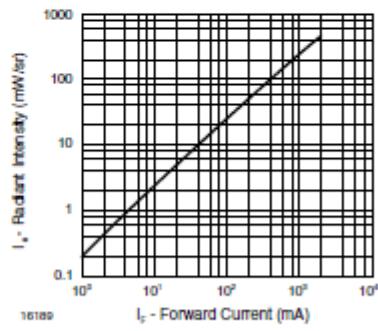


Fig. 4 - Radiant Intensity vs. Forward Current

TCND5000

Vishay Semiconductors Reflective Optical Sensor with PIN
Photodiode Output

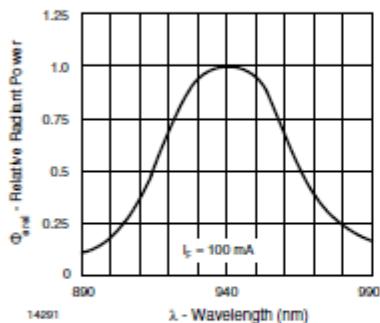


Fig. 5 - Relative Radiant Power vs. Wavelength

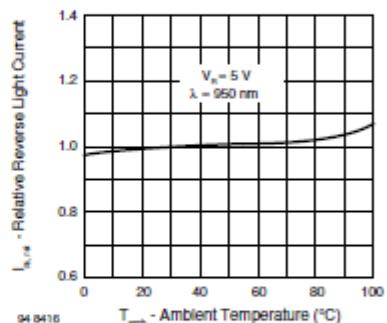


Fig. 8 - Relative Reverse Light Current vs. Ambient Temperature

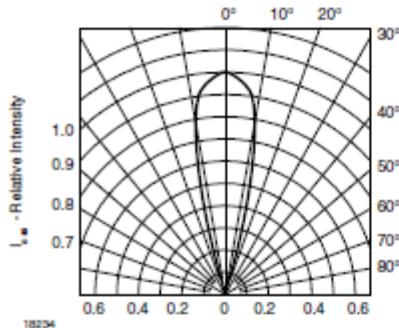


Fig. 6 - Relative Radiant Intensity vs. Angular Displacement

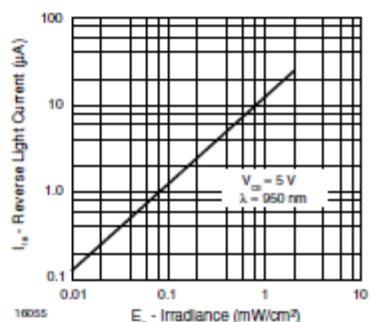


Fig. 9 - Reverse Light Current vs. Irradiance

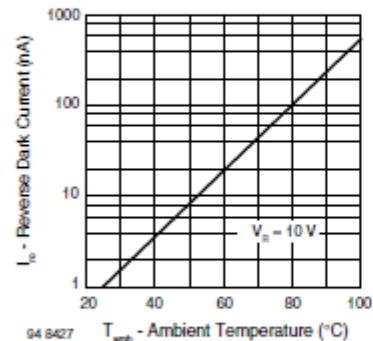


Fig. 7 - Reverse Dark Current vs. Ambient Temperature

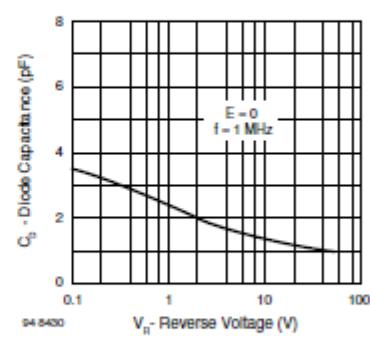


Fig. 10 - Diode Capacitance vs. Reverse Voltage



TCND5000

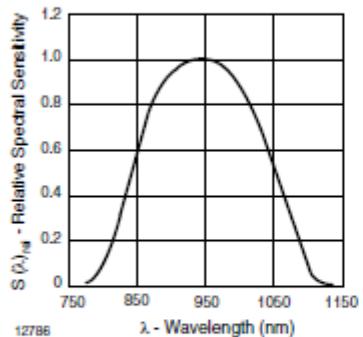
Reflective Optical Sensor with PIN Vishay Semiconductors
Photodiode Output

Fig. 11 - Relative Spectral Sensitivity vs. Wavelength

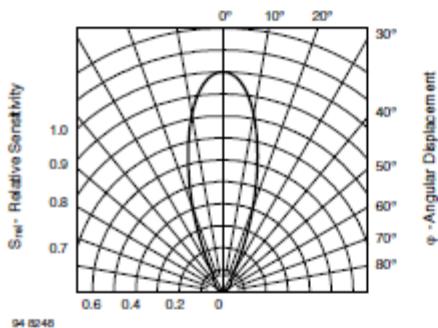


Fig. 12 - Relative Radiant Sensitivity vs. Angular Displacement

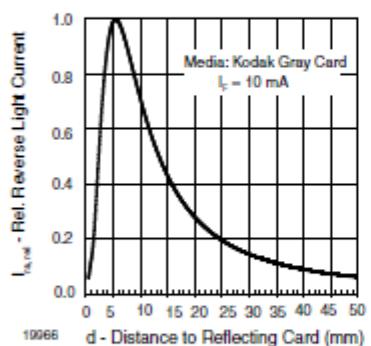


Fig. 13 - Relative Reverse Light Current vs. Distance



TCND5000

Vishay Semiconductors Reflective Optical Sensor with PIN
Photodiode Output

PRECAUTIONS FOR USE

1. Over-current-proof

Customer must apply resistors for protection, otherwise slight voltage shift will cause big current change (Burn out will happen).

2. Storage

2.1 Storage temperature and rel. humidity conditions are: 5 °C to 30 °C, RH 60 %

2.2 Floor life must not exceed 72 h, acc. to JEDEC level 4, J-STD-020.

Once the package is opened, the products should be used within 72 h. Otherwise, they should be kept in a damp proof box with desiccant.

Considering tape life, we suggest to use products within one year from production date.

2.3 If opened more than 72 h in an atmosphere 5 °C to 30 °C, RH 60 %, devices should be treated at 60 °C ± 5 °C for 15 h.

2.4 If humidity indicator in the package shows pink color (normal blue), then devices should be treated with the same conditions as 2.3

REFLOW SOLDER PROFILES

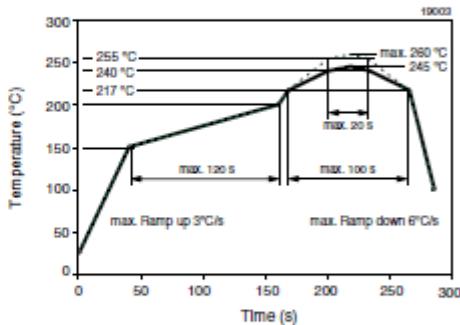


Fig. 14 - Lead (Pb)-Free Reflow Solder Profile

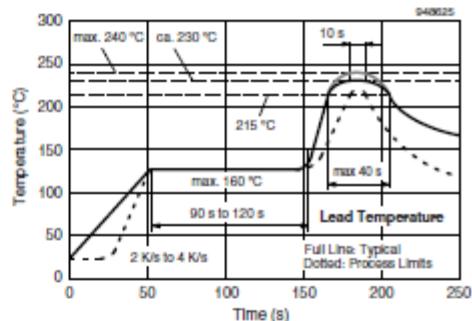


Fig. 15 - Lead Tin (SnPb) Reflow Solder Profile

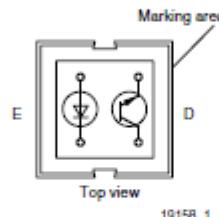
ANEXO B
DATASHEET DEL SENSOR CNY70



CNY70

Vishay Semiconductors

Reflective Optical Sensor with Transistor Output



FEATURES

- Package type: leaded
- Detector type: phototransistor
- Dimensions (L x W x H in mm): 7 x 7 x 6
- Peak operating distance: < 0.5 mm
- Operating range within > 20 % relative collector current: 0 mm to 5 mm
- Typical output current under test: $I_C = 1 \text{ mA}$
- Emitter wavelength: 950 nm
- Daylight blocking filter
- Lead (Pb)-free soldering released
- Compliant to RoHS directive 2002/95/EC and in accordance to WEEE 2002/96/EC



DESCRIPTION

The CNY70 is a reflective sensor that includes an infrared emitter and phototransistor in a leaded package which blocks visible light.

APPLICATIONS

- Optoelectronic scanning and switching devices i.e., Index sensing, coded disk scanning etc. (optoelectronic encoder assemblies).

PRODUCT SUMMARY

PART NUMBER	DISTANCE FOR MAXIMUM CTR _{rel} ⁽¹⁾ (mm)	DISTANCE RANGE FOR RELATIVE I _{out} > 20 % (mm)	TYPICAL OUTPUT CURRENT UNDER TEST ⁽²⁾ (mA)	DAYLIGHT BLOCKING FILTER INTEGRATED
CNY70	0	0 to 5	1	Yes

Notes

(¹) CTR: current transfer ratio, I_{out}/I_{in}

(²) Conditions like in table basic characteristics/sensors

ORDERING INFORMATION

ORDERING CODE	PACKAGING	VOLUME ⁽¹⁾	REMARKS
CNY70	Tube	MOQ: 4000 pcs, 80 pcs/tube	-

Note

(¹) MOQ: minimum order quantity

ABSOLUTE MAXIMUM RATINGS⁽¹⁾

PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
COUPLER				
Total power dissipation	$T_{amb} \leq 25^\circ\text{C}$	P_{tot}	200	mW
Ambient temperature range		T_{amb}	-40 to +85	°C
Storage temperature range		T_{stg}	-40 to +100	°C
Soldering temperature	Distance to case 2 mm, $t \leq 5 \text{ s}$	T_{sd}	260	°C
INPUT (EMITTER)				
Reverse voltage		V_R	5	V
Forward current		I_F	50	mA
Forward surge current	$t_p \leq 10 \mu\text{s}$	I_{FSM}	3	A
Power dissipation	$T_{amb} \leq 25^\circ\text{C}$	P_V	100	mW
Junction temperature		T_J	100	°C

CNY70

Vishay Semiconductors

Reflective Optical Sensor with
Transistor Output

ABSOLUTE MAXIMUM RATINGS⁽¹⁾				
PARAMETER	TEST CONDITION	SYMBOL	VALUE	UNIT
OUTPUT (DETECTOR)				
Collector emitter voltage		V_{CEO}	32	V
Emitter collector voltage		V_{ECO}	7	V
Collector current		I_C	50	mA
Power dissipation	$T_{amb} \leq 25^{\circ}\text{C}$	P_V	100	mW
Junction temperature		T_J	100	$^{\circ}\text{C}$

Note

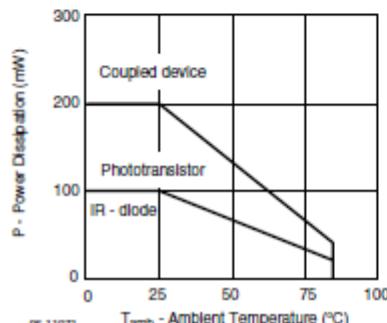
(1) $T_{amb} = 25^{\circ}\text{C}$, unless otherwise specified**ABSOLUTE MAXIMUM RATINGS**

Fig. 1 - Power Dissipation vs. Ambient Temperature

BASIC CHARACTERISTICS⁽¹⁾						
PARAMETER	TEST CONDITION	SYMBOL	MIN.	TYP.	MAX.	UNIT
COUPLER						
Collector current	$V_{CE} = 5\text{ V}$, $I_F = 20\text{ mA}$, $d = 0.3\text{ mm}$ (figure 1)	I_C ⁽²⁾	0.3	1.0		mA
Cross talk current	$V_{CE} = 5\text{ V}$, $I_F = 20\text{ mA}$, (figure 2)	I_{CX} ⁽³⁾			600	nA
Collector emitter saturation voltage	$I_F = 20\text{ mA}$, $I_C = 0.1\text{ mA}$, $d = 0.3\text{ mm}$ (figure 1)	V_{CEsat} ⁽²⁾			0.3	V
INPUT (EMITTER)						
Forward voltage	$I_F = 50\text{ mA}$	V_F		1.25	1.6	V
Radiant intensity	$I_F = 50\text{ mA}$, $t_p = 20\text{ ms}$	I_0			7.5	mW/sr
Peak wavelength	$I_F = 100\text{ mA}$	λ_P	940			nm
Virtual source diameter	Method: 63 % encircled energy	d		1.2		mm
OUTPUT (DETECTOR)						
Collector emitter voltage	$I_C = 1\text{ mA}$	V_{CEO}	32			V
Emitter collector voltage	$I_E = 100\text{ }\mu\text{A}$	V_{ECO}	5			V
Collector dark current	$V_{CE} = 20\text{ V}$, $I_F = 0\text{ A}$, $E = 0\text{ lx}$	I_{CEO}			200	nA

Notes

(1) $T_{amb} = 25^{\circ}\text{C}$, unless otherwise specified

(2) Measured with the "Kodak neutral test card", white side with 90 % diffuse reflectance

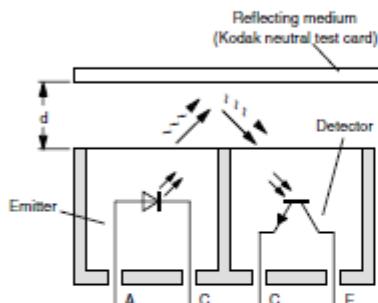
(3) Measured without reflecting medium



CNY70

Reflective Optical Sensor with
Transistor Output

Vishay Semiconductors



96 10008

Fig. 2 - Pulse diagram

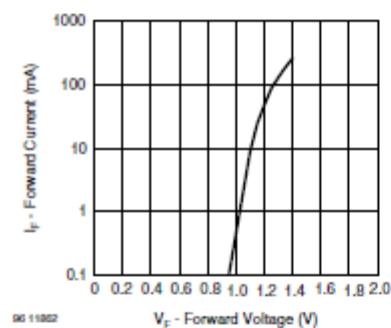
BASIC CHARACTERISTICS $T_{amb} = 25^\circ\text{C}$, unless otherwise specified

Fig. 3 - Forward Current vs. Forward Voltage

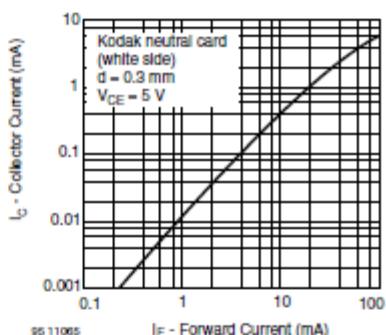


Fig. 5 - Collector Current vs. Forward Current

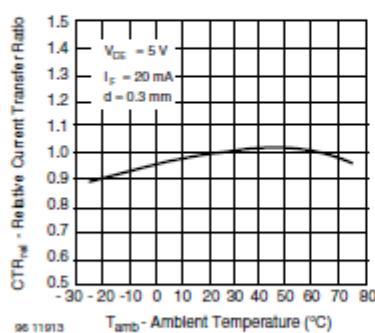


Fig. 4 - Relative Current Transfer Ratio vs. Ambient Temperature

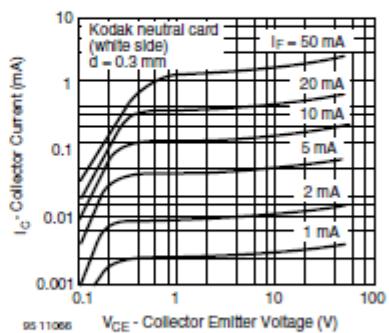


Fig. 6 - Collector Current vs. Collector Emitter Voltage

CNY70

Vishay Semiconductors

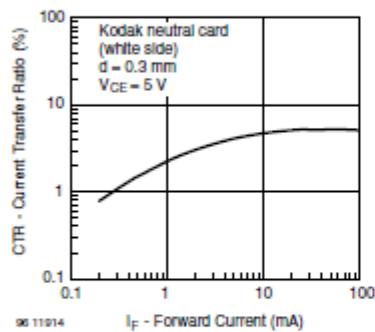
Reflective Optical Sensor with
Transistor Output

Fig. 7 - Current Transfer Ratio vs. Forward Current

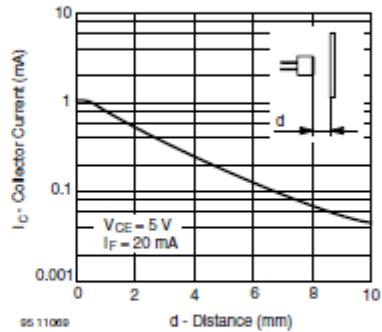


Fig. 9 - Collector Current vs. Distance

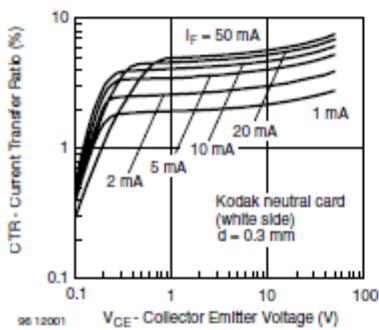


Fig. 8 - Current Transfer Ratio vs. Collector Emitter Voltage

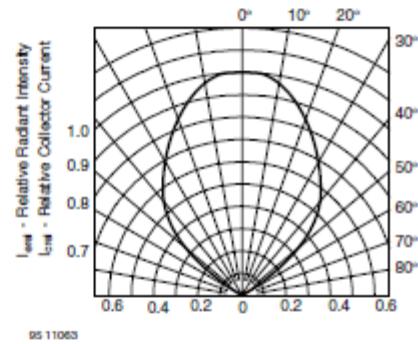


Fig. 10 - Relative Radiant Intensity/Collector Current vs. Angular Displacement

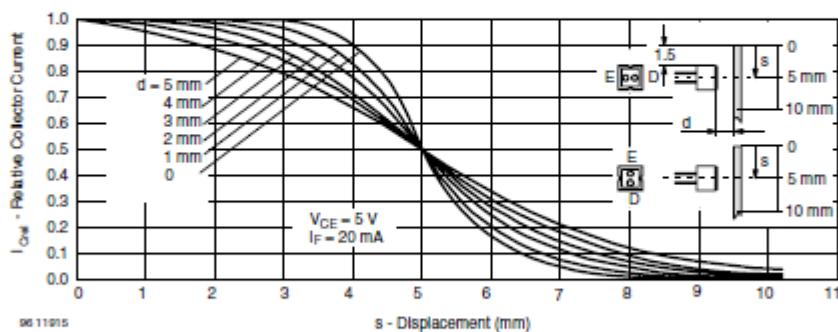


Fig. 11 - Relative Collector Current vs. Displacement

ANEXO C

**NOTA DE APLICACIÓN – SENSORES REFLEXIVOS VISHAY
SEMICONDUCTORS**

Application of Optical Sensors

Vishay Semiconductors



Optical Sensors - Reflective

Vishay is a leading manufacturer of optical sensors. These sensors integrate an infrared emitter and photo detector in a single package. The most common types of optical sensors are transmissive and reflective sensors.

Transmissive sensors, also called interrupter sensors, incorporate an infrared emitter and photo detector that face each other as shown in Figure 1. When an object is located between the emitter and detector in the sensing path, it interrupts or breaks the optical beam of the emitter. The amount of light energy reaching the detector is reduced. This change in light energy or photo current is used to affect an event in the application.

Reflective sensors incorporate an infrared emitter and photo detector adjacent to each other as shown in Figure 2. When an object is in the sensing area, the emitted light is reflected back towards the photo detector, the amount of light energy reaching the detector increases. This change in light energy or photo current is similarly used as an input signal in the application.

This application note describes the proper use of Vishay's reflective sensors. It describes several factors that must be considered when using a reflective sensor. Vishay manufactures many reflective sensors in lead and surface mount packages. One is just right for your application. Should you have any design questions, Vishay's Application Engineers are ready to assist you.

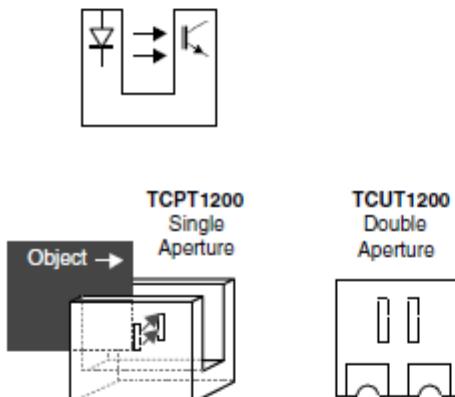


Figure 1.

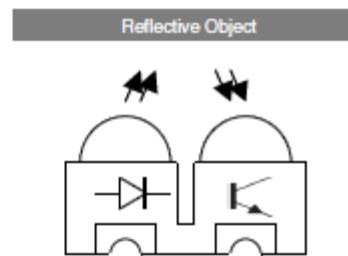


Figure 2.

Datasheet Parameter Values

The datasheets of each sensor include the absolute maximum ratings, and electrical and optical characteristics. The absolute maximum ratings of the emitter, detector and the sensor combined are provided. Maximum values for parameters like reverse and forward voltage, collector current, power dissipation, and ambient and storage temperatures are defined. The reflective sensors must be operated within these limits. In practice, applications should be designed so that there is large margin between the operating conditions and the absolute maximum ratings. The elec-

trical and optical characteristics indicate the performance of the sensor under specific operating conditions. Generally, the minimum and/or maximum values are provided. These values are guaranteed and are tested during the manufacture of the sensor. Typical values, while sometimes provided, should only be used as a guide in the design process. They may or may not be tested during the manufacturing process and are not guaranteed. Table 2 at the end of this note provides the symbol, parameter and definition of data found in reflective sensor datasheets.



Application of Optical Sensors

Vishay Semiconductors

Reflective Materials

The reflective sensor parameter values are measured using a metal mirror or an industry-standard reference surface called the Kodak neutral card also known as the gray or white card. The white side of the card has a reflection factor of 90 % while the gray side has a factor of 18 %. To learn more about the Kodak neutral card refer to Kodak's publication No. Q-13, CAT 1527654. Table 1 shows the relative values of measured reflection of a number of materials. They

were measured with the TCRT1000, with a forward current of 20 mA, at distance where the collector current was highest and with a wavelength of 950 nm. While the TCRT1000 was used, these values apply to all reflective sensors under the same operating conditions. These measurements have important practical use when designing a reflective sensor application. The reflection of surfaces in the infrared range can vary significantly from that in the visible range.

Table 1. Relative collector current (or coupling factor) of thereflex sensors for reflection on various materials. Reference is the white side of the Kodak neutral card. The sensor is positioned perpendicular to the surface. The wavelength is 950 nm

Kodak neutral card		Plastics, glass	
White side (reference medium)	100 %	White PVC	90 %
Gray side	20 %	Gray PVC	11 %
Paper		Blue, green, yellow, red PVC	40 - 80 %
Typewriting paper	94 %	White polyethylene	90 %
Drawing card, white (Schoeller Durex)	100 %	White polystyrene	120 %
Card, light gray	67 %	Gray partinax	9 %
Envelope (beige)	100 %	Fiber glass board material	
Packing card (light brown)	84 %	Without copper coating	12 - 19 %
Newspaper paper	97 %	With copper coating on the reverse side	30 %
Pergament paper	30 - 42 %	Glass, 1 mm thick	9 %
Black on white typewriting paper		Plexiglass, 1 mm thick	10%
Drawing ink (Higgins, Pelikan, Rotring)	4 - 6 %	Metals	
Foil ink (Rotring)	50 %	Aluminum, bright	110 %
Fiber-tip pen (Edding 400)	10 %	Aluminum, black anodized	60 %
Fiber-tip pen, black (Stabilo)	76 %	Cast aluminum, matt	45 %
Photocopy	7 %	Copper, matt (not oxidized)	110 %
Plotter pen		Brass, bright	160 %
HP fiber-tip pen (0.3 mm)	84 %	Gold plating, matt	150 %
Black 24 needle printer (EPSON LQ-500)	28 %	Textiles	
Ink (Pelikan)	100 %	White cotton	110 %
Pencil, HB	26 %	Black velvet	1.5 %

Application of Optical Sensors

Vishay Semiconductors



Operating Range and Peak Operating Distance

The phototransistor collector current is also dependent on the distance of the reflecting material from the sensor. Figure 3 shows the relative collector current versus the distance of the material from the sensor for the TCRT1000. This curve is included in each reflective sensor datasheet. The data was recorded using the Kodak neutral card's 90 % diffuse reflecting surface. The distance was measured from the surface of the sensor. The emitter current, I_F , was held constant during the measurement. This curve is called the working diagram. The working diagram of all reflective sensors shows a maximum collector current at a certain distance. For greater distances, collector current decreases. The working diagram is an important input to the reflective sensor circuit design. Choosing an operating distance at or near the sensors maximum collector current will provide greater design flexibility.

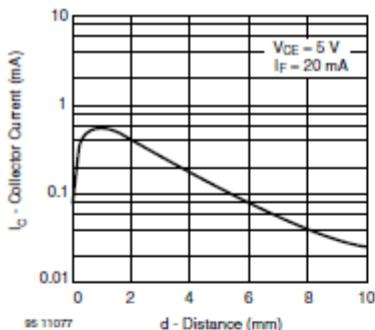


Figure 3. Collector Current vs. Distance

Switching Distance and Resolution

As an object moves over a reflective sensor the radiation reflected back to the detector changes gradually. For example, imagine a surface with an area high reflectivity and low reflectivity. As it moves over the sensor, Figure 4, the emitted radiation is reflected back to the detector. As the low-reflective surface moves into the sensing area of the detector, the collector current begins to drop-off. As this motion continues, a point is reached where the low-reflective surface completely envelops the detectors field of view. The edge of a sheet of paper, a black line on a shaft or the gaps in an encoding wheel will all see this gradual rise and fall in collector current. The switching distance, X_d , is the displacement relating to the width

from 90 % I_{C1} to 10 % of I_{C2} . This distance is predominantly dependent on the mechanical and optical design of the sensor, and the distance to the reflecting surface. The resolution of the sensor is the capability to recognize a change in reflectivity. If the width of a black line on a spinning shaft is less than X_d , then the change in collector current may not be large enough and recognition by the sensor uncertain. The shorter the switching distance, the higher the sensors resolution.

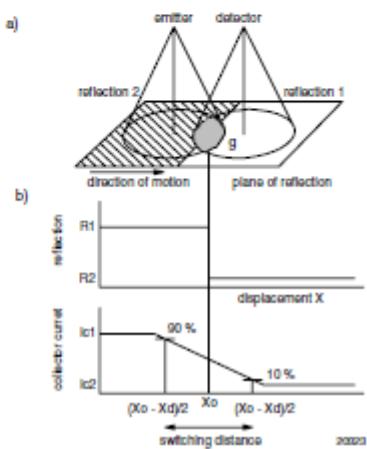


Figure 4. Abrupt reflection change with associated I_C curve

Cross Talk

The lowest light current that can be processed as a useful signal in the sensor's detector determines the weakest useable reflection and defines the sensitivity of the reflective sensor. This light current is determined by two parameters: cross talk and dark current. Whether the reflective sensor is lead-frame or PCB based, some of the emitted light will be internally reflected or channeled within the package to the detector. This is called optical cross talk. It is measured by operating the sensor without a reflective medium. While Vishay's sensors are designed to minimize crosstalk, the current must be considered when defining the circuit. The maximum cross talk current for each of Vishay's reflective sensors is specified in data sheets.

Reflection of the emitted light off of windows or surfaces surrounding the sensor is another source of cross talk to account for in the application design. In many applications this ambient crosstalk will be much higher than internal crosstalk of the sensor components and will determine signal to noise ratio or operating distance.



Application of Optical Sensors

Vishay Semiconductors

Dark Current

When a phototransistor is placed in the dark, or zero ambient illumination, and a voltage is applied from collector to emitter, a certain amount of current will flow. This current is called the dark current. It consists of the leakage current of the collector-base junction multiplied by the DC current gain of the transistor. The presence of this current prevents the phototransistor from being considered completely "off" or being an ideal "open switch". In datasheets, the dark current is described as being the maximum collector current permitted to flow at a given collector-emitter voltage. The dark current is a function of this voltage and temperature, Figure 5. Vishay phototransistors are tested at a V_{CE} applied voltage of 20 V. All reflective sensors which use a phototransistor specify a maximum dark current of 200 nA at 25 °C (typical 1 nA).

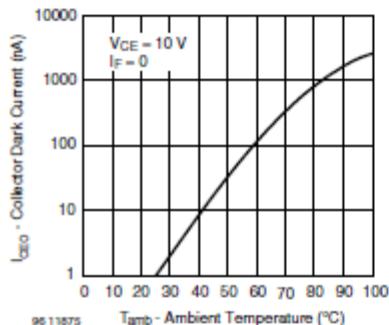


Figure 5. Collector Dark Current vs. Ambient Temperature

Temperature

Photo transistors and infrared emitting diodes are temperature dependent. As temperature increases, the light and dark current increases while emitter output decreases. An increase in the light current of the phototransistor is off-set by a decrease in the output of the emitter, Figure 6 and 7. Consequently, the change in the output of reflective sensors due to temperature change is comparatively small at less than 10 % from -10 °C to +70 °C, Figure 8. Because of this, it is not recommended to try to compensate for changes in temperature in the design of reflective sensor circuit.

Temperature also plays an important role in determining the emitter forward current in the application. As an example, for the TCRT1000, the maximum forward current at an ambient temperature of 25 °C is 50 mA. As shown in Figure 9, the forward current

must be reduced according to changes in the ambient temperature. If the ambient temperature is 60 °C, the maximum current is 25 mA. This means a current exceeding 25 mA must not flow into the emitter. In practice, the actual current should include a large safety margin and the lowest possible current should be used.

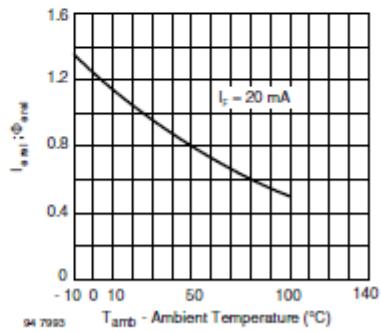


Figure 6. Rel. Radiant Intensity/Power vs. Ambient Temperature

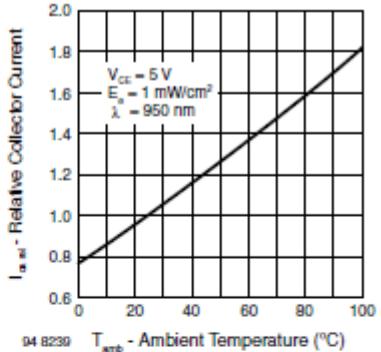


Figure 7. Rel. Collector Current vs. Ambient Temperature

Application of Optical Sensors

Vishay Semiconductors

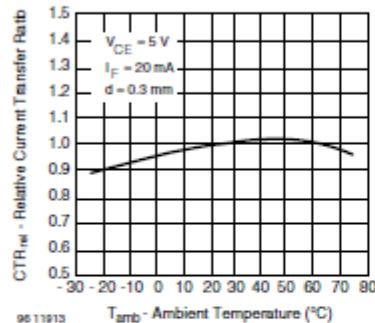


Figure 8. Rel. Current Transfer Ratio vs. Ambient Temperature

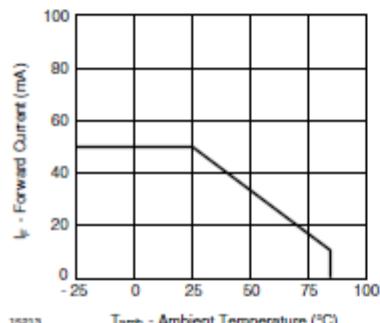


Figure 9. Forward Current vs. Ambient Temperature

Ambient Light

Ambient light can impair the sensitivity of the reflective sensor. Steady light falling directly on the detector reduces the sensor's sensitivity. Strong light can saturate the phototransistor and, in this condition, the sensor is blind. Varying ambient light results in incorrect signals and non-existent reflection changes. In applications where the ambient light source is known and relatively weak, in most cases it is enough to estimate the expected power of this light on the detector and to consider the result when defining the circuit. However, in many applications, it is difficult to precisely determine the ambient light and its effects. Ambient light is not only a problem when falling on the detector but can also be a problem when falling on the reflective surface. If the ambient light affects the object's and background's reflective factor in the same way, the ambient light effect can be ignored for low intensities. On the other hand, the object and

background's reflective factor can differ. The background may reflect ambient light much more than the object. In this case, ambient light may reduce the contrast between the object and the background and the object may not be detected. Conversely, the sensor may detect a non-targeted feature because it reflects the ambient light much more than the surroundings. Therefore, the influence of ambient light must be minimized by using optical filters, inspired mechanical design and, if necessary, AC operation. Vishay's reflective sensors are molded from epoxy that blocks visible light. Still, a large portion of sunlight is in the infrared. Locate or house the sensor so it is recessed to eliminate direct light. Pulsed operation can be helpful in some applications. AC operation is the most effective protection against ambient light.

Emitter Intensity

Emitter intensity depends largely on the forward current, I_F , optical efficiency of the lens and an internal reflector cup if included. The absolute maximum forward current of Vishay's TCRT1000, TCRT5000 and CNY70 is 50 mA, while the TCND5000 is 100 mA at an ambient temperature of 25 °C. The lower limit of the forward current of the emitter of any reflective sensor must be 5 mA minimum. If the forward current is too low, the optical output of the emitter will not be stable. A current limiting resistor is required. Without it, the current of the diode is theoretically limitless and the diode will burn out. The value of the current limiting resistor is calculated using the formula

$$R_L = (V_{CC} - V_F) / I_F$$

where the forward voltage of the emitter, V_F , typically 1.25 V, is subtracted from the supply voltage, V_{CC} , and divided by the forward current. Again, design in safety margin between actual operating conditions and the absolute maximum ratings. The external current limiting resistor defines the light intensity of the emitter. Driving the emitter with higher forward current to obtain larger reflected signal strength is not always be the best solution.



Application of Optical Sensors

Vishay Semiconductors

Response Time and Load Resistor

The speed of response of a phototransistor is dominated by the capacitance of the collector-base junction and the value of the load resistance. A phototransistor takes a certain amount of time to respond to sudden changes in light intensity. The response time is usually expressed by the rise time and fall time of the detector. As long as the light source driving the phototransistor is not intense enough to cause optical saturation, characterized by the storage of excess amounts of charge carriers in the base region, rise time equals fall time. If optical saturation occurs, fall time can become much larger than rise time. The selection of the load resistor, R_L , will also determine the amount of current-to-voltage conversion in the circuit. Reducing the value of R_L will result in a faster response time at the expense of a smaller voltage signal.

Degradation

End-users purchasing a reflective sensor want an accurate estimate of how long the sensor will last. Many will have minimum life requirements. Unlike most traditional light sources, infrared emitting diodes do not fail catastrophically. Instead, the light output degrades over time, Figure 10. Therefore the useful life of a reflective sensor can be defined by the time when it fails to provide sufficient light for the intended application. Infrared and visible light emitting diode life is often quoted to be 100000 hours but this is based on the average life span of a single, 5 mm epoxy encapsulated emitter. Vishay's reflective sensors also have a single emitter that is epoxy encapsulated. With some similarity, average life span can be considered comparable. As a rule-of-thumb, plan for 30 % degradation of the emitter over the life time of the sensor.

The three main causes of degradation are:

- A loss of efficiency caused by mechanical stress deforming the crystal structure
- A loss of optical coupling caused by delamination between epoxy and chip
- A loss of efficiency caused by thermal stress on the crystal structure

The rate of degradation or aging is affected by:

- Chip technology: GaAs and GaAlAs Double Hetero (DH) technologies result in lower rates, while GaAlAs and GaAlAs/GaAs technologies result in higher rates of aging
- Package technology: metal can packaging technologies result in lower rates, and epoxy packaging technologies result in higher rates of aging
- Chip size: The smaller the chip, the higher the current density. A higher current density results in faster aging

There are a number of ways to minimize emitter degradation or aging. First, minimize the junction temperature. As long as the junction temperature, T_J , is kept below 100 °C, heating of the pn-junction will cause no significant degradation. To reduce junction temperature, minimize the forward current and the ambient temperature. Second, in applications where there is temperature cycling, keep the forward current for the corresponding temperature well below that shown in Figure 9. This is especially important since degradation due to mechanical stress and delamination is potentially greater in epoxy-based sensors. Third, in applications where response time is not critical, pulse the emitter instead of constant current operation. Reflective sensor datasheets include a curve showing Total Power Dissipation versus Ambient Temperature. Use this curve as a guide to minimize degradation.

Vishay features state-of-the-art chip technologies and high quality standards in the assembly process resulting in low degradation rate of our sensor components.

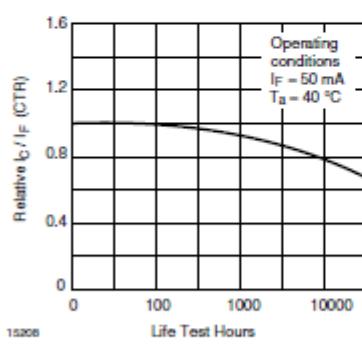


Figure 10.

Application of Optical Sensors

Vishay Semiconductors



Table 2.

Parameter	Symbol	Definition
V_R	Reverse voltage	The maximum permissible applied voltage to the anode of the LED such that the current flows in the reverse direction
I_F	Forward Current	The direct or continuous current flowing in the forward direction of a diode, from the anode to the cathode
I_{FSM}	Forward surge current	The maximum permissible surge or pulse current allowed for a specified temperature and period in the forward direction
P_V	Power dissipation	The maximum power that is consumed by the collector junction of a phototransistor
T_J	Junction temperature	The spatial mean value of the collector junction temperature during operation
V_{CEO}	Collector emitter voltage	The positive voltage applied to the collector of a phototransistor with the emitter at a reference potential and open base
V_{ECO}	Emitter collector voltage	The positive voltage applied to the emitter of a phototransistor with the collector at a reference potential and open base
I_C	Collector current	The current that flows to the collector junction of a phototransistor
T_{amb}	Ambient Temperature	The maximum permissible ambient temperature
T_{stg}	Storage Temperature	The maximum permissible storage temperature without an applied voltage
V_F	Forward voltage	The voltage drop across the diode in the forward direction when a specified forward current is applied
I_{CEO}	Collector dark current	The current leakage of the phototransistor when a specified bias voltage is applied so that the polarity of the collector is positive and that of the emitter is negative on condition that the illumination of the sensor is zero
I_{CX}	Cross talk current	The output current measured at a specified voltage and forward current when there is no reflective medium
V_{CEsat}	Collector emitter saturation voltage	The continuous voltage between the collector and emitter when the detector is in its "ON" state as measured with the Kodak neutral test card, white side
t_r	Rise time	Amount of time it takes the output voltage to go from 10 % of the lower specified value to 90 % of the upper specified value
t_f	Fall time	The time required for the output voltage to go from 90 % of the upper specified value to 10 % of the lower specified value

ANEXO D

DATASHEET DEL SERVOMOTOR PARALLAX 900-00008



Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

Parallax Continuous Rotation Servo (#900-00008)

The Parallax Standard Servo is ideal for robotics and basic movement projects.

Features

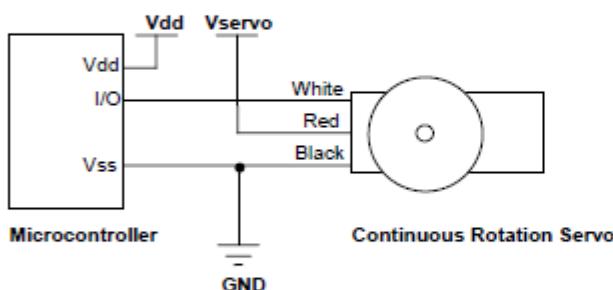
- Bidirectional continuous rotation
 - 0 to 50 RPM, with a linear response to PWM for easy ramping
 - Accepts four mounting screws
 - Easy to interface with any Parallax microcontroller or PWM-capable device
 - Very easy to control with PBASIC's or SX/B's PULSOUT commands
 - Manufactured for Parallax exclusively by Futaba



Technical Specifications

- Power requirements: 4 to 6 VDC
 - Maximum current draw: 140 +/- 50 mA at 6 VDC when operating in no load conditions
15 mA when in static state
 - Communication: pulse-width modulation; TTL/CMOS 3.3 to 5V
 - Dimensions: approx 2.2 x 0.8 x 1.6 in (5.58x 1.9 x 40.6 cm) excluding servo horn
 - Operating temperature range: 14 to 122°F (-10 to 50°C)
 - Weight: 1.50 oz (42.5 g)

Quick-Start Circuit



Vdd = microcontroller voltage supply

Vservo = 4 to 6 VDC, regulated or battery (See Board of Education Servo Header Connection Diagram, page 2)

I/O = PWM TTL or CMOS output signal, 3.3 to 5 V, not to exceed V_{servo} + 0.2 V

Device Information

The Parallax continuous rotation servo relies on pulse width modulation to control the speed and direction of the servo shaft. Before utilizing the servo in a project, it is important to calibrate the center position of the servo in order to define the point where the servo is at rest (see Calibration – "Center" the Servo below).

Specifications

Pin	Name	Description	Minimum	Typical	Maximum	Units
1 (White)	Signal	Input; TTL or CMOS	3.3	5.0	Vservo + 0.2	V
2 (Red)	Vservo	Power Supply	4.0	5.0	6.0*	V
3 (Black)	Vss	Ground		0		V

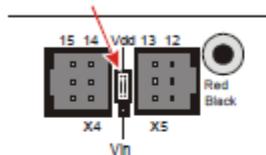
*See Board of Education Servo Header Connection Diagram, page 2.

Power Precautions

- Do not use this servo with an unregulated wall-mount supply. Such power supplies may deliver variable voltage far above the stated voltage.
- Do not power this servo through the BASIC Stamp Module's Vin pin, this can deliver voltages above the stated voltage. See the Board of Education Connection Diagram below for jumper settings.
- Servo current draw can spike while under peak load; be sure your application's regulator is prepared to supply adequate current for all servos used in combination.

Board of Education Servo Header Connection Diagram

When connecting the servo to the Board of Education Rev C or higher's servo header, be sure the jumper is set to Vdd as shown in the figure below. Failure to place the jumper at this setting can cause damage to your servo!



Calibration – "Center" the Servo

The servo has a potentiometer access port, allowing the user to adjust the servo to hold completely still when receiving a 1.5 ms pulse width. This is the value in the "center" of the range of control pulses the servo will accept.

To center the servo, program your host device to deliver a 1.5 ms pulse, continually refreshed every 20 ms. Sample calibration code is given below for all BASIC Stamp models, Spin for the Propeller, and SX/B for the SX chip. All are available for download from the 900-00008 product page at www.parallax.com.

Connect the servo to your microcontroller's I/O pin. The example programs below specify an I/O pin. Program

BASIC Stamp Calibration Code - for all BS2 models

- ✓ Connect the servo to BASIC Stamp 1/O pin P12, or update the ToServo PIN declaration.
- ✓ Run the program, and gently twist the potentiometer adjustment screw until the servo does not turn or vibrate. *NOTE: Calibrating the servo may take some patience. The potentiometer is very sensitive so a very light touch will be required.*

```
' {$STAMP BS2}
' {$PBASIC 2.5}

#SELECT $Stamp
#CASE BS2, BS2E, BS2PE      ' PULSOUT Duration units are 2 us for these models
    Center CON 750
#CASE BS2SX, BS2P, BS2PX    ' PULSOUT Duration units are 0.8 us for these models
    Center CON 1875
#ENDSELECT

ToServo PIN 12              ' connect servo to I/O pin P12, or change it here

DO
    PULSOUT ToServo, Center      ' ToServo pin outputs 1.5 ms pulse
    PAUSE 20                      ' refresh pulse every 20 milliseconds
LOOP
```

Propeller Chip Calibration Code – for P8X32A

- ✓ Download and unzip the Propeller code file from the 900-00008 product page.
- ✓ Connect the servo line to pin 0.
- ✓ Run the program CenterServo.spin, and gently twist the potentiometer adjustment screw until the servo does not turn or vibrate. *NOTE: Calibrating the servo may take some patience. The potentiometer is very sensitive so a very light touch will be required.*

```
CenterServo.spin

CON
_clkmode = xtal1 + pll16x          ' System clock + 80 MHz
_xinfreq = 5 000 000

PUB CenterServo | tInc, tc, tHa, t

ctrA[30..26] := #00100             ' Configure Counter A to NCO
ctrA[8..0] := 0

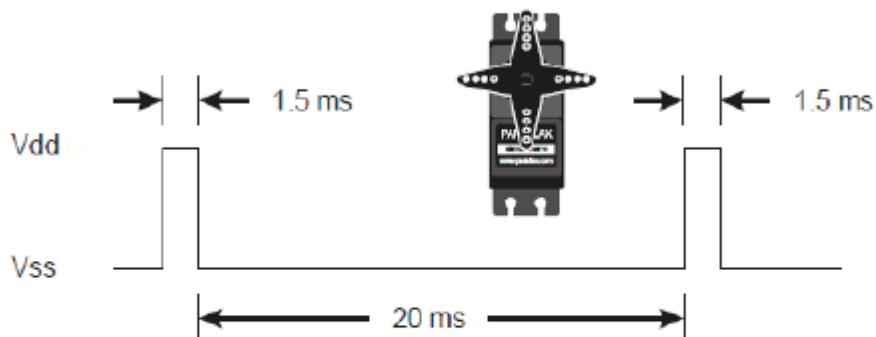
frqa := 1
dira[0]~~

' Set up cycle and high times
tInc := clkfreq/1 000 000
tc   := tInc * 21 500
tHa  := tInc * 1500
t    := cnt                         ' Mark counter time

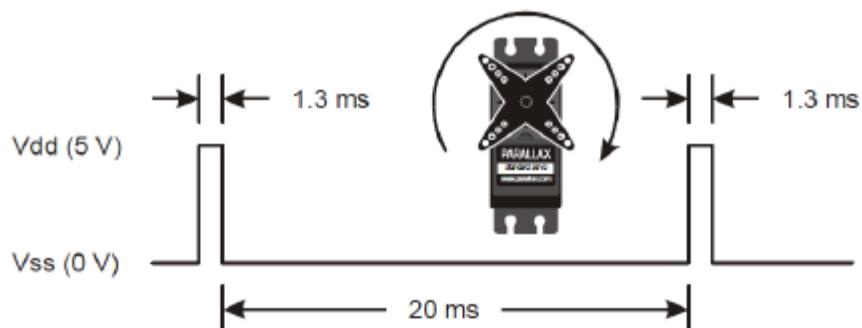
repeat
    phsa := -tHa
    t += tc                          ' Repeat PWM signal
    Set up the pulse
    Calculate next cycle repeat
    waitcnt(t)                      ' Wait for next cycle
```

Communication Protocol

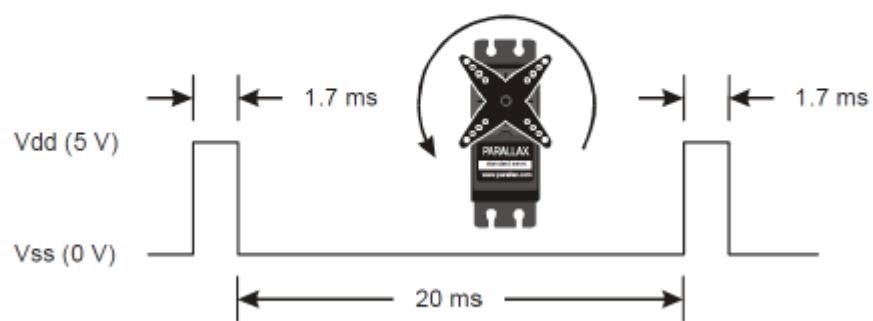
The Parallax Continuous Rotation Servo is controlled through pulse width modulation, where the rotational speed and direction are determined by the duration of the pulse. In order for smooth rotation, the servo needs a 20 ms pause between pulses. Below is a sample timing diagram for a centered servo:



As the length of the pulse decreases from 1.5 ms, the servo will gradually rotate faster in the clockwise direction, as can be seen in the figure below:



Likewise, as the length of the pulse increases from 1.5 ms, the servo will gradually rotate faster in the counter-clockwise direction, as can be seen in the figure below:



BASIC Stamp Programming Examples

PBASIC has a PULSOUT command that sets the I/O Pin to an output and sends a pulse of the specified *Duration*. Since the servo needs this pulse refreshed every 20 ms for continuous operation, the PULSOUT command is put in a counted FOR...NEXT loop to sustain continuous operation for the specified number of cycles.

PULSOUT Pin, Duration

Different BASIC Stamp modules use different units for the PULSOUT command's *Duration* argument. When adapting BS2 code to another BASIC Stamp model, you may need to make adjustments. The table below lists the PULSOUT ranges for each BASIC Stamp microcontroller. See the BASIC Stamp Manual or BASIC Stamp Editor Help for more information.

BASIC Stamp Module	1.3 ms	1.5 ms	1.7 ms
BS1	100	150	200
BS2, BS2e, BS2pe	500	750	1000
BS2sx, BS2px, BS2p24/40	1250	1875	2500

The example shown below for a BASIC Stamp 2 causes a servo connected to BASIC Stamp 1/0 pin 12 to first rotate full-speed clockwise for about 5 seconds, hold still for about 5 seconds, then rotate counterclockwise for 5 seconds.

```
' {$STAMP BS2}
' {$PBASIC 2.5}

counter VAR Word

FOR counter = 1 TO 100          ' Rotate clockwise for ~5 seconds
    PULSOUT 12, 650
    PAUSE 20

NEXT

FOR counter = 1 TO 100          ' Hold still for ~5 seconds
    PULSOUT 12, 750
    PAUSE 20

NEXT

FOR counter = 1 TO 100          ' Rotate counterclockwise for ~5 seconds
    PULSOUT 12, 650
    PAUSE 20

NEXT
```

For more examples with the BASIC Stamp 2, including 2-wheeled robot maneuvers and ramping, see "Robotics with the Boe-Bot" Chapter 4, available for free download from the 28132 product page at www.parallax.com.

Propeller Application

The program below uses counter modules to rotate the servo first clockwise at full speed for 2 seconds, then rests for 2 seconds, and rotates counterclockwise at full speed for another 2 seconds. This code can also be downloaded from the 900-00008 product page.

```
ServoRotation.spin

CON
  clkmode = xtall + pll16x           ' System clock + 80 MHz
  _xinfreq = 5_000_000

PUB CenterServo | tInc, tc, tCtr, tCw, tCcw, t

  ctra[30..26] := #00100             ' Configure Counter A to NCO
  ctra[8..0] := 0

  frqa := 1
  dira[0]~~

  tInc := clkfreq/1_000_000
  tc := tInc * 21_500
  tCtr := tInc * 1500
  tCw := tInc * 1300
  tCcw := tInc * 1700
  t := cnt
  : 1 µs increment
  : Low pulse
  : Center pulse = 1.5 ms
  : Clockwise fast = 1.3 ms
  : Counter-Clockwise fast = 1.7 ms
  : Mark counter time

repeat 100
  phsa := -tCw
  t += tC
  waitcnt(t)
  : Repeat PWM signal 100x
  : Set up clockwise fast pulse
  : Calculate next cycle repeat
  : Wait for next cycle (20 ms)

repeat 100
  phsa := -tCtr
  t += (tC + 200)
  waitcnt(t)
  : Repeat PWM signal 100x
  : Set up the center pulse
  : Calculate next cycle repeat
  : Wait for next cycle (20 ms)

repeat 100
  phsa := -tCcw
  t += (tC - 200)
  waitcnt(t)
  : Repeat PWM signal 100x
  : Set up counter-clockwise fast pulse
  : Calculate next cycle repeat
  : Wait for next cycle (20 ms)
```

ANEXO E

ALGORITMO NONDOMINATED SORTING GENETIC ALGORITHM-II (NSGA-II)

Fast Nondominated Sorting Approach

For the sake of clarity, we first describe a naive and slow procedure of sorting a population into different nondomination levels. Thereafter, we describe a fast approach.

In a naive approach, in order to identify solutions of the first nondominated front in a population of size N , each solution can be compared with every other solution in the population to find if it is dominated. This requires $O(MN)$ comparisons for each solution, where M is the number of objectives. When this process is continued to find all members of the first nondominated level in the population, the total complexity is $O(MN^2)$. At this stage, all individuals in the first nondominated front are found. In order to find the individuals in the next nondominated front, the solutions of the first front are discounted temporarily and the above procedure is repeated. In the worst case, the task of finding the second front also requires $O(MN^2)$ computations, particularly when $O(N)$ number of solutions belong to the second and higher nondominated levels. This argument is true for finding third and higher levels of nondomination. Thus, the worst case is when there are N fronts and there exists only one solution in each front. This requires an overall $O(MN^3)$ computations. Note that $O(N)$ storage is required for this procedure. In the following paragraph and equation shown at the bottom of the page, we describe a fast nondominated sorting approach which will require $O(MN^2)$ computations.

First, for each solution we calculate two entities: 1) domination count n_p , the number of solutions which dominate the solution p , and 2) S_p , a set of solutions that the solution p dominates. This requires $O(MN^2)$ comparisons.

All solutions in the first nondominated front will have their domination count as zero. Now, for each solution p with $n_p = 0$, we visit each member (q) of its set S_p and reduce its domination count by one. In doing so, if for any member q the domination count becomes zero, we put it in a separate list Q . These

members belong to the second nondominated front. Now, the above procedure is continued with each member of Q and the third front is identified. This process continues until all fronts are identified.

For each solution p in the second or higher level of nondomination, the domination count n_p can be at most $N - 1$. Thus, each solution p will be visited at most $N - 1$ times before its domination count becomes zero. At this point, the solution is assigned a nondomination level and will never be visited again. Since there are at most $N - 1$ such solutions, the total complexity is $O(N^2)$. Thus, the overall complexity of the procedure is $O(MN^2)$. Another way to calculate this complexity is to realize that the body of the first inner loop (for each $p \in \mathcal{F}_i$) is executed exactly N times as each individual can be the member of at most one front and the second inner loop (for each $q \in S_p$) can be executed at maximum $(N - 1)$ times for each individual [each individual dominates $(N - 1)$ individuals at maximum and each domination check requires at most M comparisons] results in the overall $O(MN^2)$ computations. It is important to note that although the time complexity has reduced to $O(MN^2)$, the storage requirement has increased to $O(N^2)$.

B. Diversity Preservation

We mentioned earlier that, along with convergence to the Pareto-optimal set, it is also desired that an EA maintains a good spread of solutions in the obtained set of solutions. The original NSGA used the well-known sharing function approach, which has been found to maintain sustainable diversity in a population with appropriate setting of its associated parameters. The sharing function method involves a sharing parameter σ_{share} , which sets the extent of sharing desired in a problem. This parameter is related to the distance metric chosen to calculate the proximity measure between two population members. The pa-

fast-non-dominated-sort(P)	
for each $p \in P$	
$S_p = \emptyset$	
$n_p = 0$	
for each $q \in P$	
if $(p \prec q)$ then	If p dominates q
$S_p = S_p \cup \{q\}$	Add q to the set of solutions dominated by p
else if $(q \prec p)$ then	
$n_p = n_p + 1$	Increment the domination counter of p
if $n_p = 0$ then	p belongs to the first front
$p_{rank} = 1$	
$\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$	
<i>i</i> = 1	Initialize the front counter
while $\mathcal{F}_i \neq \emptyset$	
$Q = \emptyset$	Used to store the members of the next front
for each $p \in \mathcal{F}_i$	
for each $q \in S_p$	
$n_q = n_q - 1$	
if $n_q = 0$ then	q belongs to the next front
$q_{rank} = i + 1$	
$Q = Q \cup \{q\}$	
<i>i</i> = <i>i</i> + 1	
$\mathcal{F}_i = Q$	

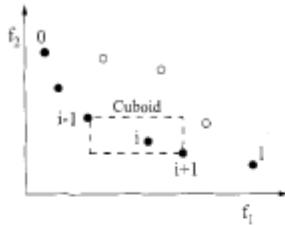


Fig. 1. Crowding-distance calculation. Points marked in filled circles are solutions of the same nondominated front.

parameter σ_{share} denotes the largest value of that distance metric within which any two solutions share each other's fitness. This parameter is usually set by the user, although there exist some guidelines. There are two difficulties with this sharing function approach.

- 1) The performance of the sharing function method in maintaining a spread of solutions depends largely on the chosen σ_{share} value.
- 2) Since each solution must be compared with all other solutions in the population, the overall complexity of the sharing function approach is $O(N^2)$.

In the proposed NSGA-II, we replace the sharing function approach with a crowded-comparison approach that eliminates both the above difficulties to some extent. The new approach does not require *any* user-defined parameter for maintaining diversity among population members. Also, the suggested approach has a better computational complexity. To describe this approach, we first define a density-estimation metric and then present the crowded-comparison operator.

1) Density Estimation: To get an estimate of the density of solutions surrounding a particular solution in the population, we calculate the average distance of two points on either side of this point along each of the objectives. This quantity $i_{distance}$ serves as an estimate of the perimeter of the cuboid formed by using the nearest neighbors as the vertices (call this the *crowding distance*). In Fig. 1, the crowding distance of the i th solution in its front (marked with solid circles) is the average side length of the cuboid (shown with a dashed box).

The crowding-distance computation requires sorting the population according to each objective function value in ascending order of magnitude. Thereafter, for each objective function, the boundary solutions (solutions with smallest and largest function values) are assigned an infinite distance value. All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. This calculation is continued with other objective

functions. The overall crowding-distance value is calculated as the sum of individual distance values corresponding to each objective. Each objective function is normalized before calculating the crowding distance. The algorithm as shown at the bottom of the page outlines the crowding-distance computation procedure of all solutions in an nondominated set \mathcal{I} .

Here, $\mathcal{I}[i], m$ refers to the m th objective function value of the i th individual in the set \mathcal{I} and the parameters f_m^{\max} and f_m^{\min} are the maximum and minimum values of the m th objective function. The complexity of this procedure is governed by the sorting algorithm. Since M independent sortings of at most N solutions (when all population members are in one front \mathcal{I}) are involved, the above algorithm has $O(MN \log N)$ computational complexity.

After all population members in the set \mathcal{I} are assigned a distance metric, we can compare two solutions for their extent of proximity with other solutions. A solution with a smaller value of this distance measure is, in some sense, more crowded by other solutions. This is exactly what we compare in the proposed crowded-comparison operator, described below. Although Fig. 1 illustrates the crowding-distance computation for two objectives, the procedure is applicable to more than two objectives as well.

2) Crowded-Comparison Operator: The crowded-comparison operator (\prec_m) guides the selection process at the various stages of the algorithm toward a uniformly spread-out Pareto-optimal front. Assume that every individual i in the population has two attributes:

- 1) nondomination rank (i_{rank});
- 2) crowding distance ($i_{distance}$).

We now define a partial order \prec_m as

$$i \prec_m j \quad \text{if } (i_{rank} < j_{rank}) \\ \text{or } ((i_{rank} = j_{rank}) \text{ and } (i_{distance} > j_{distance}))$$

That is, between two solutions with differing nondomination ranks, we prefer the solution with the lower (better) rank. Otherwise, if both solutions belong to the same front, then we prefer the solution that is located in a lesser crowded region.

With these three new innovations—a fast nondominated sorting procedure, a fast crowded distance estimation procedure, and a simple crowded comparison operator, we are now ready to describe the NSGA-II algorithm.

C. Main Loop

Initially, a random parent population P_0 is created. The population is sorted based on the nondomination. Each solution is assigned a fitness (or rank) equal to its nondomination level (1

<u>crowding-distance-assignment(\mathcal{I})</u>	
$\mathcal{I} = [\mathcal{I}]$	number of solutions in \mathcal{I}
for each i , set $\mathcal{I}[i]_{distance} = 0$	initialize distance
for each objective m	
$\mathcal{I} = \text{sort}(\mathcal{I}, m)$	sort using each objective value
$\mathcal{I}[1]_{distance} = \mathcal{I}[l]_{distance} = \infty$	so that boundary points are always selected
for $i = 2$ to $(l-1)$	for all other points
$\mathcal{I}[i]_{distance} = \mathcal{I}[i]_{distance} + (\mathcal{I}[i+1], m - \mathcal{I}[i-1], m) / (f_m^{\max} - f_m^{\min})$	

is the best level, 2 is the next-best level, and so on). Thus, minimization of fitness is assumed. At first, the usual binary tournament selection, recombination, and mutation operators are used to create a offspring population Q_t of size N . Since elitism is introduced by comparing current population with previously found best nondominated solutions, the procedure is different after the initial generation. We first describe the t th generation of the proposed algorithm as shown at the bottom of the page.

The step-by-step procedure shows that NSGA-II algorithm is simple and straightforward. First, a combined population $R_t = P_t \cup Q_t$ is formed. The population R_t is of size $2N$. Then, the population R_t is sorted according to nondomination. Since all previous and current population members are included in R_t , elitism is ensured. Now, solutions belonging to the best non-dominated set \mathcal{F}_1 are of best solutions in the combined population and must be emphasized more than any other solution in the combined population. If the size of \mathcal{F}_1 is smaller than N , we definitely choose all members of the set \mathcal{F}_1 for the new population P_{t+1} . The remaining members of the population P_{t+1} are chosen from subsequent non-dominated fronts in the order of their ranking. Thus, solutions from the set \mathcal{F}_2 are chosen next, followed by solutions from the set \mathcal{F}_3 , and so on. This procedure is continued until no more sets can be accommodated. Say that the set \mathcal{F}_l is the last non-dominated set beyond which no other set can be accommodated. In general, the count of solutions in all sets from \mathcal{F}_1 to \mathcal{F}_l would be larger than the population size. To choose exactly N population members, we sort the solutions of the last front \mathcal{F}_l using the crowded-comparison operator \prec_n in descending order and choose the best solutions needed to fill all population slots. The NSGA-II procedure is also shown in Fig. 2. The new population P_{t+1} of size N is now used for selection, crossover, and mutation to create a new population Q_{t+1} of size N . It is important to note that we use a binary tournament selection operator but the selection criterion is now based on the crowded-comparison operator \prec_n . Since this operator requires both the rank and crowded distance of each solution in the population, we calculate these quantities while forming the population P_{t+1} , as shown in the above algorithm.

Consider the complexity of one iteration of the entire algorithm. The basic operations and their worst-case complexities are as follows:

- 1) nondominated sorting is $O(M(2N)^2)$;
- 2) crowding-distance assignment is $O(M(2N)\log(2N))$;
- 3) sorting on \prec_n is $O(2N\log(2N))$.

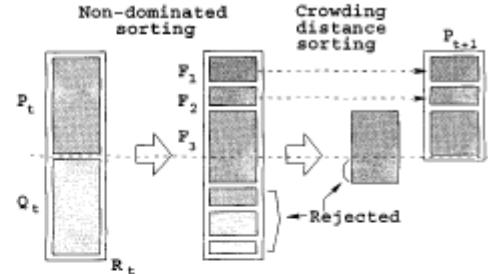


Fig. 2. NSGA-II procedure.

The overall complexity of the algorithm is $O(MN^2)$, which is governed by the nondominated sorting part of the algorithm. If performed carefully, the complete population of size $2N$ need not be sorted according to nondomination. As soon as the sorting procedure has found enough number of fronts to have N members in P_{t+1} , there is no reason to continue with the sorting procedure.

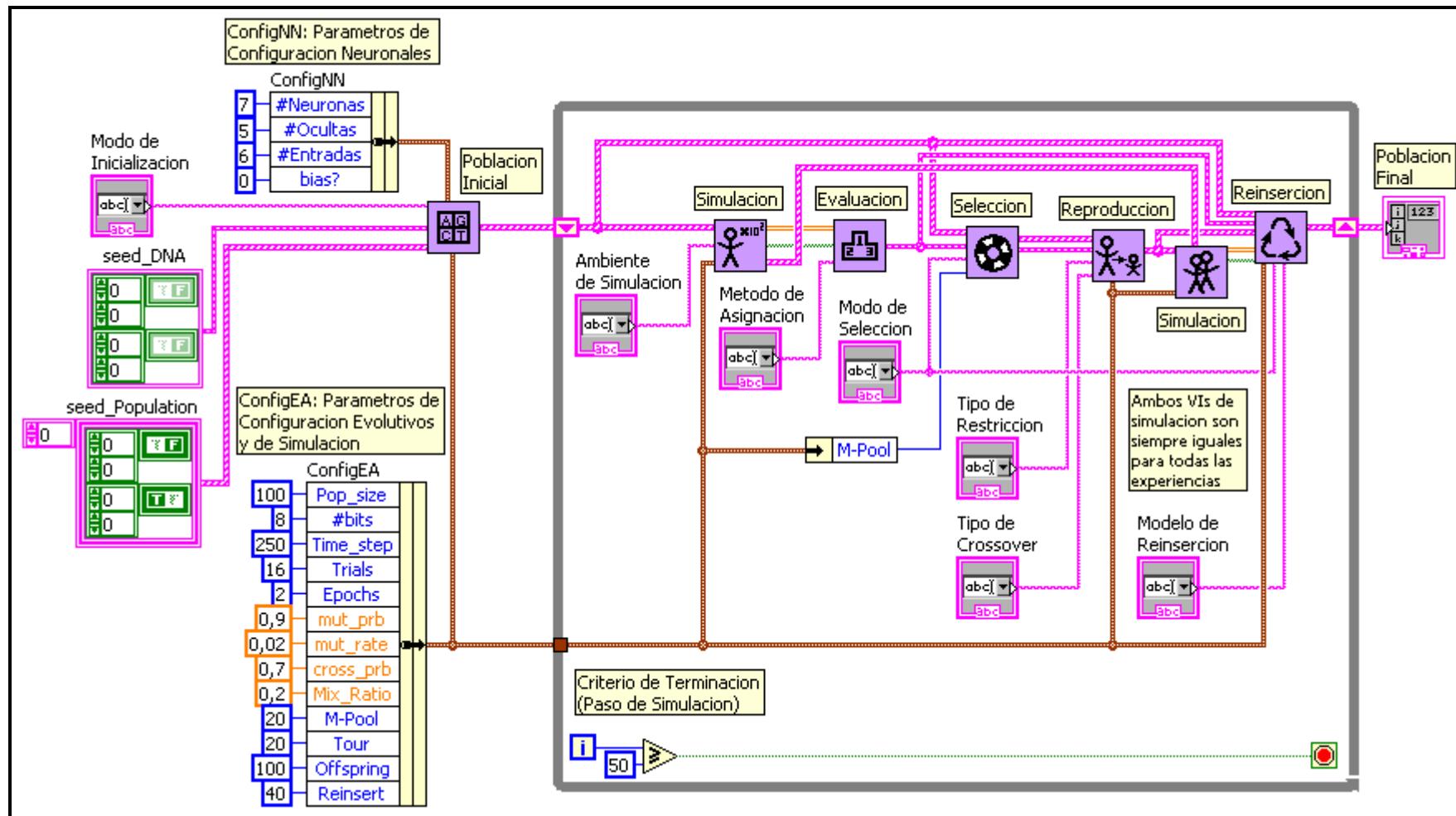
The diversity among non-dominated solutions is introduced by using the crowding comparison procedure, which is used in the tournament selection and during the population reduction phase. Since solutions compete with their crowding-distance (a measure of density of solutions in the neighborhood), no extra niching parameter (such as σ_{plane} needed in the NSGA) is required. Although the crowding distance is calculated in the objective function space, it can also be implemented in the parameter space, if so desired.

$R_t = P_t \cup Q_t$ $\mathcal{F} = \text{fast-non-dominated-sort}(R_t)$ $P_{t+1} = \emptyset$ and $i = 1$ until $ P_{t+1} + \mathcal{F}_i \leq N$ crowding-distance-assignment (\mathcal{F}_i) $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$ $i = i + 1$ Sort (\mathcal{F}_i , \prec_n) $P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - P_{t+1})]$ $Q_{t+1} = \text{make-new-pop}(P_{t+1})$ $t = t + 1$	combine parent and offspring population $\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \dots)$, all non-dominated fronts of R_t until the parent population is filled calculate crowding-distance in \mathcal{F}_i include i th non-dominated front in the parent pop check the next front for inclusion sort in descending order using \prec_n choose the first $(N - P_{t+1})$ elements of \mathcal{F}_i use selection, crossover and mutation to create a new population Q_{t+1} increment the generation counter
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

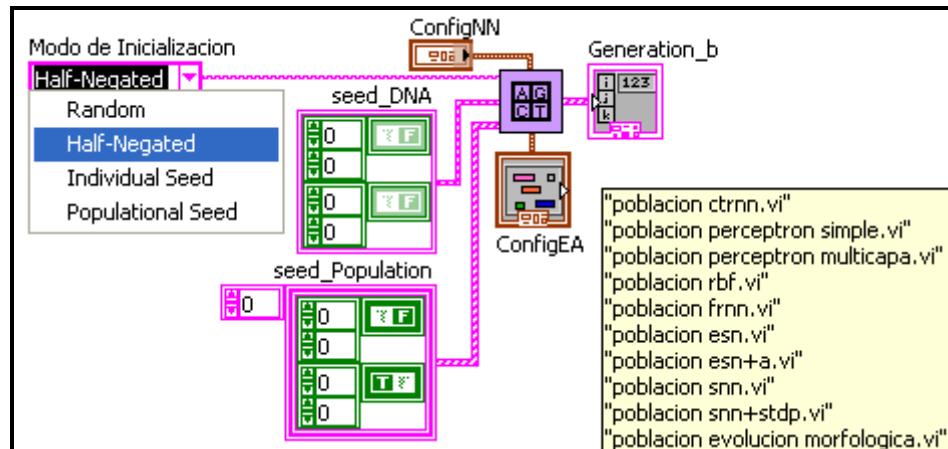
ANEXO F

DESARROLLO ALGORITMICO – OPERADORES EVOLUTIVOS

Algoritmo Evolutivo Básico

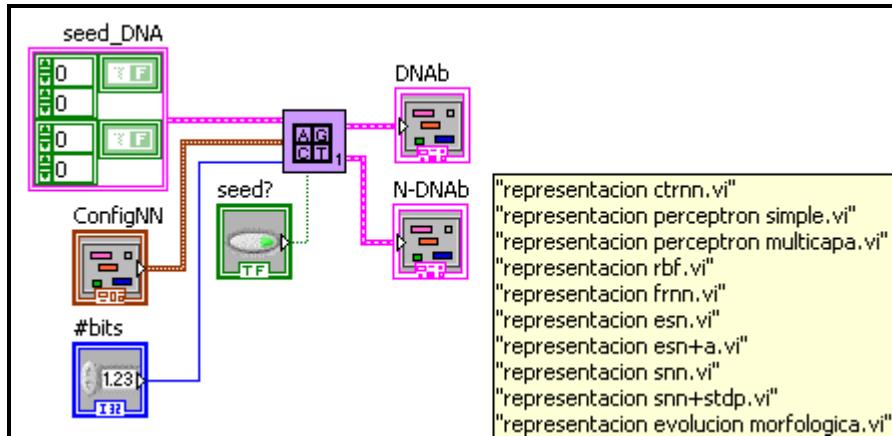


Población



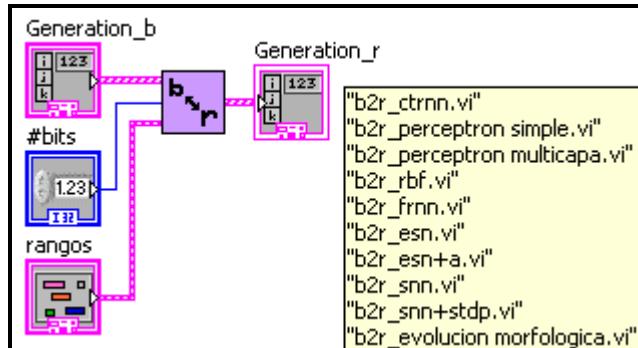
FUNCION		
Crea una nueva Población de cromosomas o copia una Población ya existente.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Modo de Inicializacion	Combo-box	<ul style="list-style-type: none"> • Random: Todos los cromosomas se generan aleatoriamente. • Half-Negated: La mitad de la Población se crea aleatoriamente, la mitad restante resulta de la negación lógica de la anterior. • Individual Seed: Copia repetida de un cromosoma ya existente. • Populational Seed: Copia de una Población ya existente.
Seed_DNA	Cluster (cromosoma)	Cromosoma que se copiará.
Seed_Population	Array de clusters	Población de cromosomas binarios que se copiará.
ConfigEA	Cluster (configuración)	Incluye parámetros como: "Pop_size" (tamaño de Población) y "#bits".
ConfigNN	Cluster (configuración)	Valores de configuración para la red neuronal.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Generation_b	Array de clusters	Población obtenida; representada internamente mediante un array de cromosomas binarios.

Representación Cromosomática Directa



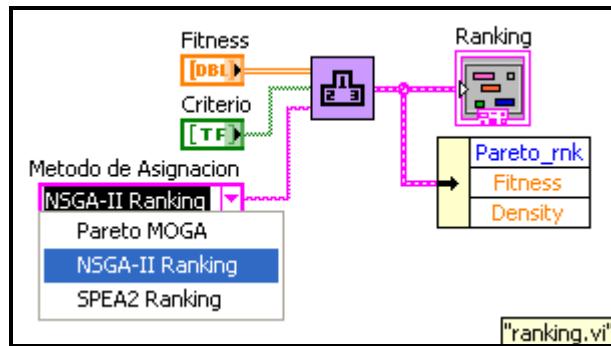
FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Seed_DNA	Cluster (cromosoma)	Cromosoma que se copiará.
ConfigNN	Cluster (configuración)	Valores de configuración para la red neuronal, incluye: total de neuronas, número de neuronas ocultas, número de entradas y la existencia de bias.
#bits	Numérica (I32)	Define la longitud de bits de los parámetros evolucionables.
seed?	Booleana	Si es falsa se creará un cromosoma nuevo, si es verdadera se copiará Seed_DNA.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
DNAAb	Cluster (cromosoma)	Cromosoma obtenido; internamente se representa mediante dos matrices booleanas "Pesos" y "Constantes". <ul style="list-style-type: none"> • Pesos: Representa todas las conexiones sinápticas de la red. • Constantes: Representa todos los parámetros neuronales.
N-DNAAb	Cluster (cromosoma)	En todos los casos resulta de la negación lógica de las matrices booleanas "DNAAb".

Conversión de Binario a Real



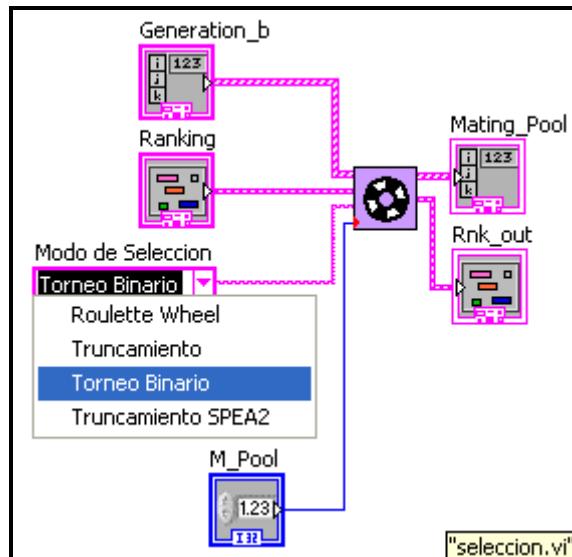
FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Generation_b	Array de clusters	Población representada internamente mediante un array de cromosomas binarios, cada arquitectura neuronal distinta denota un genoma diferente.
#bits	Numérica (I32)	Determina la longitud de bits de cada gen en un cromosoma.
rangos	Cluster de clusters	Dependiendo de la red neuronal que se emplee, define los rangos dentro de los que se linealizarán los distintos grupos de factores; como constantes de tiempo, pesos sinápticos, etc.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Generation_r	Array de clusters	Población representada internamente mediante un array de cromosomas reales, que se obtuvieron a partir de la linealización de los parámetros en genomas binarios originales.

Asignación de Fitness Multiobjetivo



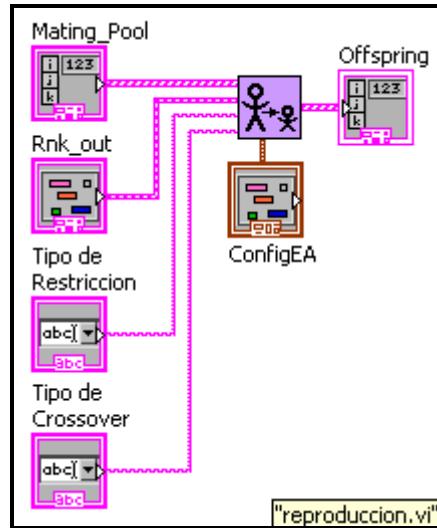
FUNCION		
Se asigna un Ranking a cada solución de la Población, a partir de los resultados obtenidos de la función de evaluación.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Fitness	Matriz (DBL)	Fila: Soluciones. Columna: Parámetros de evaluación.
Metodo de Asignacion	Combo-box	<ul style="list-style-type: none"> NSGA-II Ranking: Ejecuta el método NSGA-II. SPEA2 Ranking: Ejecuta el método SPEA2.
Criterio	Array (booleano)	Define la dirección del ordenamiento para cada parámetro de evaluación, descendente ("F": menor es mejor), ascendente ("T": mayor es mejor).
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Ranking	Cluster	<ul style="list-style-type: none"> Pareto_rnk: Array que denota la categorización de las soluciones según el método de asignación. Fitness: Copia de la matriz de entrada "Fitness". Density: Array, define una métrica de diversidad para cada individuo según el método de asignación (NSGA-II y SPEA2).

Selección



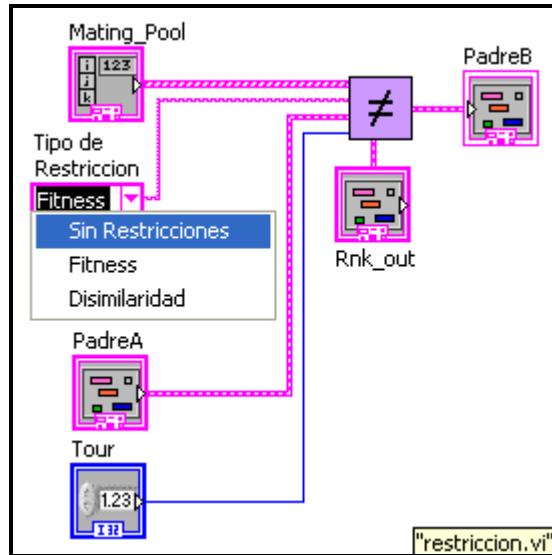
FUNCION		
Selecciona una cantidad de individuos a partir de una Población de entrada.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Generation_b	Array de clusters	Array de cromosomas binarios.
Ranking	Cluster	Contiene los arrays Pareto_rnk, Density y la matriz Fitness.
Modo de Selección	Combo-box	<ul style="list-style-type: none"> • Roulette Wheel. • Truncamiento. • Torneo Binario. • Truncamiento SPEA2.
M-Pool	Numérica (I32)	Define la cantidad de individuos a seleccionar.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Mating_Pool	Array de clusters	Array de cromosomas binarios seleccionados.
Rnk_out	Cluster	Contiene los arrays Pareto_rnk, Density y la matriz Fitness relativos a las soluciones escogidas.

Reproducción



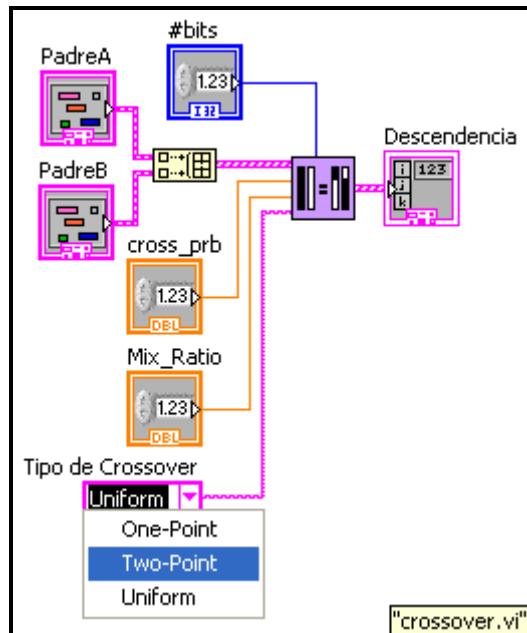
FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Mating_Pool	Array de clusters	Array de cromosomas binarios escogidos para ser progenitores.
Rnk_out	Numérica (DBL)	Contiene los arrays Pareto_rnk, Density y la matriz Fitness relativos a los progenitores.
Tipo de Restriccion	Combo-box	<ul style="list-style-type: none"> • Sin Restricciones. • Fitness. • Disimilaridad.
Tipo de Crossover	Combo-box	<ul style="list-style-type: none"> • One-Point. • Two-Point. • Uniform.
ConfigEA	Cluster (configuración)	Cluster que contiene parámetros de configuración del algoritmo evolutivo, en el subVI "Reproducción" se usan: "Offspring", "Tour", "#bits", "cross_prb", "Mix_Ratio", "mut_prb", "mut_rate".
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Offspring	Array de clusters	Array de genotipos que representan la descendencia total creada.

Restricciones



FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Mating_Pool	Array de clusters	Array de cromosomas binarios escogidos para ser progenitores.
Tipo de Restriccion	Combo-box	<ul style="list-style-type: none"> • Sin Restricciones: No existen. • Fitness: Se buscan soluciones de mejor/igual Ranking que PadreA. • Disimilaridad: Se buscan Padres genotípicamente distintos entre sí.
PadreA	Cluster	Compuesto por el genotipo y el Ranking de un individuo seleccionado aleatoriamente entre el Mating_Pool.
Tour	Numérica (I32)	Define el tamaño del subgrupo de Progenitores, quienes se compararán con PadreA.
Rnk_out	Cluster	Contiene los arrays Pareto_rnk, Density y la matriz Fitness.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
PadreB	Cluster (cromosoma)	Cromosoma seleccionado para combinarse con PadreA y producir descendencia.

Crossover



FUNCION

Ejecuta una operación de Crossover Binario entre 2 Padres, produciendo 2 Hijos, los tipos de Crossover permitidos son de un Punto, de dos Puntos y Uniforme. SubVI en “reproduccion.vi”.

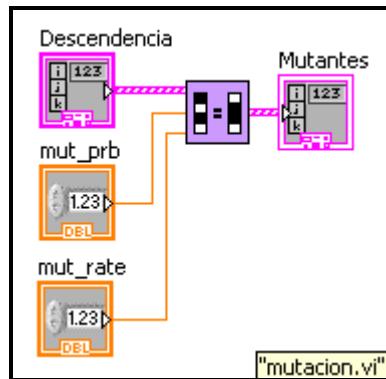
ENTRADAS

NOMBRE	TIPO	DESCRIPCION
Padres	Array de clusters	Array de 2 cromosomas binarios.
#bits	Numérica (I32)	Longitud de bits de los parámetros de la red neuronal.
Cross_prb	Numérica (DBL)	Probabilidad de que la operación de Crossover suceda.
Mix_Ratio	Numérica (DBL)	Probabilidad de intercambio de los genes de los Padres bajo el modelo de Crossover Uniforme.
Tipo de Crossover	Combo-box	<ul style="list-style-type: none"> • One-Point. • Two Point. • Uniform.

SALIDAS

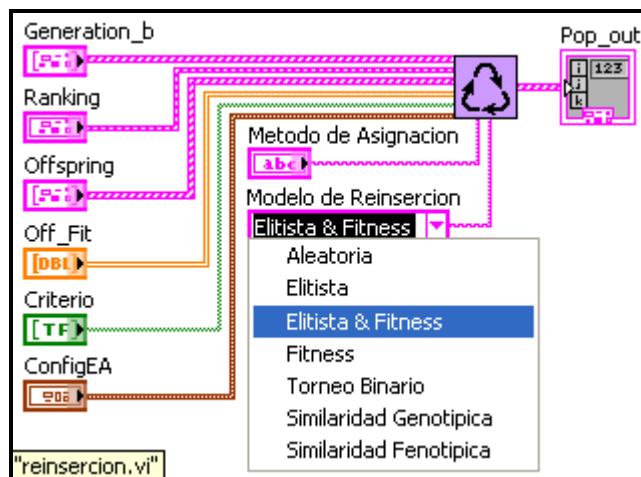
NOMBRE	TIPO	DESCRIPCION
Descendencia	Array de clusters	Array de 2 cromosomas binarios.

Mutación



FUNCION		
Ejecuta una operación de Mutación Binaria Estática basada en conmutación de bits, crea Mutantes a partir de los genotipos incluidos en Descendencia. SubVI en "reproduccion.vi".		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Descendencia	Array de clusters	Array de cromosomas binarios.
mut_prb	Numérica (DBL)	Probabilidad de que la operación de Mutación suceda.
mut_rate	Numérica (DBL)	Probabilidad de conmutación por bit.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Mutantes	Array de clusters	Array de cromosomas binarios.

Reinserción

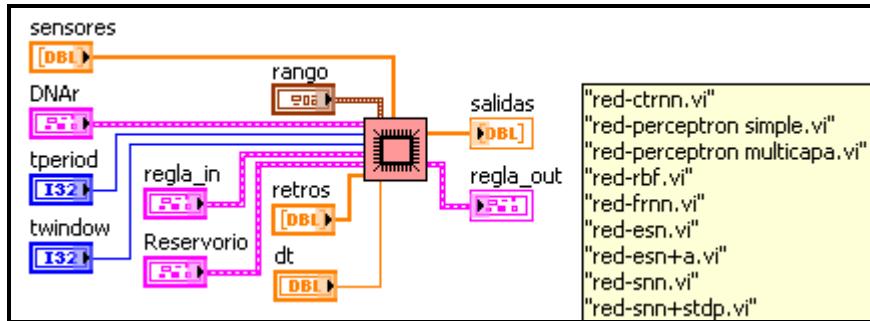


FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Generation_b	Array de clusters	Array de cromosomas binarios.
Ranking	Cluster	Clasificación de Generation_b.
Offspring	Array de clusters	Array de genotipos.
Off_Fit	Matriz (DBL)	Resultado de la simulación.
Criterio	Array (booleano)	Direcciones de mejor Fitness.
ConfigEA	Cluster (configuración)	Contiene los parámetros “Reinsert” “Time_step”, “Trials” y “Epochs”.
Metodo de Asignacion	Combo-box	Define como se realizará el ranking de las soluciones en la Población.
Modelo de Reinsercion	Combo-box	<ul style="list-style-type: none"> • Aleatoria. • Elitista. • Elitista basada en Fitness. • Basada en Fitness. • Basada en Torneo Binario. • Basada en similaridad Genotípica. • Basada en similaridad Fenotípica.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Pop_out	Array de clusters	Nueva Población de cromosomas.

ANEXO G

DESARROLLO ALGORITMICO – ESTRUCTURAS NEURONALES

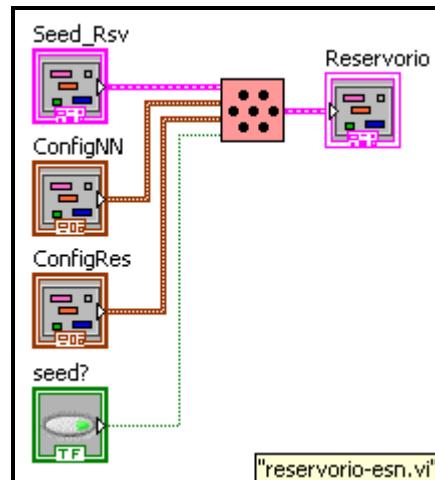
Red Neuronal



FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
sensores	Array (DBL)	Denota las entradas sensoriales.
DNAr	Cluster	Contiene el genotipo decodificado para ser empleado en la simulación.
rango	Cluster (DBL)	En redes RBF incorpora los límites de los pesos sinápticos de la 2da capa.
retros	Array (DBL o Boolean)	Denota la retroalimentación en la red.
dt	Numérica (DBL)	Define el paso de simulación.
tperiod*	Numérica (I32)	En redes SNN define la frecuencia de interface entre la red y la simulación.
twindow*	Numérica (I32)	En redes SNN define el periodo sobre el cual se calcula las salidas motoras.
regla_in*	Cluster	En redes SNN incorpora 3 matrices que definen en cada instante el peso y potenciales históricos por sinapsis.
Reservorio*	Cluster	En redes ESN incorpora 3 matrices que denotan los pesos fijos de la red.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
salidas	Array (DBL o Boolean)	Activación resultante de cada neurona.
regla_out*	Cluster	En redes SNN contiene las matrices actualizadas de pesos y potenciales históricos de las sinapsis.

*Parámetros solo existen para estructuras neuronales específicas.

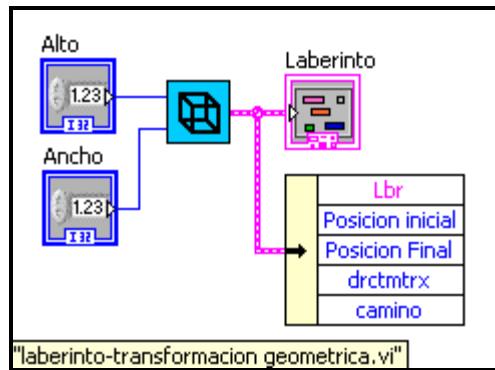
Reservorio



FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Seed_Rsv	Cluster (Reservorio)	Contiene un Reservorio, para el cual sus matrices han sido definidas por el usuario de forma manual.
ConfigNN	Cluster (configuración)	Valores de configuración para la red neuronal, incluye: total de neuronas, número de neuronas ocultas, número de entradas y la existencia de bias.
ConfigRes	Cluster (configuración)	Configuración del Reservorio incluye: radio espectral, factor de conectividad para las matrices "Connectivity", "Inp-Res" e "Inp-Out", constante de peso para "Inp-Res" e "Inp-Out".
seed?	Booleana	Si es falsa se creará un Reservorio nuevo, si es verdadera se copiará Seed_Rsv.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Reservorio	Cluster (Reservorio)	Contiene 3 matrices: "Connectivity", "Inp-Res" e "Inp-Out", definen pesos entre neuronas del Reservorio, entre Entradas y el Reservorio y entre las Entradas y Salidas respectivamente.

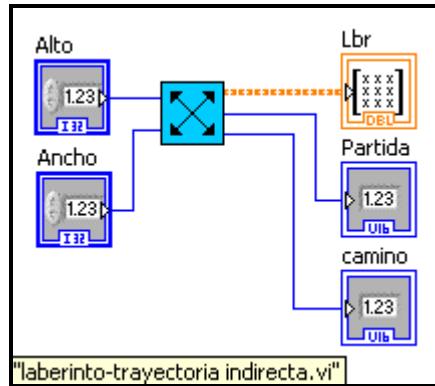
ANEXO H
DESARROLLO ALGORITMICO – SIMULACION

Laberinto – Transformación Geométrica



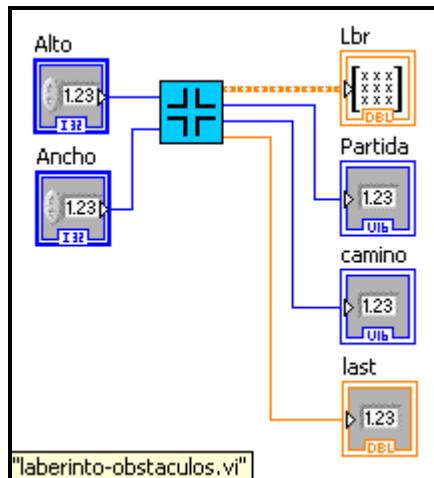
FUNCION		
Transforma los marcadores numéricos de la matriz "Lbr (DBL)" en indicadores de tipo alfanuméricos que representan ambientes físicos específicos. Es un subVI en "simpop_(NN).vi".		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Alto	Numérica (I32)	Determina la dimensión vertical del ambiente que se va a crear, sin considerar el factor de escala.
Ancho	Numérica (I32)	Determina la dimensión horizontal del ambiente que se va a crear, sin considerar el factor de escala.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Laberinto	Cluster (Laberinto)	<ul style="list-style-type: none"> • Lbr (string) es una matriz, la cual define marcadores que describen formas geométricas particulares. • "Posicion inicial" es un vector que señala el lugar donde se ubica la Partida dentro de "Lbr (string)". • "Posicion final" es un vector que señala el lugar donde se ubica la Meta dentro de "Lbr (string)". • "drctmtrx", matriz entera (I16) que indica la longitud de cada camino indirecto. • "Camino" es un parámetro, el cual señala la longitud de la trayectoria directa.

Laberinto – Trayectoria Indirecta



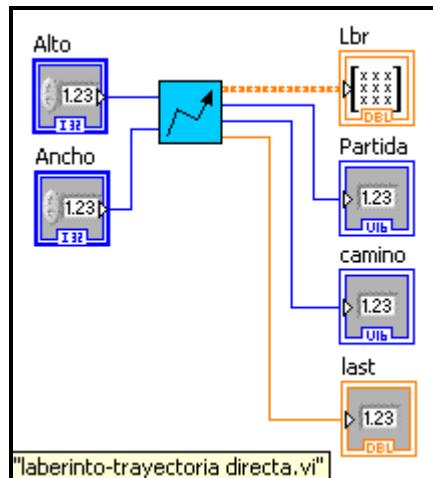
FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Alto	Numérica (I32)	Determina la dimensión vertical del ambiente que se va a crear, sin considerar el factor de escala.
Ancho	Numérica (I32)	Determina la dimensión horizontal del ambiente que se va a crear, sin considerar el factor de escala.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Lbr	Matriz (DBL)	Describe las particularidades de la trayectoria generada, en base a marcadores numéricos específicos.
Partida	Numérica (U16)	Señala dentro de "Lbr" la posición de partida del laberinto, se requiere sólo una variable, el índice de columna; ya que la partida siempre se encontrará en la fila de mayor orden.
Camino	Numérica (U16)	Indica la longitud de la trayectoria.

Laberinto – Obstáculos



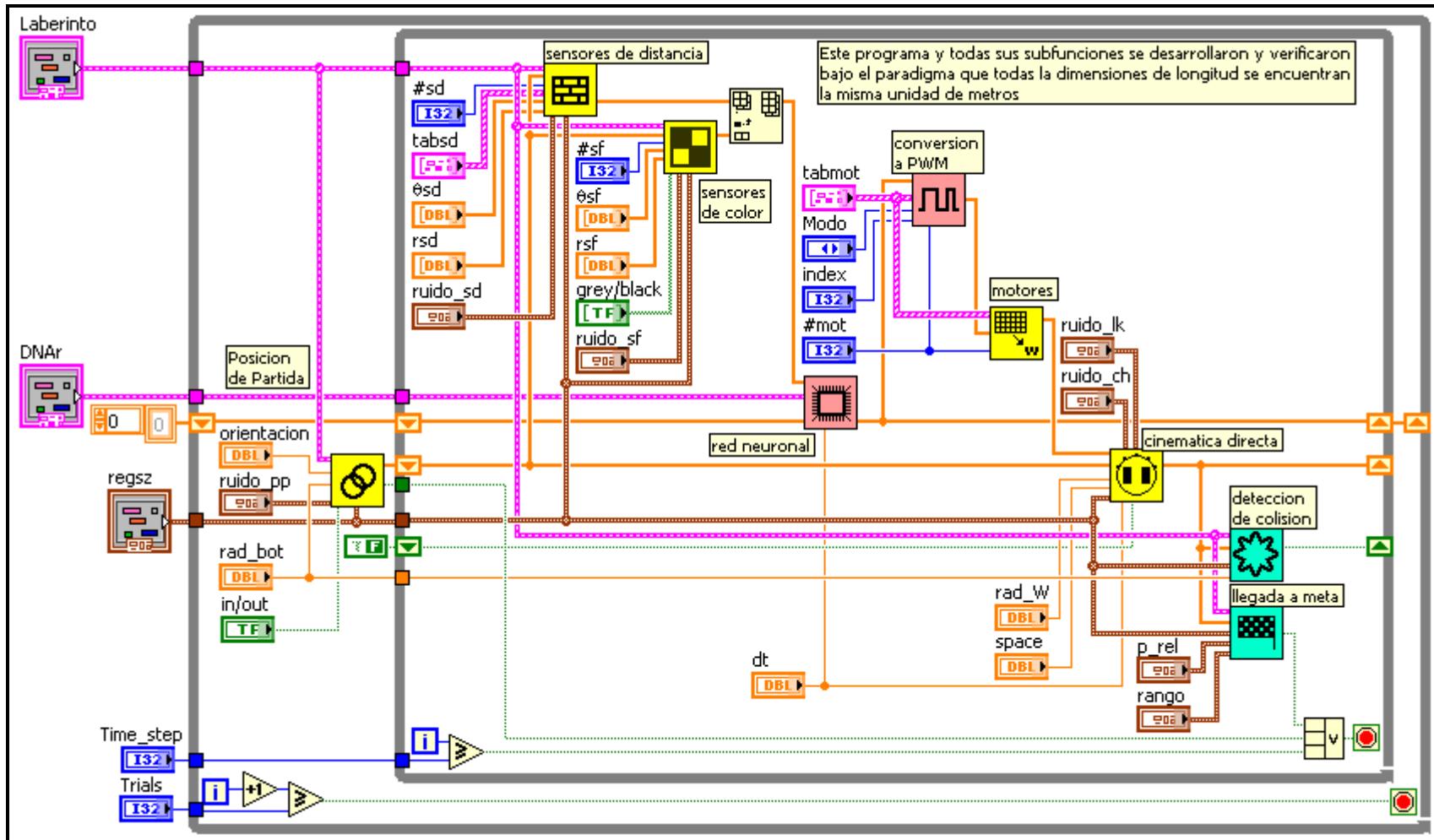
FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Alto	Numérica (I32)	Determina la dimensión vertical del ambiente que se va a crear, sin considerar el factor de escala.
Ancho	Numérica (I32)	Determina la dimensión horizontal del ambiente que se va a crear, sin considerar el factor de escala.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Lbr	Matriz (DBL)	Describe las particularidades de la trayectoria generada, en base a marcadores numéricos específicos.
Partida	Numérica (U16)	Señala dentro de "Lbr" la posición de partida del laberinto, se requiere sólo una variable, el índice de columna; ya que la partida siempre se encontrará en la fila de mayor orden.
Camino	Numérica (U16)	Indica la longitud de la trayectoria.
Last	Numérica (DBL)	Indica el marcador utilizado por el último sub-camino.

Laberinto – Trayectoria Directa

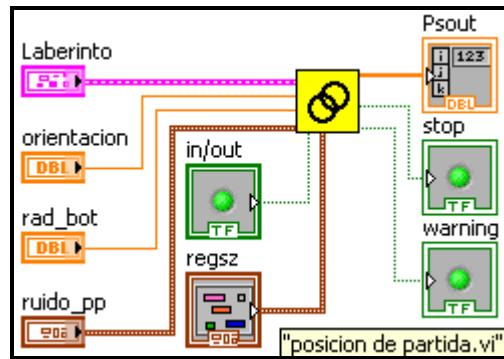


FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Alto	Numérica (I32)	Determina la dimensión vertical del ambiente que se va a crear, sin considerar el factor de escala.
Ancho	Numérica (I32)	Determina la dimensión horizontal del ambiente que se va a crear, sin considerar el factor de escala.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Lbr	Matriz (DBL)	Describe las particularidades de la trayectoria generada, en base a marcadores numéricos específicos.
Partida	Numérica (U16)	Señala dentro de "Lbr" la posición de partida del laberinto, se requiere sólo una variable, el índice de columna; ya que la partida siempre se encontrará en la fila de mayor orden.
Camino	Numérica (U16)	Indica la longitud de la trayectoria.
Last	Numérica (DBL)	Indica el marcador utilizado por el último sub-camino.

Algoritmo de Simulación Básico

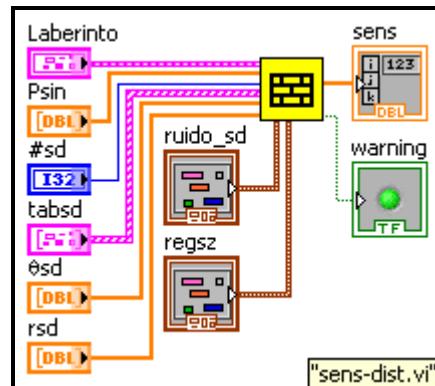


Posición de Partida



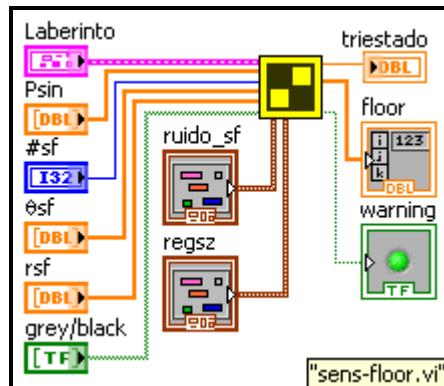
FUNCION		
Agrega aleatoriedad a la “Partida” descrita en “Laberinto”.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Laberinto	Cluster	Define el ambiente en que se llevará a cabo la simulación.
regsz	Cluster (DBL)	Define dimensiones horizontal (thi) y vertical (tvi) de las cuadrículas base que componen el ambiente.
orientacion	Numérica (DBL)	Orientación inicial del móvil (radianes).
rad_bot	Numérica (DBL)	Para el caso particular de un robot con forma circular, define su radio.
ruido_pp	Cluster (DBL)	Define los valores máximo y mínimo que pueden variar las coordenadas y orientación iniciales (x, y, θ) en metros y radianes correspondientemente.
in/out	Booleana	Si es “True” define si la simulación se desea detener en caso la posición del móvil se encuentre fuera de los límites del ambiente o cometa una colisión.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Psout	Array (x, y, θ)	Indica las coordenadas (x,y) del punto medio entre la ruedas del móvil, así como la orientación de una recta perpendicular a dicho eje (θ).
stop	Booleana	Indica si la simulación se detiene (T) o no (F), es siempre “F” si “in/out=F”.
warning	Booleana	Proviene de “colision.vi” .

Sensores de Distancia



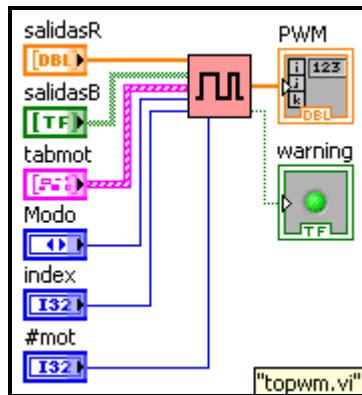
FUNCION		
Determina el estado de activación de cada sensor de distancia.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Laberinto	Cluster	Define el ambiente de simulación.
Psin	Array (x, y, θ)	Define posición y orientación del robot.
#sd	Numérica (I32)	Define el número de sensores usados.
tabsd	Array de Clusters	Define para cada sensor una tabla de valores que indica su comportamiento.
θsd	Array (θ1, ..., θn)	Define la posición angular relativa de cada sensor respecto de la orientación (θ) descrita en "Psin" en radianes.
rsd	Array (r1, ..., rn)	Define la distancia radial relativa de cada sensor respecto del punto (x, y) perteneciente al array "Psin".
ruido_sd	Cluster (Δang, Δdist)	Define máximos y mínimos para el ruido que es agregado a la posición angular (grados.) y distancia virtual.
regsz	Cluster (DBL)	Define dimensiones horizontal (thi) y vertical (tvi) de la cuadrícula base que compone el ambiente.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
sens	Array (sd1, ..., sdn)	Indica la activación de cada sensor.
warning	Booleana	Indica si el tamaño de los arrays de entrada corresponde al número de sensores de distancia.

Sensores de Color



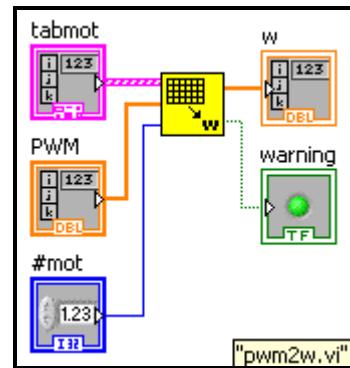
FUNCION		
Determina el estado de activación de cada sensor de color.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Laberinto	Cluster	Define el ambiente de simulación.
Psin	Array (x, y, θ)	Define posición y orientación del robot.
#sf	Numérica (I32)	Define el número de sensores usados.
θsf	Array (θ1, ..., θn)	Define la posición angular rel. (rad.) de cada sensor respecto de "θ" en "Psin".
rsf	Array (r1, ..., rn)	Define la distancia radial relativa de cada sensor respecto del punto (x, y) perteneciente al array "Psin".
grey/black	Array (booleano)	Define el modo de operación de cada sensor, "T" indica que diferencia entre gris/negro o blanco, y "F" señala que el sensor detecta blanco/gris o negro.
regsz	Cluster (DBL)	Define dimensiones horizontal (thi) y vertical (tvi) de la cuadrícula base.
ruido_sf	Cluster (DBL)	Define constantes para la generación de ruido (metros y probabilidades).
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
triestado	Numérica (DBL)	Indica si el robot se encuentra sobre piso blanco (0), gris (0.5) o negro (1).
floor	Array (sd1, ..., sdn)	Indica la activación de cada sensor.
warning	Booleana	Indica si el tamaño de los arrays de entrada corresponde a #sf.

PWM



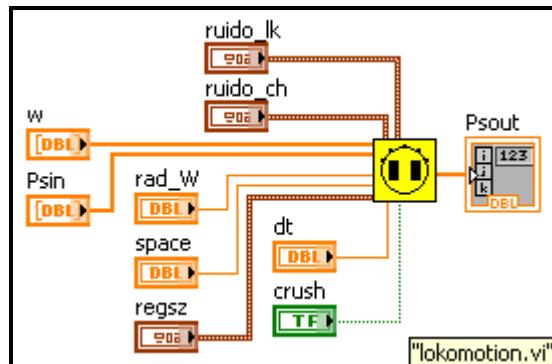
FUNCION		
Transforma la salida del controlador en una señal PWM.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
salidasR	Array (DBL)	Salidas de todas las neuronas de una red neuronal, excepto redes SNN.
salidasB	Matriz (booleana)	Registros de los "n" últimos estados de todas las entradas y neuronas que pertenecen a una estructura SNN.
tabmot	Array de Clusters	Define para cada motor una tabla de valores que indica su comportamiento.
Modo	Seleccionador (Enum)	Presenta tres opciones ("[0,1]", "[-1,1]", "pulses") que definen con que entrada se trabajará ("salidasR" o "salidasB"), y el rango en el que la señal de salida del controlador se encuentra.
index	Numérica (I32)	Para "salidasR" define la 1ra posición del vector que se usará para generar una señal PWM y para "salidasB" la 1ra fila que será considerada.
#mot	Numérica (I32)	Define el número de motores usados.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
PWM	Array (s1, ..., sn)	Señal PWM (μ s.) para cada motor.
warning	Booleana	Indica si el tamaño de las entradas, el index o el número de motores no es el adecuado.

Motores



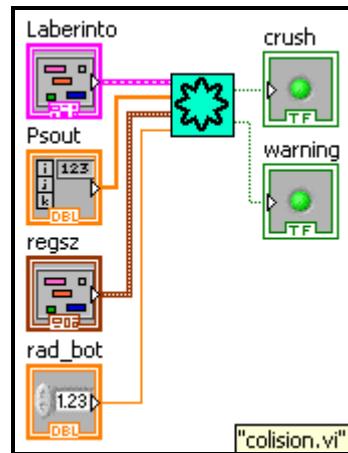
FUNCION		
Obtiene la velocidad angular (w) que exhibirá cada motor de acuerdo a su tabla de valores correspondiente.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
tabmot	Array de Clusters	Define para cada motor una tabla de valores que indica su comportamiento.
PWM	Array (s1, ..., sn)	Define el ancho de pulso que controla la velocidad de cada motor (ms.).
#mot	Numérica (I32)	Define el número de motores usados.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
w	Array (w1, ..., wn)	Indica la velocidad angular en rad/s que manifiesta cada motor.
warning	Booleana	Indica si algún valor de ancho pulso se encuentra fuera de los límites de la escala en la tabla respectiva.

Cinemática Directa



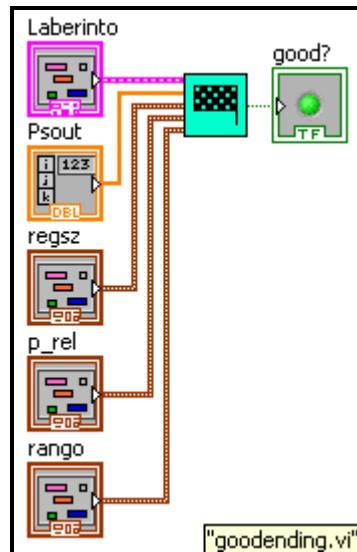
FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
w	Array (w1, ..., wn)	Define la velocidad angular en rad/s que manifiesta cada motor.
Psin	Array (x, y, θ)	Define la posición y la orientación previa del robot.
rad_W	Numérica (DBL)	Define el radio de las ruedas.
space	Numérica (DBL)	Define la distancia entre los centros de ambas ruedas.
regsz	Cluster (DBL)	Define dimensiones horizontal (thi) y vertical (tvi) de la cuadricula base.
dt	Numérica (DBL)	Define el paso de simulación (seg.).
crush	Booleana	Define si hubo una colisión (T) o no (F) en el ciclo de simulación anterior.
ruido_lk	Cluster (DBL)	Define los valores máximo y mínimo que pueden variar las coordenadas y orientación ideal (x, y, θ), todos estos valores definidos como porcentajes.
ruido_ch	Cluster (DBL)	Define los valores máximo y mínimo que pueden variar las coordenadas y orientación del robot luego de una colisión (metros y radianes).
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Psout	Array (x, y, θ)	Indica la posición y la orientación actual del robot.

Detección de Colisión



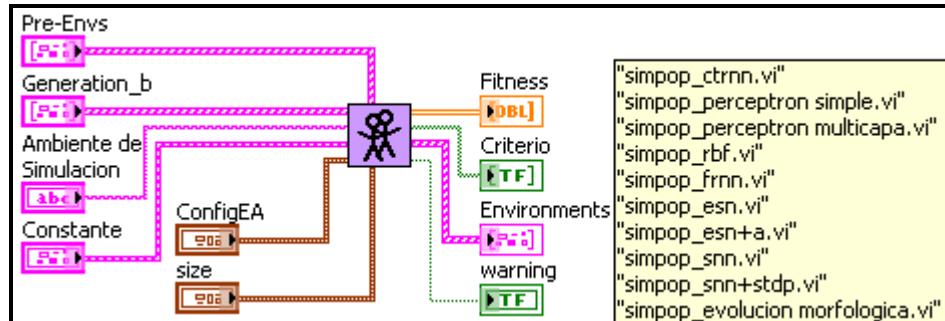
FUNCION		
Determina si existe una colisión del robot contra los límites o paredes del ambiente de simulación o laberinto.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Laberinto	Cluster	Define el ambiente de simulación.
Psout	Array (x, y, θ)	Define la posición y la orientación actual del robot.
regsz	Cluster (DBL)	Define dimensiones horizontal (thi) y vertical (tvi) de la cuadrícula base.
rad_bot	Numérica (DBL)	Define el radio del perímetro exterior del robot, obviamente se considera en esta función un móvil cuyo contorno presenta una forma circular.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
crush	Booleana	Indica si ha ocurrido (T) o no (F) una colisión entre el robot y su alrededor.

Terminación por Meta



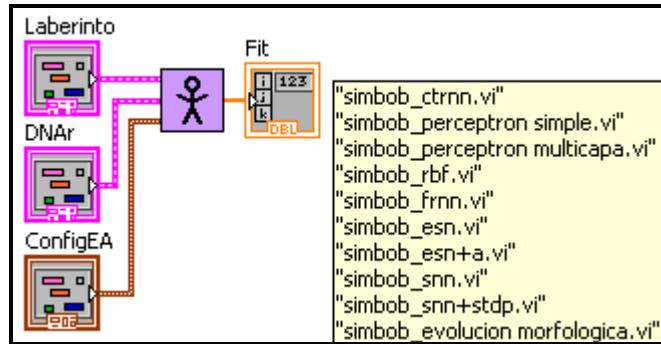
FUNCION		
Determina si el robot llegó hasta la posición de “Meta”.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Laberinto	Cluster	Define el ambiente de simulación.
Psout	Array (x, y, θ)	Define la posición y la orientación actual del robot.
regsz	Cluster (DBL)	Define dimensiones horizontal (thi) y vertical (tvi) de la cuadrícula base.
p_rel	Cluster (DBL)	Coordenadas relativas (x, y) respecto del punto “Psout”.
rango	Cluster (DBL)	Define las dimensiones (x+, x-, y+, y-) de la región que debe alcanzar el robot para concluir la simulación.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
good?	Booleana	Indica si el robot logró llegar hasta la posición de “Meta” (T) o si todavía se encuentra en camino (F).

Simulación de una Población en un EA



FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Generation_b	Array de clusters	Población representada internamente mediante un array de cromosomas.
Ambiente de Simulacion	Combo-box	Ambientes que se crean en cada paso generacional, se han implementado 4 opciones: "Aleatorio" – dimensiones según "size"; "Simple-T" – una única bifurcación T en matriz 4x7; "Mini-T" – única bifurcación T en matriz 2x3; "Constante" – predeterminado.
Constante	Cluster	Ambiente predeterminado.
ConfigEA	Cluster (configuración)	Incluye parámetros como: "Epochs", "#bits", "Time_step", "Trials".
size	Cluster (DBL)	Dimensiones del laberinto "Aleatorio".
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Fitness	Matriz (DBL)	Resultado de la simulación. Fila: Soluciones. Columna: Parámetros de evaluación.
Criterio	Array (booleano)	Indica la dirección de mejor Fitness (ascendente o descendente) para cada parámetro de evaluación.
Environments	Array de Clusters	Ambientes de Simulación por época.
warning	Booleana	Si el # de ambientes es menor que el # de épocas (Epochs) señala "T".

Simulación de un Individuo en un EA

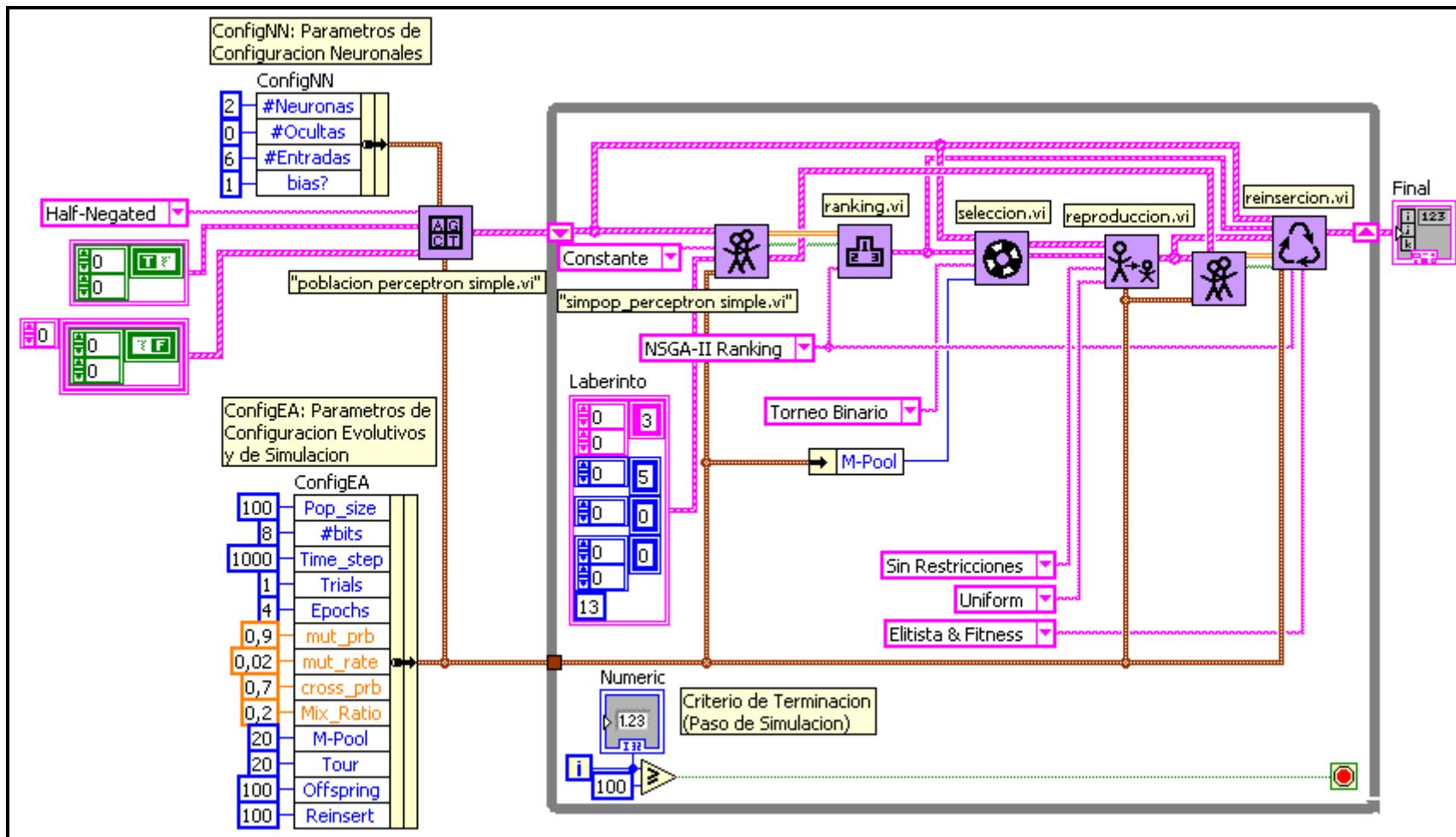


FUNCION		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Laberinto	Cluster	Define el ambiente de simulación.
DNAr	Cluster (cromosoma)	Genoma de un individuo; internamente se representa mediante una o dos matrices reales.
ConfigEA	Cluster (configuración)	Incluye los parámetros: "Time_step", "Trials".
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Fit	Array (DBL)	Indica el desempeño del individuo, cada elemento del array representa un parámetro de evaluación distinto.

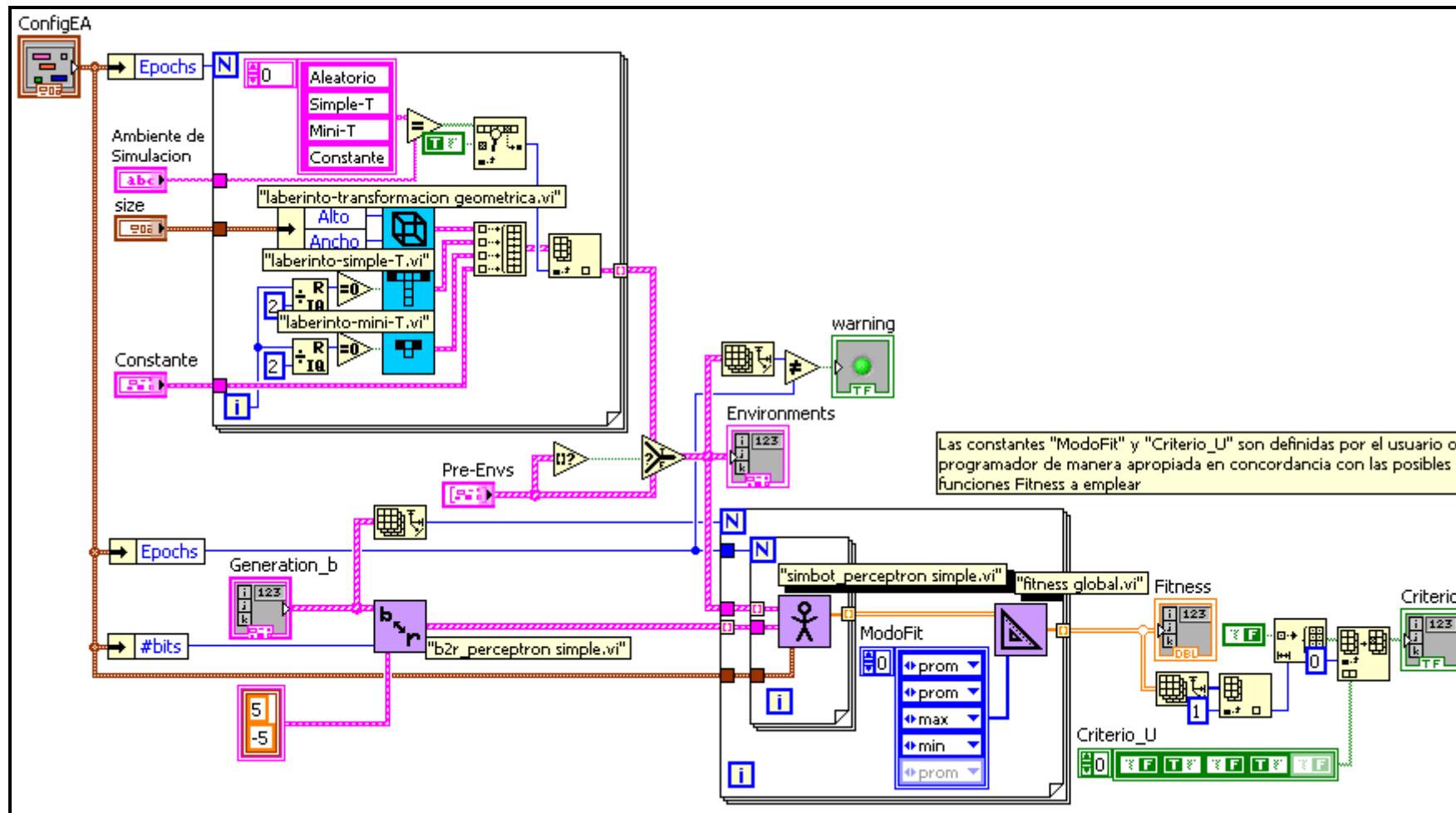
ANEXO I

DESARROLLO ALGORITMICO – EXP#1: FUNCIONES FITNESS

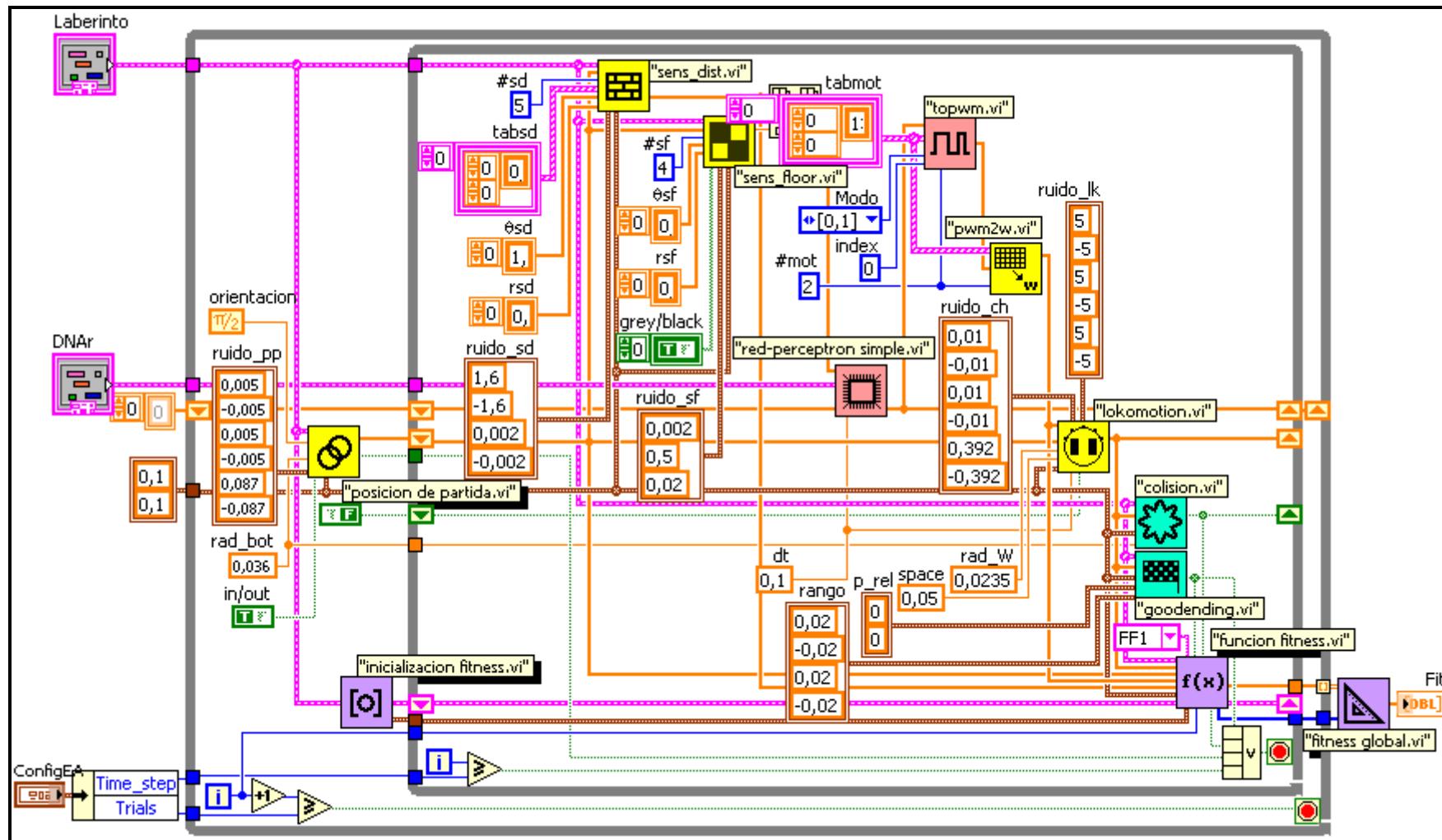
Experiencia 1: Algoritmo Evolutivo (Diagrama de Bloques)



Experiencia 1: Simulación de la Población (Diagrama de Bloques)



Experiencia 1: Simulación de un Individuo (Diagrama de Bloques)



Experiencia 1: Constantes

Laberinto										tabmot									
0	3E3	2H2	1N1	2H2	303	3E3	2H2	2NE	0	0	1300	-53,2	0	1300	-49,5	0	1300	-49,5	0
0	0	0	2V2	0	0	0	0	2V2	0	0	1320	-52,6	0	1320	-49,1	0	1320	-49,1	0
	3E3	2H2	000	2H2	2H2	2H2	2H2	000	2H2	2SE	1340	-51,7		1340	-48,3		1340	-48,3	
	0	0	2V2	0	0	0	0	2V2	0	0	1360	-50,1		1360	-46,5		1360	-46,5	
	3E3	2H2	000	2H2	303	0	0	250	2H2	303	1380	-42,5		1380	-40,8		1380	-40,8	
	0	0	3N3	0	0	0	0	0	0	0	1400	-35,8		1400	-34,8		1400	-34,8	
	0	0	5	2							1420	-28,1		1420	-26,7		1420	-26,7	
Posicion inicial											1440	-20,7		1440	-19,9		1440	-19,9	
	0	0	5	2							1460	-12,3		1460	-12,8		1460	-12,8	
Posicion Final											1480	-6,2		1480	-6,4		1480	-6,4	
	0	0	5	2							1500	0		1500	0		1500	0	
	0	0	0	0	0	-1	-1	-2	0	0	1520	6,8		1520	5,8		1520	5,8	
	0	0	6	0	0	0	0	-1	0	0	1540	12,8		1540	11,5		1540	11,5	
	0	2	-2	-1	-1	-1	-1	-2	4	0	1560	21,2		1560	19,2		1560	19,2	
	0	0	-1	0	0	0	0	4	0	0	1580	29,3		1580	26,4		1580	26,4	
	0	2	-1	2	0	0	0	0	0	0	1600	37,7		1600	33,7		1600	33,7	
	0	0	-1	0	0	0	0	0	0	0	1620	44		1620	39,8		1620	39,8	
	13										1640	50,4		1640	45,7		1640	45,7	
											1660	52,1		1660	47,9		1660	47,9	
											1680	53,3		1680	48,3		1680	48,3	
											1700	53,7		1700	48,8		1700	48,8	

Experiencia 1: Constantes (continuación)

	θsd	rsd	
	[0] 1,308996938996 0,6544984694979 0 -0,6544984694979 -1,308996938996	[0] 0,031 0,031 0,031 0,031 0,031	
	θsf	rsf	grey/black
	[0] 0,7853981633974 2,356194490192 3,926990816987 5,497787143782	[0] 0,016 0,016 0,016 0,016	[0] T F * F T F * F
tabsd	sensor1		
	[0] 0,07 -16 -14 -12 -10 -8 -6 -4 -2 0 2 4 6 8 10 12 14 16 [0] 0,005 0,2 0,36 0,59 0,8 0,87 0,93 0,96 0,97 1 0,98 0,96 0,92 0,88 0,81 0,6 0,35 0,22 0,01 0,15 0,24 0,44 0,57 0,63 0,66 0,7 0,7 0,72 0,71 0,69 0,66 0,62 0,57 0,42 0,26 0,16 0,015 0,08 0,15 0,23 0,33 0,36 0,39 0,41 0,41 0,42 0,41 0,4 0,38 0,36 0,34 0,25 0,14 0,11 0,02 0,07 0,11 0,16 0,22 0,24 0,25 0,26 0,27 0,28 0,28 0,26 0,25 0,25 0,23 0,17 0,09 0,07 0,025 0,07 0,08 0,13 0,16 0,17 0,18 0,19 0,2 0,2 0,19 0,18 0,18 0,17 0,17 0,13 0,07 0,07 0,03 0,07 0,07 0,09 0,12 0,12 0,12 0,13 0,14 0,15 0,13 0,13 0,13 0,12 0,12 0,1 0,07 0,07 0,035 0,07 0,07 0,07 0,09 0,1 0,1 0,1 0,1 0,11 0,11 0,11 0,1 0,09 0,08 0,07 0,07 0,07 0,07 0,04 0,07 0,07 0,07 0,07 0,07 0,08 0,08 0,09 0,09 0,09 0,09 0,08 0,08 0,08 0,07 0,07 0,07 0,07 0,07 0,045 0,07 0,07 0,07 0,07 0,07 0,07 0,07 0,07 0,08 0,08 0,08 0,07 0,07 0,07 0,07 0,07 0,07 0,07 0,07		

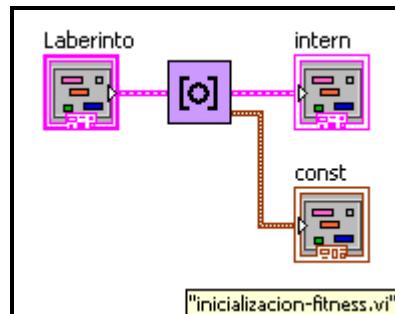
Experiencia 1: Constantes (continuación)

tabsd																		
sensor2																		
0	0,07	-16	-14	-12	-10	-8	-6	-4	-2	0	2	4	6	8	10	12	14	16
0	0,005	0,200	0,352	0,572	0,836	0,847	0,913	0,965	0,984	1	0,975	0,944	0,882	0,856	0,777	0,611	0,353	0,230
0,01	0,153	0,232	0,437	0,562	0,616	0,640	0,708	0,720	0,745	0,711	0,688	0,675	0,625	0,550	0,407	0,268	0,155	
0,015	0,083	0,144	0,233	0,342	0,350	0,380	0,391	0,425	0,425	0,417	0,391	0,382	0,345	0,345	0,242	0,143	0,106	
0,02	0,070	0,113	0,160	0,210	0,235	0,244	0,253	0,263	0,267	0,266	0,255	0,250	0,246	0,237	0,175	0,090	0,07	
0,025	0,07	0,077	0,128	0,155	0,166	0,173	0,184	0,190	0,193	0,184	0,182	0,179	0,173	0,163	0,127	0,070	0,07	
0,03	0,07	0,07	0,089	0,115	0,123	0,123	0,134	0,139	0,152	0,131	0,130	0,129	0,120	0,114	0,095	0,07	0,07	
0,035	0,07	0,07	0,072	0,088	0,098	0,100	0,100	0,100	0,107	0,107	0,106	0,099	0,090	0,077	0,07	0,07	0,07	
0,04	0,07	0,07	0,07	0,070	0,072	0,076	0,082	0,089	0,094	0,092	0,089	0,081	0,081	0,072	0,07	0,07	0,07	
0,045	0,07	0,07	0,07	0,07	0,07	0,070	0,070	0,070	0,078	0,077	0,071	0,070	0,07	0,07	0,07	0,07	0,07	
sensor3																		
0	0,07	-16	-14	-12	-10	-8	-6	-4	-2	0	2	4	6	8	10	12	14	16
0	0,005	0,192	0,342	0,592	0,825	0,839	0,944	0,954	0,982	1	0,961	0,945	0,944	0,847	0,843	0,617	0,335	0,227
0,01	0,143	0,245	0,423	0,556	0,634	0,668	0,685	0,705	0,750	0,742	0,665	0,636	0,590	0,560	0,418	0,265	0,163	
0,015	0,080	0,143	0,238	0,323	0,365	0,397	0,410	0,418	0,439	0,403	0,398	0,385	0,367	0,325	0,244	0,145	0,108	
0,02	0,072	0,113	0,158	0,220	0,245	0,259	0,262	0,265	0,280	0,268	0,264	0,258	0,257	0,225	0,168	0,088	0,07	
0,025	0,07	0,082	0,133	0,164	0,172	0,181	0,189	0,191	0,193	0,192	0,186	0,185	0,174	0,165	0,132	0,072	0,07	
0,03	0,07	0,07	0,086	0,114	0,125	0,125	0,136	0,145	0,148	0,132	0,131	0,128	0,122	0,116	0,099	0,07	0,07	
0,035	0,07	0,07	0,07	0,087	0,096	0,100	0,101	0,104	0,105	0,105	0,104	0,097	0,091	0,081	0,071	0,07	0,07	
0,04	0,07	0,07	0,07	0,07	0,073	0,081	0,083	0,085	0,085	0,085	0,083	0,080	0,072	0,07	0,07	0,07	0,07	
0,045	0,07	0,07	0,07	0,07	0,07	0,070	0,070	0,072	0,077	0,076	0,072	0,071	0,07	0,07	0,07	0,07	0,07	

Experiencia 1: Constantes (continuación)

tabsd																		
sensor4																		
0	0,07	-16	-14	-12	-10	-8	-6	-4	-2	0	2	4	6	8	10	12	14	16
0	0,005	0,201	0,365	0,590	0,839	0,905	0,939	0,945	0,978	1	0,989	0,944	0,938	0,838	0,784	0,575	0,358	0,214
0,01	0,150	0,244	0,460	0,571	0,625	0,651	0,666	0,705	0,720	0,716	0,681	0,631	0,612	0,550	0,421	0,266	0,167	
0,015	0,078	0,143	0,237	0,314	0,354	0,382	0,405	0,410	0,412	0,405	0,394	0,380	0,352	0,341	0,262	0,146	0,104	
0,02	0,07	0,112	0,152	0,228	0,234	0,257	0,262	0,270	0,285	0,281	0,257	0,254	0,250	0,235	0,162	0,087	0,07	
0,025	0,07	0,083	0,124	0,155	0,162	0,177	0,183	0,190	0,190	0,183	0,182	0,173	0,172	0,170	0,130	0,07	0,07	
0,03	0,07	0,070	0,087	0,114	0,120	0,121	0,130	0,138	0,142	0,133	0,131	0,123	0,122	0,118	0,101	0,07	0,07	
0,035	0,07	0,07	0,073	0,094	0,096	0,101	0,101	0,102	0,114	0,111	0,105	0,104	0,087	0,081	0,07	0,07	0,07	
0,04	0,07	0,07	0,07	0,071	0,072	0,077	0,080	0,086	0,087	0,087	0,086	0,083	0,080	0,07	0,07	0,07	0,07	
0,045	0,07	0,07	0,07	0,07	0,07	0,070	0,070	0,070	0,076	0,076	0,071	0,071	0,07	0,07	0,07	0,07	0,07	
sensor5																		
0	0,07	-16	-14	-12	-10	-8	-6	-4	-2	0	2	4	6	8	10	12	14	16
0	0,005	0,197	0,345	0,565	0,788	0,839	0,927	0,933	0,984	1	0,973	0,963	0,960	0,838	0,811	0,606	0,356	0,226
0,01	0,153	0,240	0,447	0,555	0,620	0,675	0,676	0,686	0,686	0,686	0,675	0,646	0,602	0,569	0,422	0,251	0,152	
0,015	0,080	0,153	0,231	0,316	0,375	0,386	0,397	0,412	0,438	0,410	0,402	0,387	0,349	0,326	0,257	0,133	0,110	
0,02	0,072	0,107	0,156	0,210	0,235	0,256	0,268	0,268	0,282	0,281	0,263	0,257	0,238	0,237	0,174	0,090	0,07	
0,025	0,07	0,078	0,127	0,167	0,168	0,182	0,194	0,195	0,198	0,187	0,184	0,172	0,170	0,169	0,123	0,073	0,07	
0,03	0,07	0,07	0,091	0,115	0,121	0,122	0,130	0,143	0,149	0,135	0,130	0,124	0,121	0,116	0,096	0,07	0,07	
0,035	0,07	0,07	0,07	0,091	0,098	0,100	0,101	0,102	0,106	0,106	0,104	0,099	0,085	0,083	0,071	0,07	0,07	
0,04	0,07	0,07	0,07	0,070	0,070	0,079	0,082	0,085	0,086	0,086	0,085	0,081	0,080	0,07	0,07	0,07	0,07	
0,045	0,07	0,07	0,07	0,07	0,07	0,070	0,072	0,073	0,076	0,076	0,073	0,071	0,07	0,07	0,07	0,07	0,07	

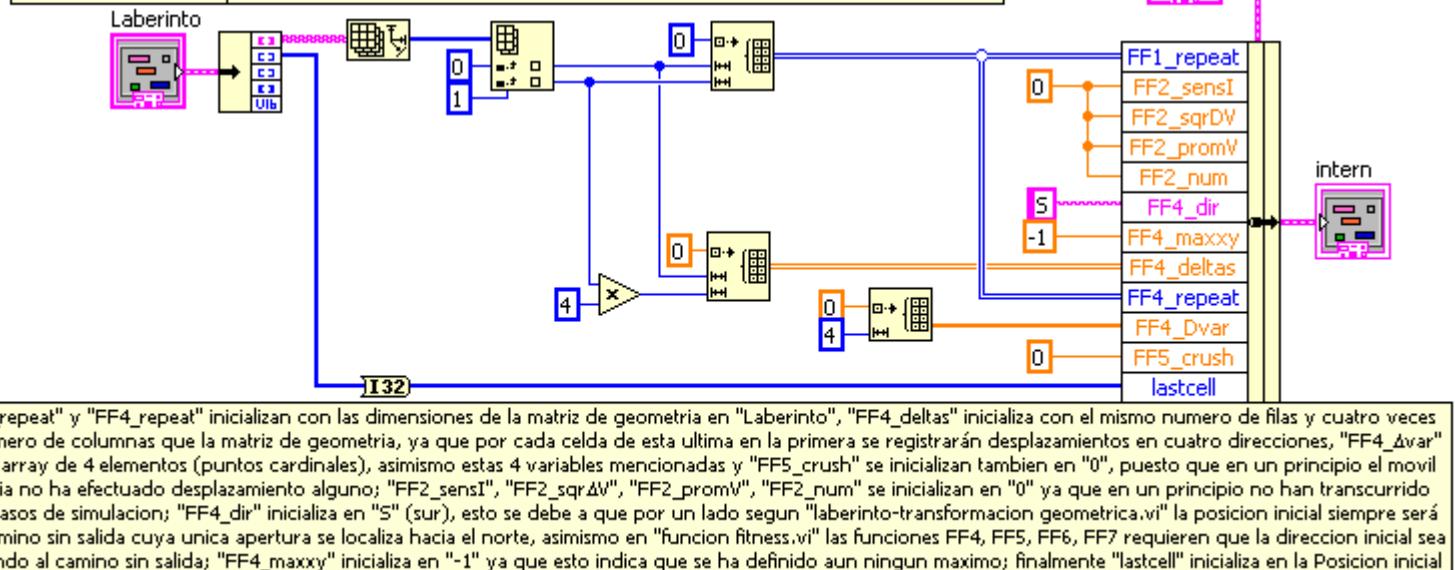
Experiencia 1: Inicialización de la Función Fitness



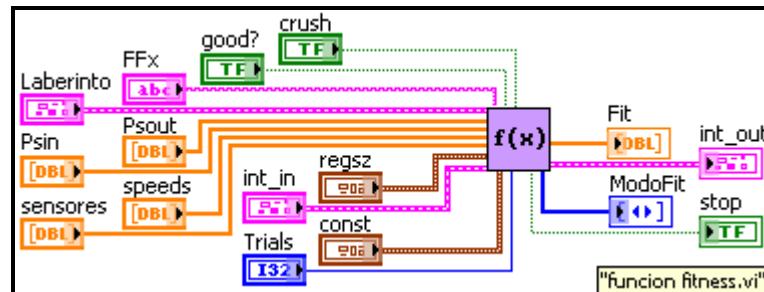
FUNCION		
Inicialización de variables y constantes que serán empleadas para calcular las diversas funciones de desempeño en "función fitness.vi".		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Laberinto	Cluster	Define el ambiente, dentro del cual se llevará a cabo la simulación.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
intern	Cluster	<p>Inicialización de variables:</p> <ul style="list-style-type: none"> • FF1_repeat (matriz I32). • FF2_sensI (DBL). • FF2_sqrΔV (DBL). • FF2_promV (DBL). • FF2_num (DBL). • FF4_dir (string). • FF4_maxxy (DBL). • FF4_deltas (matriz DBL). • FF4_repeat (matriz I32). • FF4_Δvar (array DBL). • FF5_crush (DBL). • lastcell (array I32).
const	Cluster	<p>Inicialización de constantes:</p> <ul style="list-style-type: none"> • FF1_±Δx (DBL). • FF1_±Δy (DBL). • FF2_maxw (DbL).

Experiencia 1: Inicialización de la Función Fitness (Diagrama de Bloques)

Esta función inicializa constantes y variables que serán utilizadas por la función "funcion fitness.vi", para ello es importante que "Laberinto" muestre la estructura generada en el VI "laberinto-transformacion geometrica.vi", así "FF1_±Δx" y "FF1_±Δy" dividen las cuadriculas reales inferidas en "Laberinto" en una región rectangular interna y un anillo rectangular, "FF2_maxw" representa la máxima velocidad angular (rev/min) para cualquier sentido de rotación de cualquier de los motores usados, "FF1_repeat" registra el número de veces en las que el móvil ingresó a cada una de las cuadriculas del ambiente, "FF2_sensI" registra el promedio de la activación del sensor de distancia más cercano a una pared a través de la simulación, "FF2_sqrdV" registra el promedio de la raíz cuadrada de la diferencia algebraica de velocidades angulares entre dos ruedas para móviles con tracción diferencial a través de la simulación, "FF2_promV" registra el promedio de las velocidades de todas las ruedas del móvil a través de la simulación, "FF2_num" indica el número de pasos de simulación que han transcurrido hasta el instante actual, "FF4_dir" referencia la dirección cardinal (norte, sur, este, oeste) en que se desplaza el robot, "FF4_maxxy" indica en situaciones especiales el desplazamiento máximo recorrido por el robot dentro de una cuadricula para una dirección particular, "FF4_deltas" registra el desplazamiento máximo que realizó el robot en cada cuadricula y en cada dirección (norte, sur, este, oeste), "FF4_repeat" registra las veces en que el robot logró el desplazamiento máximo en alguna dirección para cada cuadricula, "FF4_Dvar" referencia los desplazamientos máximos en todas las direcciones mientras el móvil se encuentre en una bifurcación o cruce, "FF5_crush" cuenta el número de colisiones incurridas por el robot, "lastcell" indica la cuadricula en donde se encontró el robot en el paso de simulación anterior.

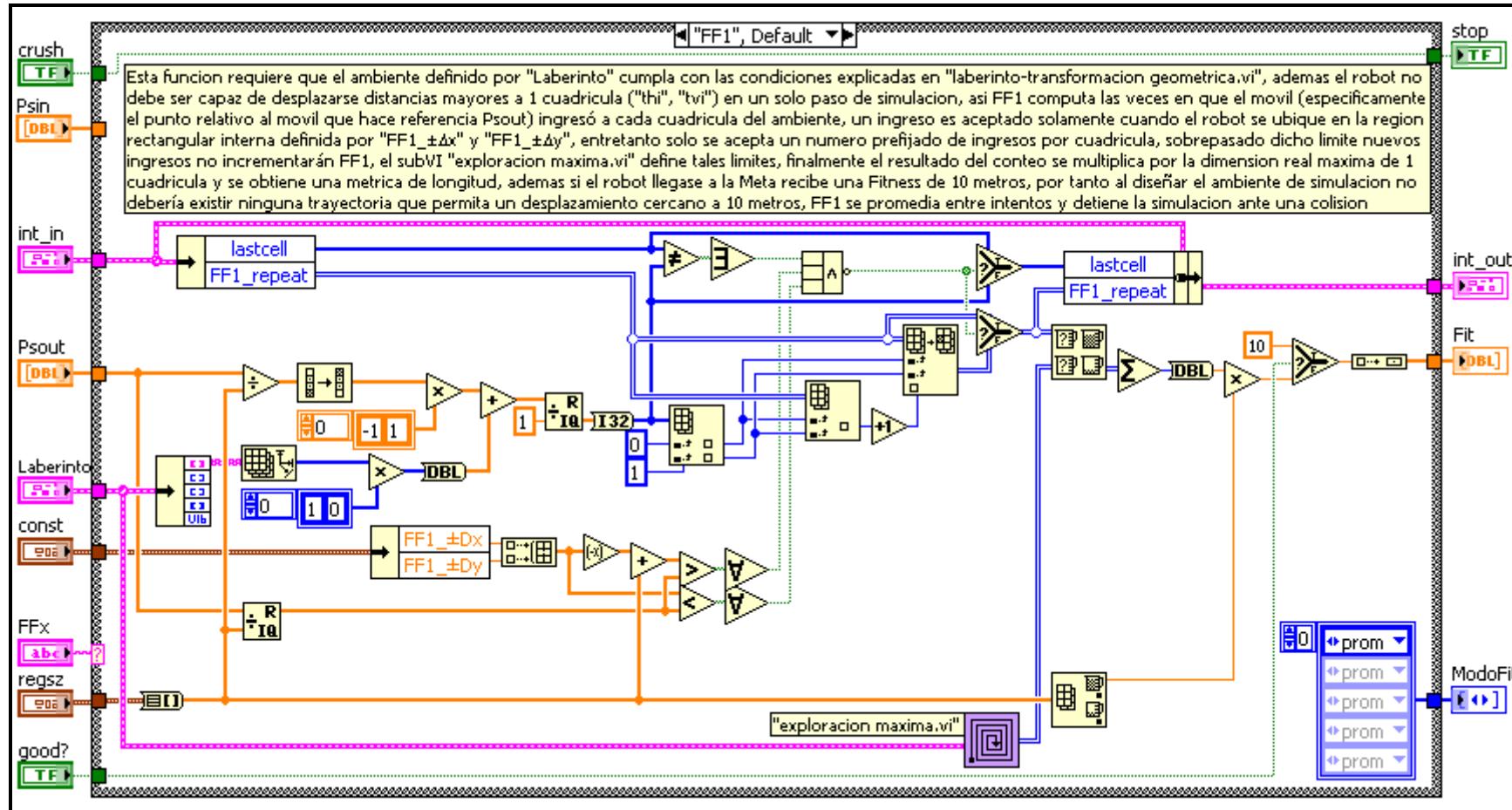


Experiencia 1: Función Fitness

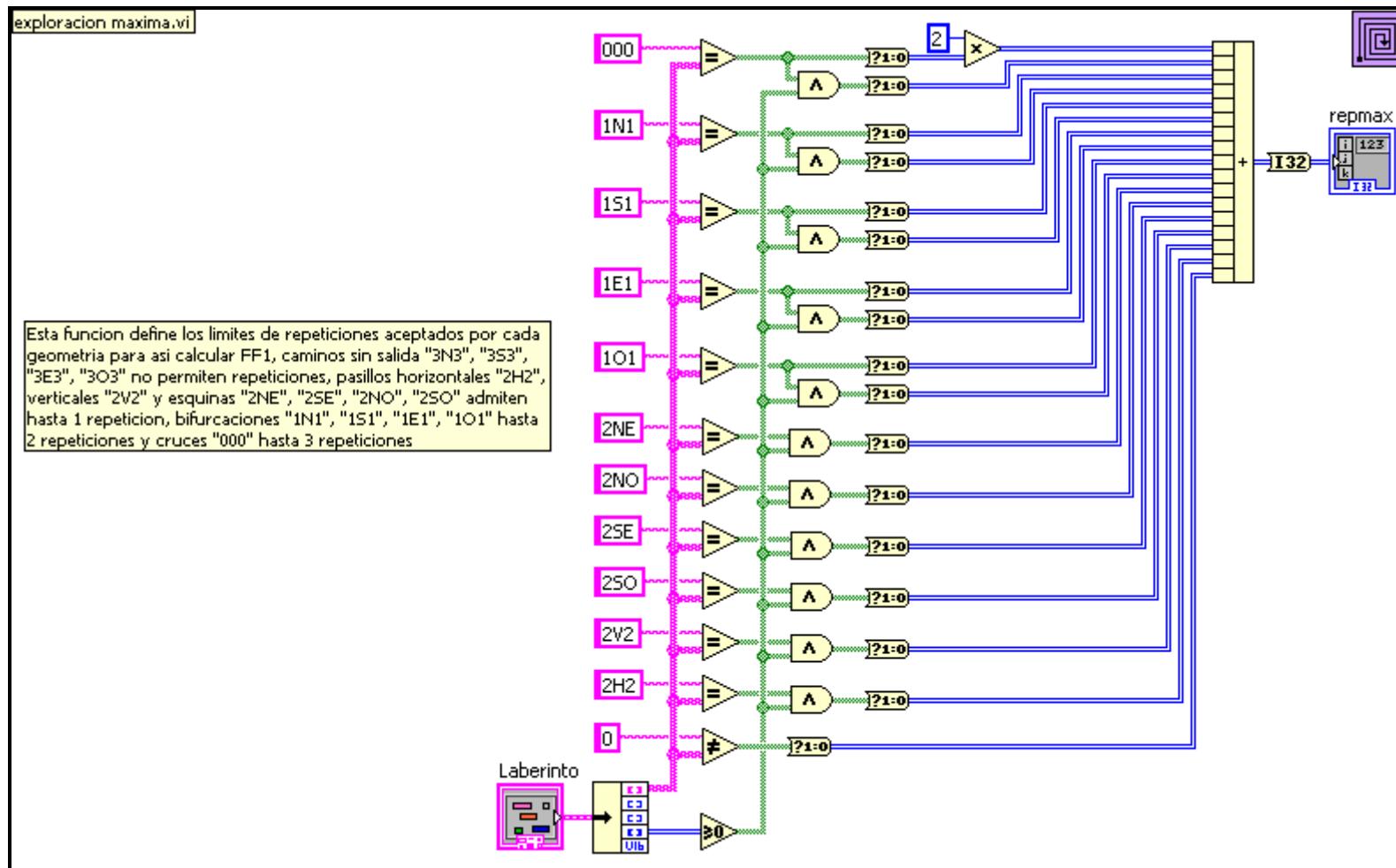


FUNCION		
Calcula la Fitness resultante de cada individuo después de un intento o época.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
FFx	Combo-box	Especifica una función Fitness.
Laberinto	Cluster	Define el ambiente de simulación.
Psout	Array (DBL)	Posición actual del robot.
Psin	Array (DBL)	Posición anterior del robot.
speeds	Array (DBL)	ω de motores (rad/s).
sensores	Array (DBL)	Activación de sensores de distancia.
regsz	Cluster (DBL)	Dimensiones de longitud del ambiente.
int_in	Cluster	Variables requeridas por la función.
const	Cluster	Constantes requeridas por la función.
Trials	Numérica (I32)	Número de intentos ya ejecutados.
good?	Booleana	Indica si el robot llegó a la Meta.
crush	Booleana	Indica (T) cuando ocurren colisiones.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
Fit	Array (DBL)	Fitness resultante.
int_out	Cluster	Variables actualizadas por la función.
ModoFit	Seleccionador (Enum)	Forma de combinar Fitness por trial.
stop	Booleana	Detiene la simulación.

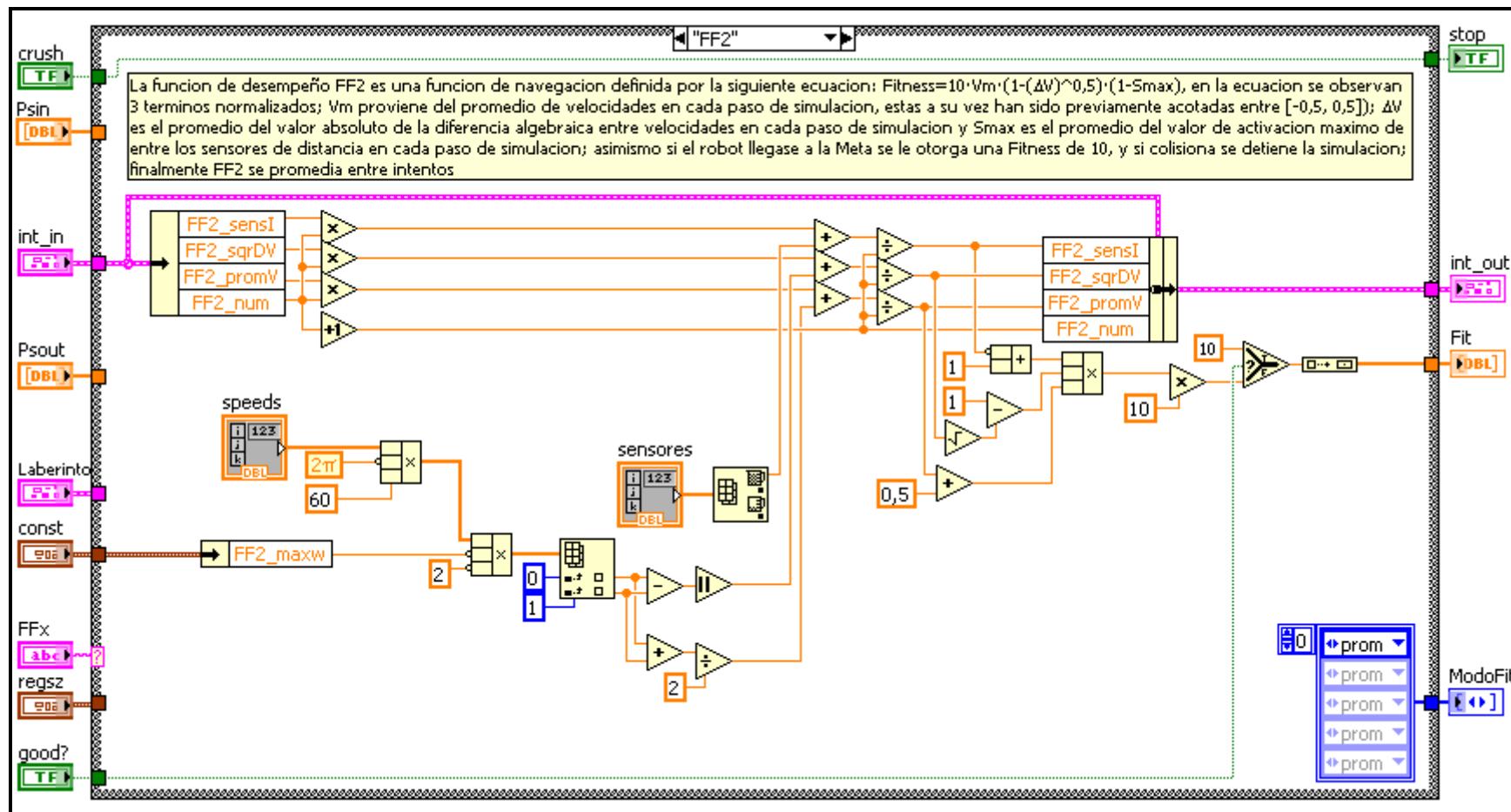
Experiencia 1: Función Fitness – FF1 (Diagrama de Bloques)



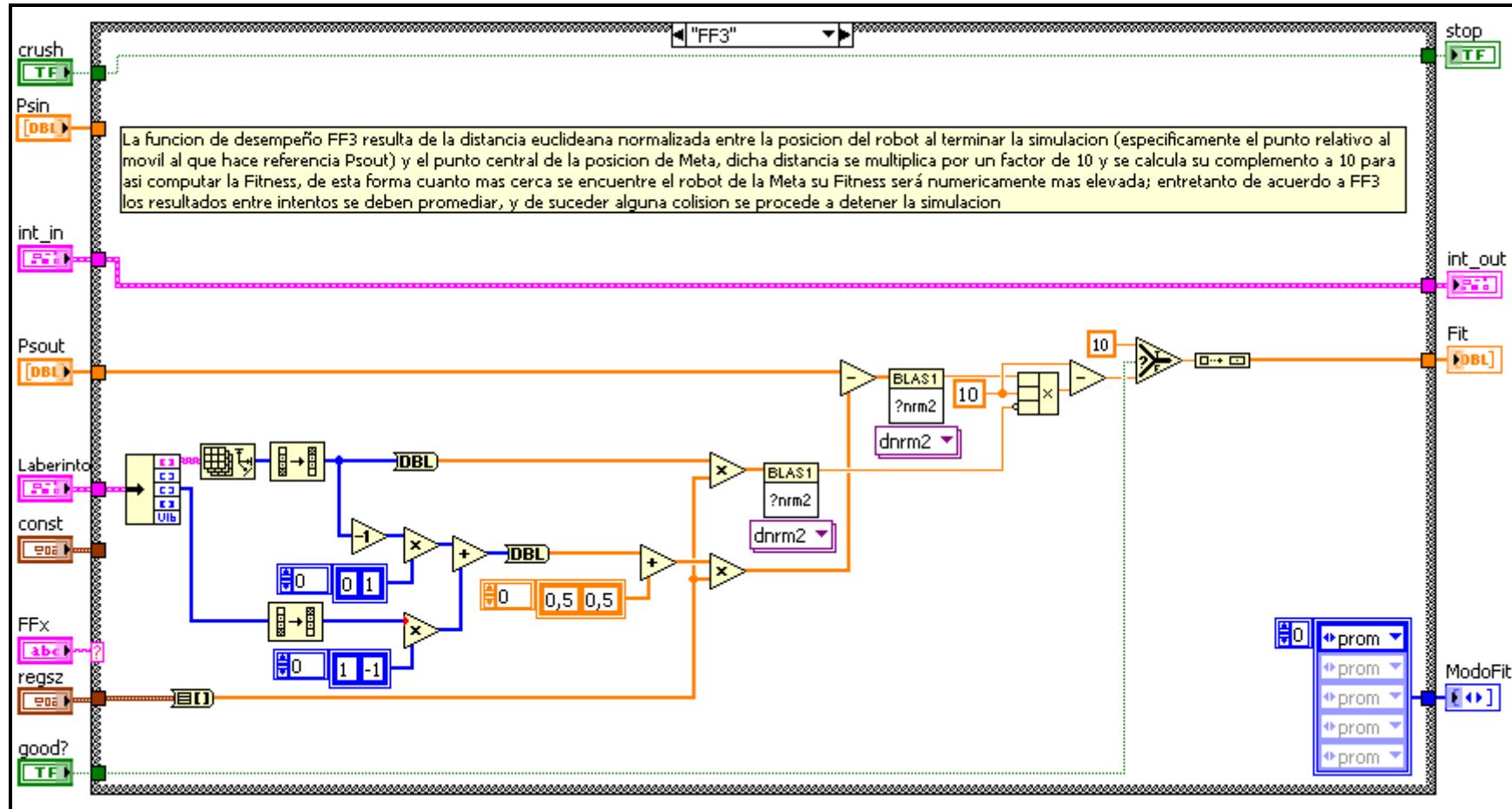
Experiencia 1: Función Fitness – FF1 (subVIs)



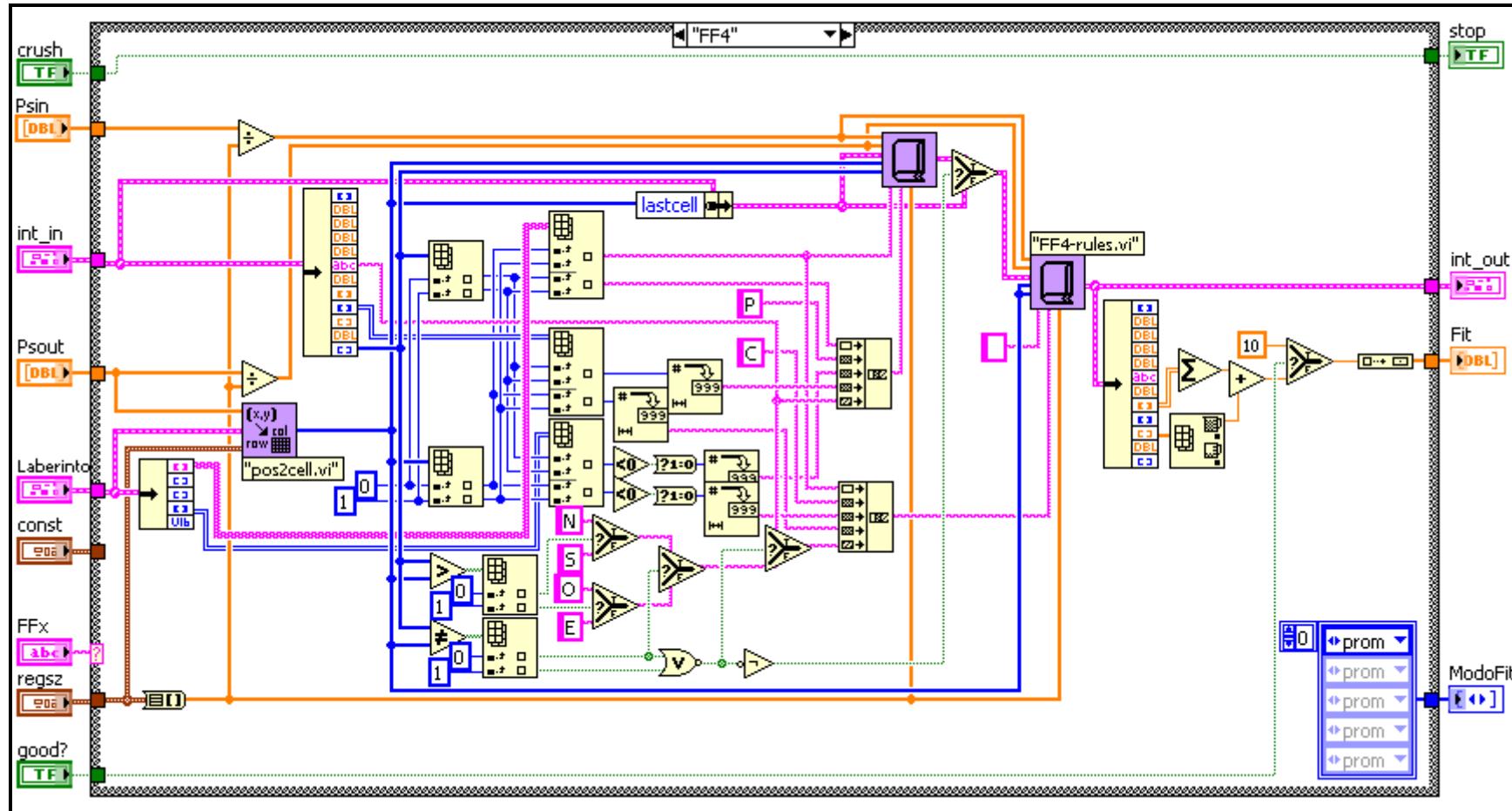
Experiencia 1: Función Fitness – FF2 (Diagrama de Bloques)



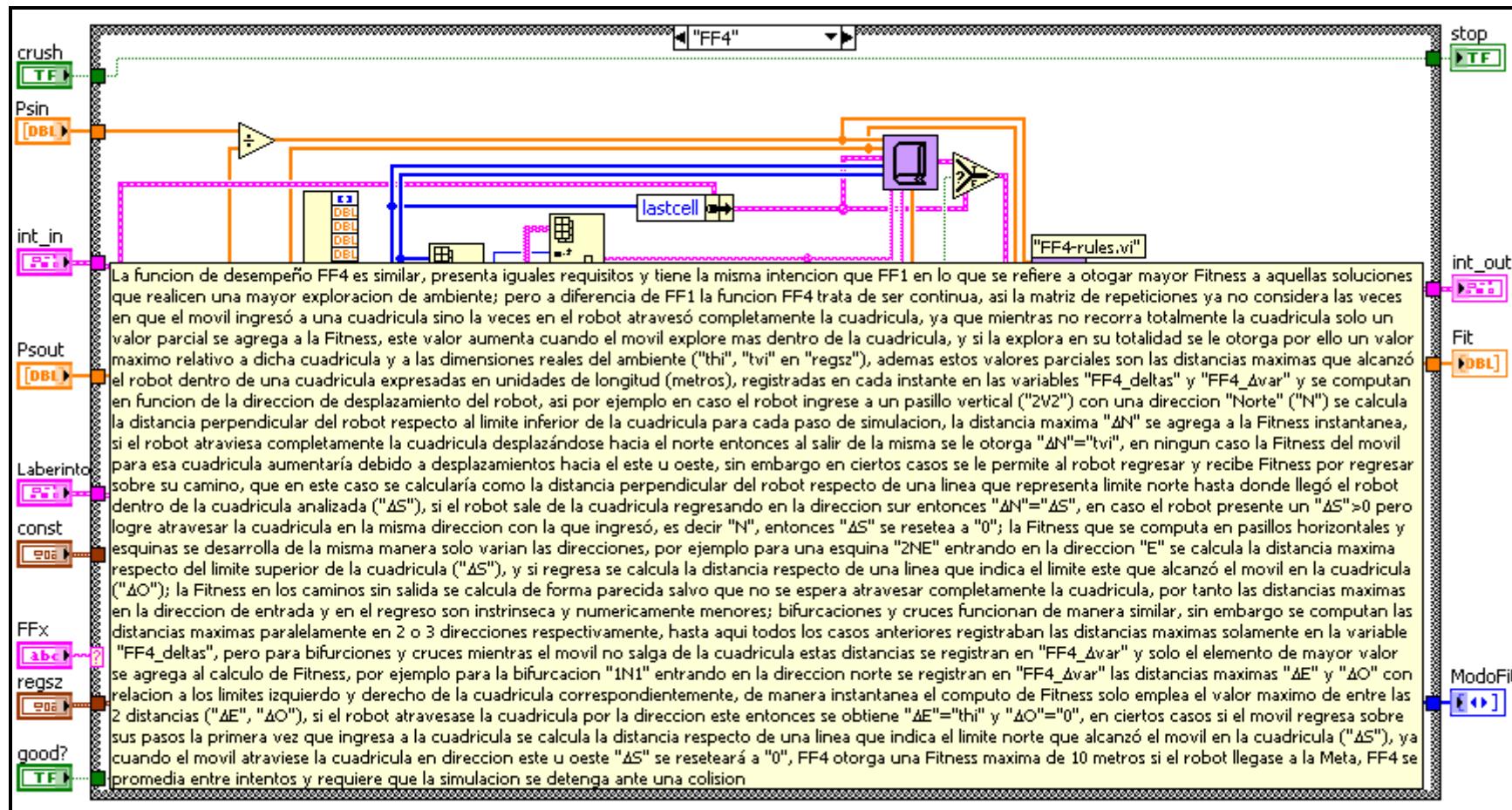
Experiencia 1: Función Fitness – FF3 (Diagrama de Bloques)



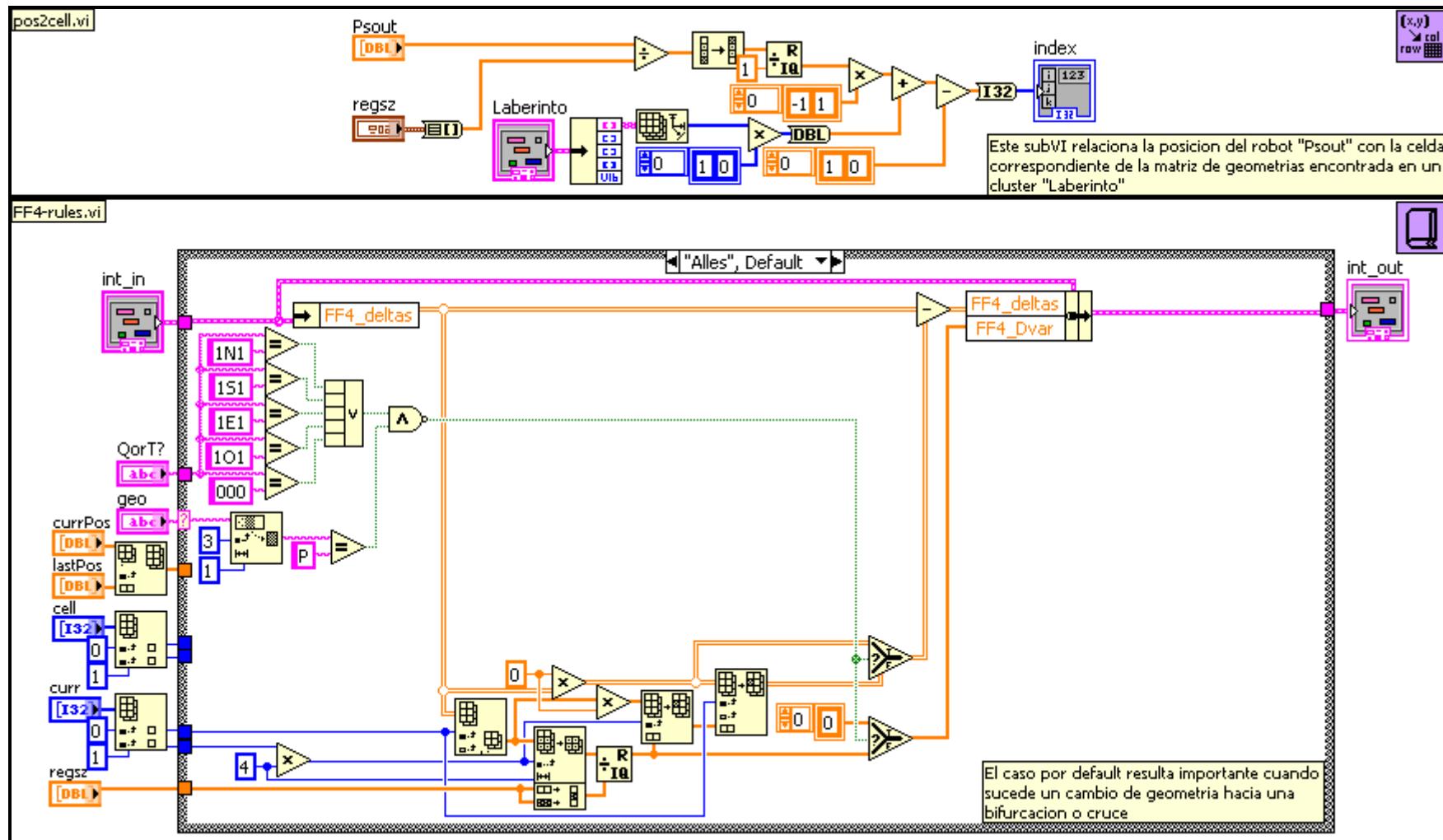
Experiencia 1: Función Fitness – FF4 (Diagrama de Bloques)



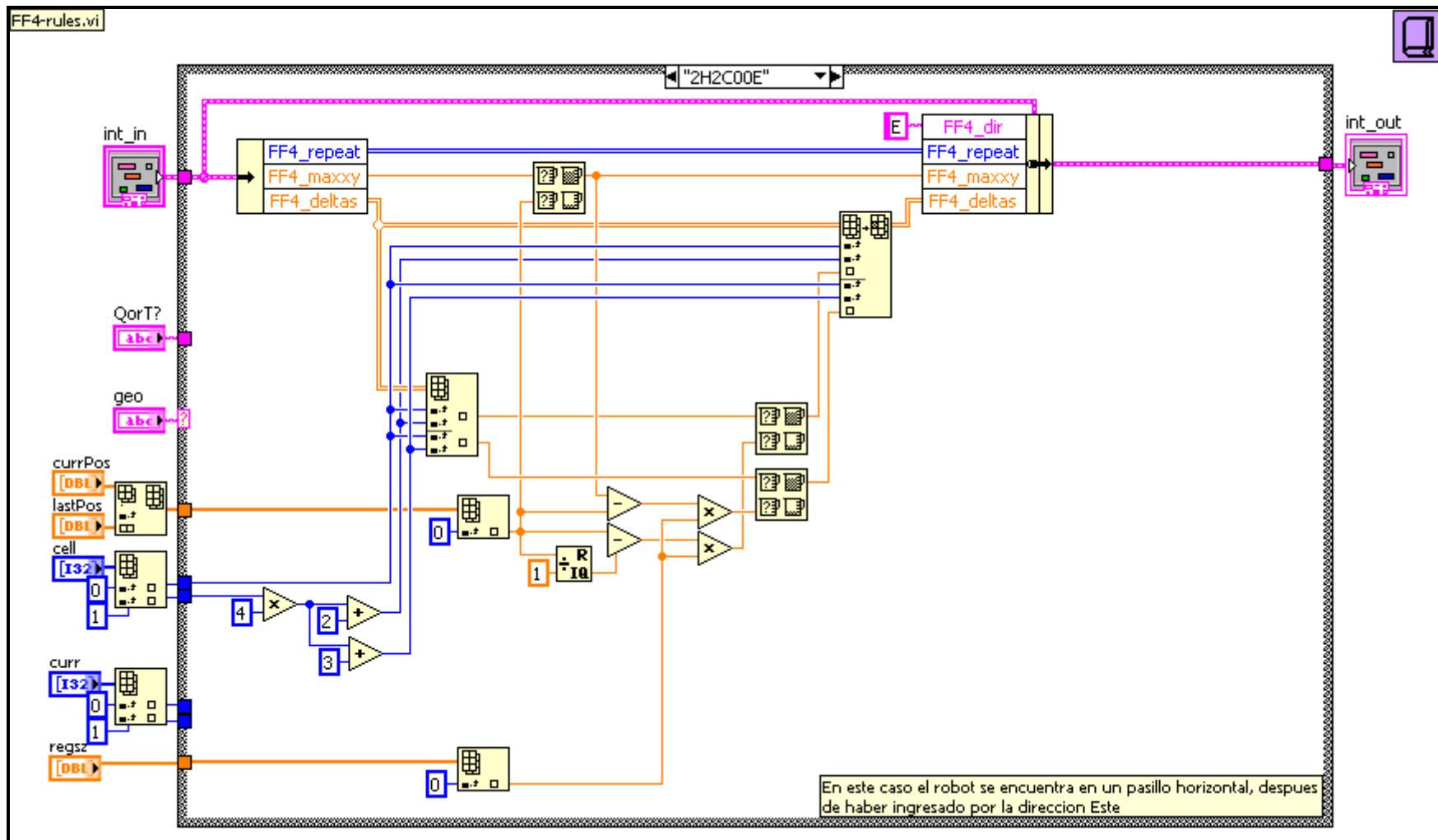
Experiencia 1: Función Fitness – FF4 (continuación)



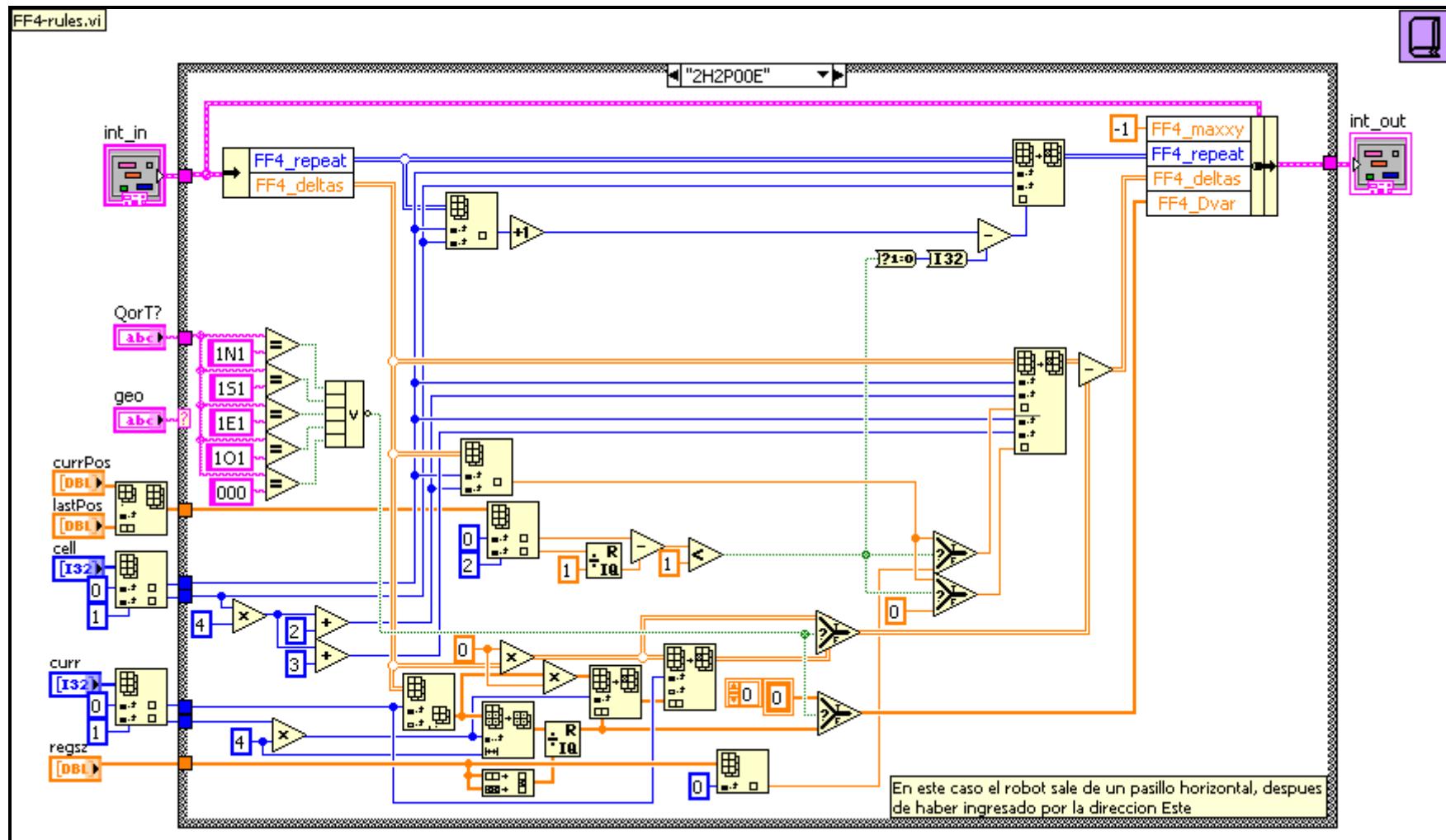
Experiencia 1: Función Fitness – FF4 (subVIs)



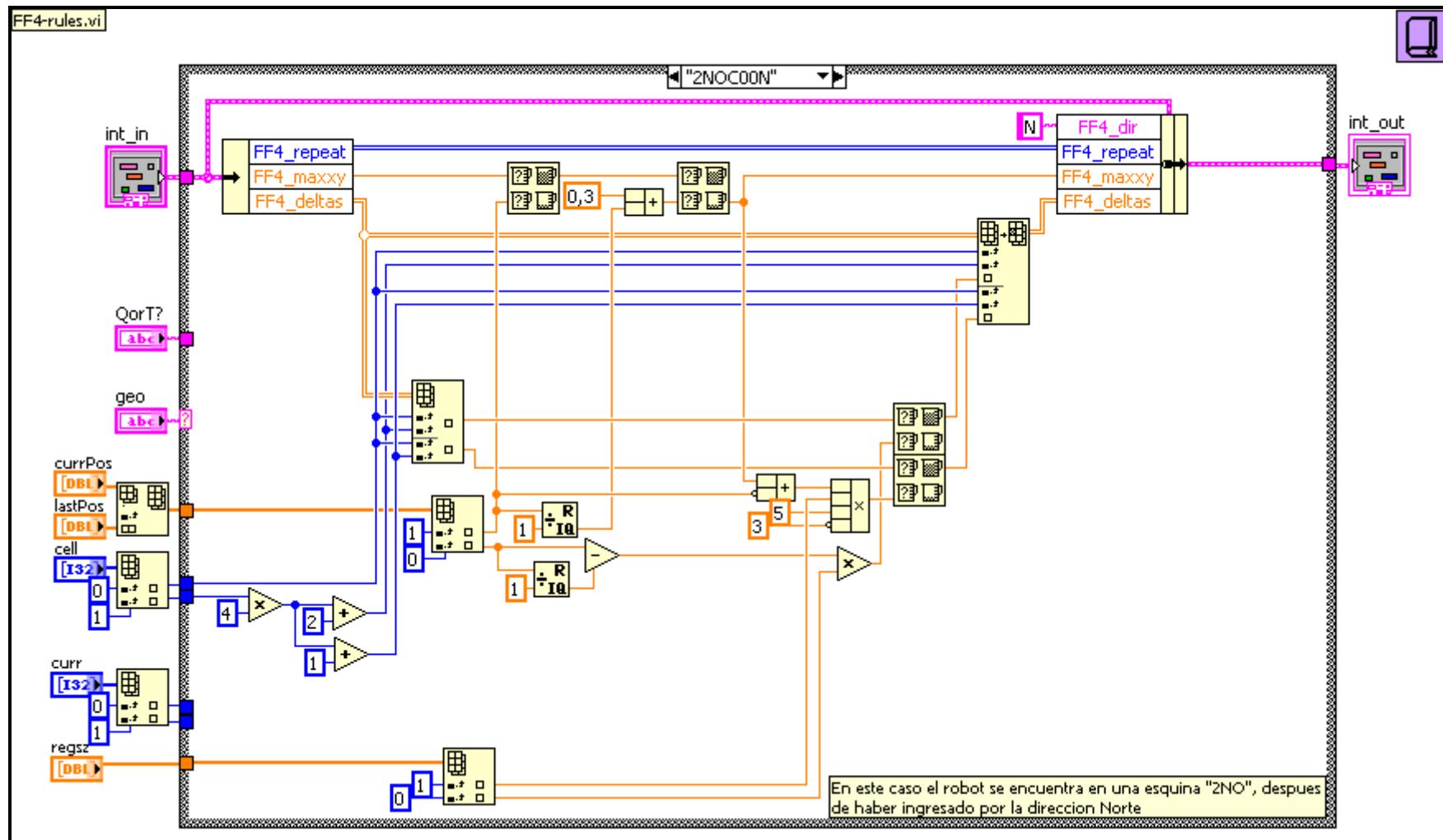
Experiencia 1: Función Fitness – FF4 (subVIs)



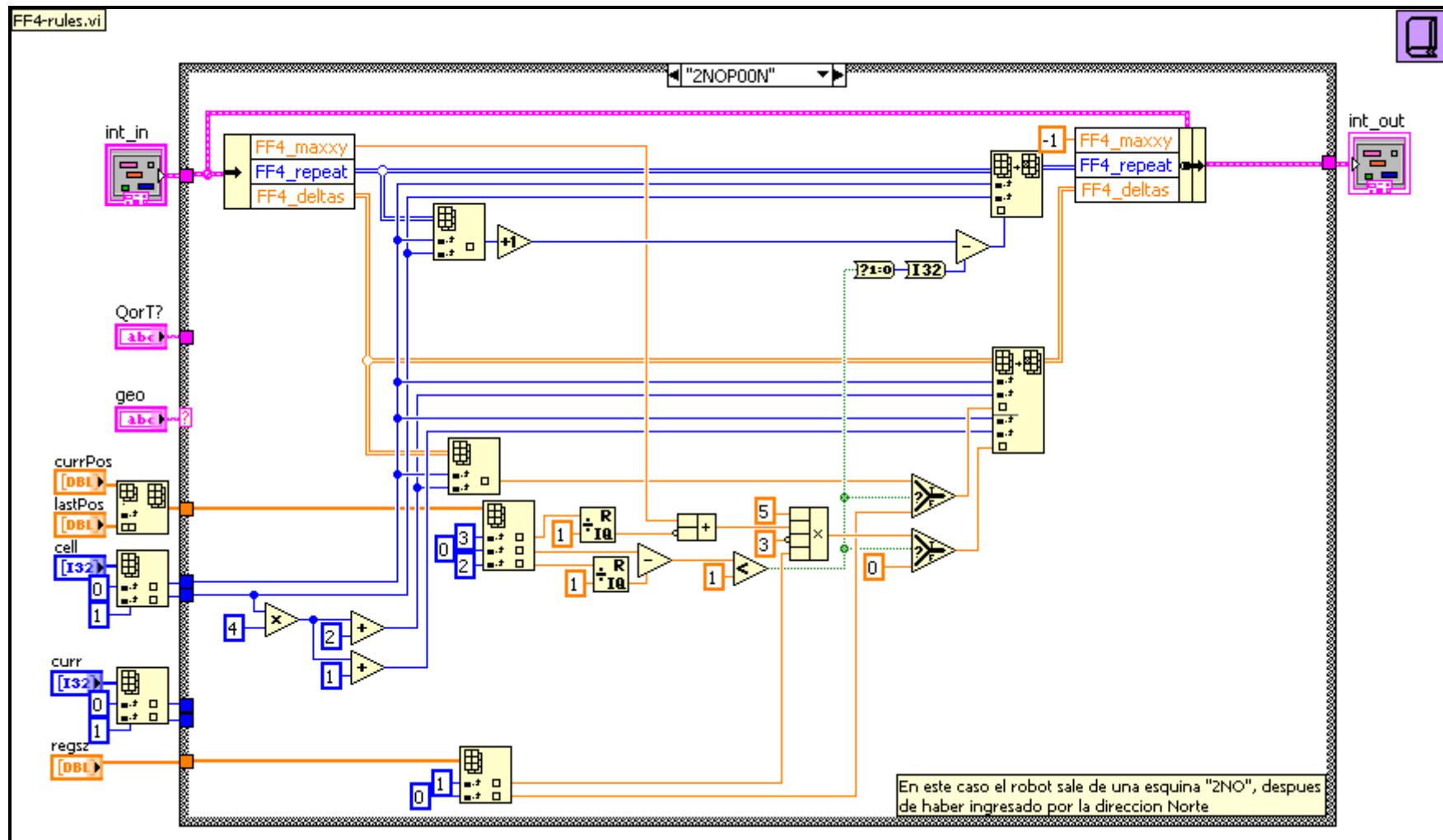
Experiencia 1: Función Fitness – FF4 (subVIs)



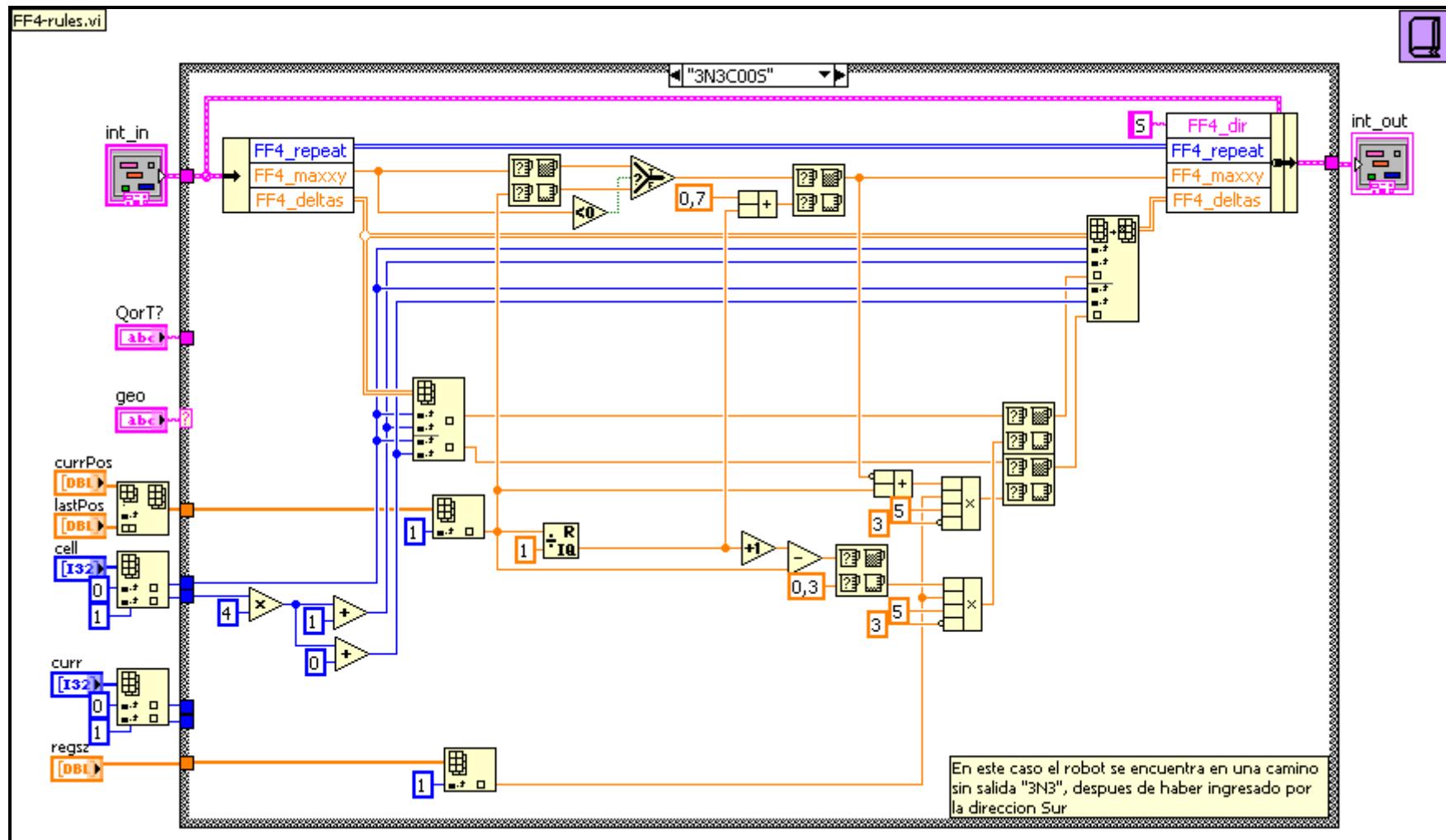
Experiencia 1: Función Fitness – FF4 (subVIs)



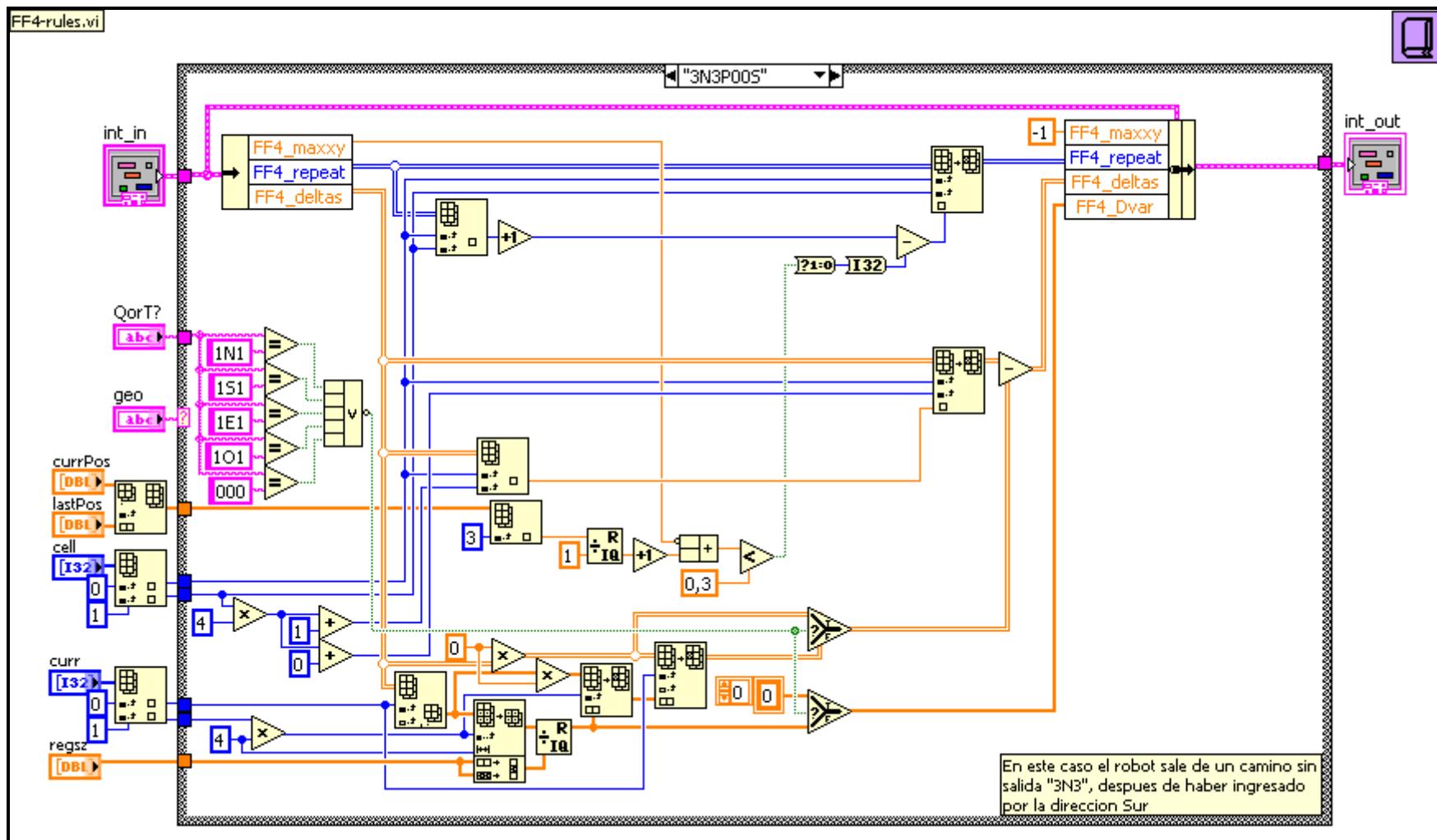
Experiencia 1: Función Fitness – FF4 (subVIs)



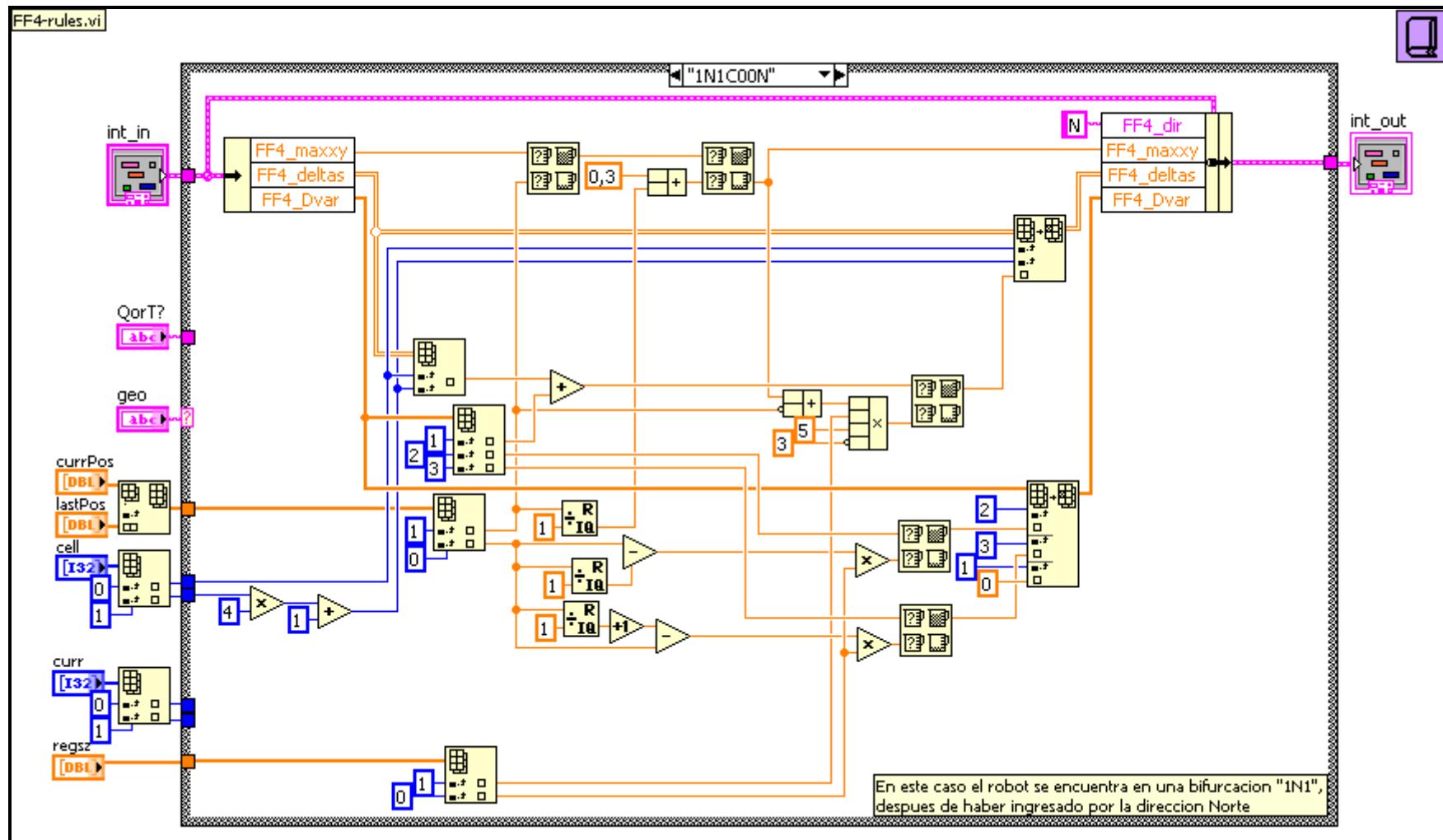
Experiencia 1: Función Fitness – FF4 (subVIs)



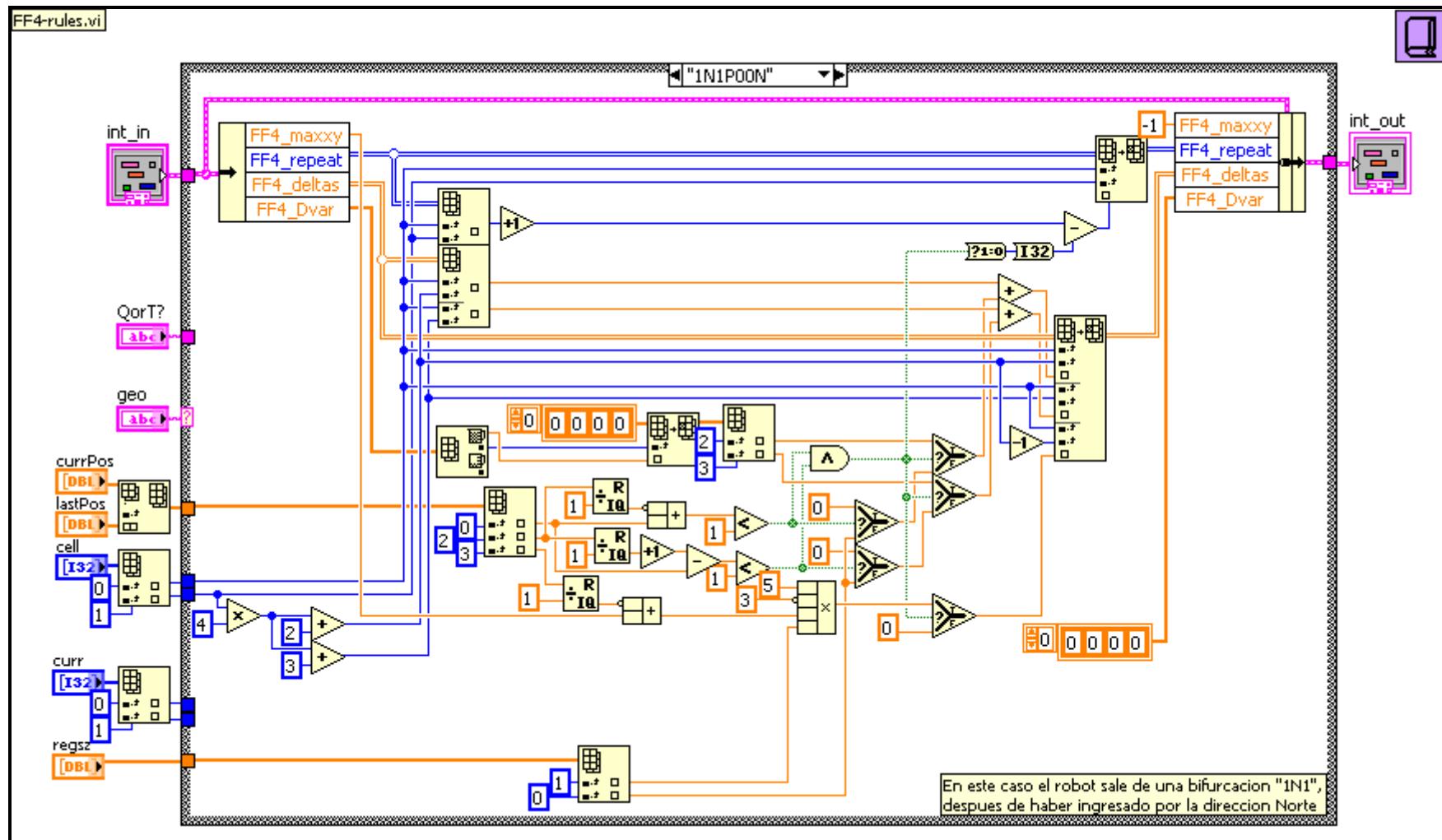
Experiencia 1: Función Fitness – FF4 (subVIs)



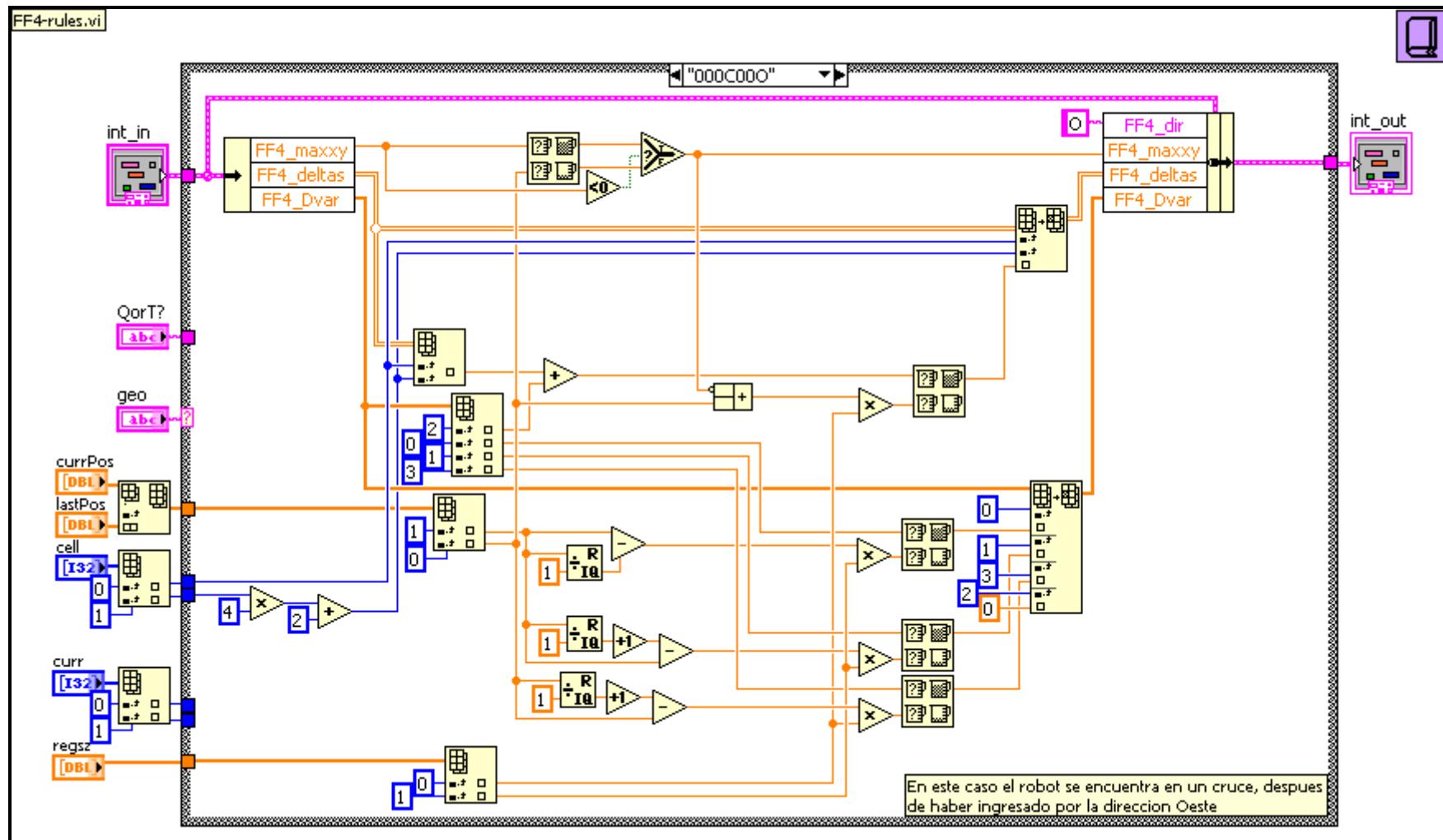
Experiencia 1: Función Fitness – FF4 (subVIs)



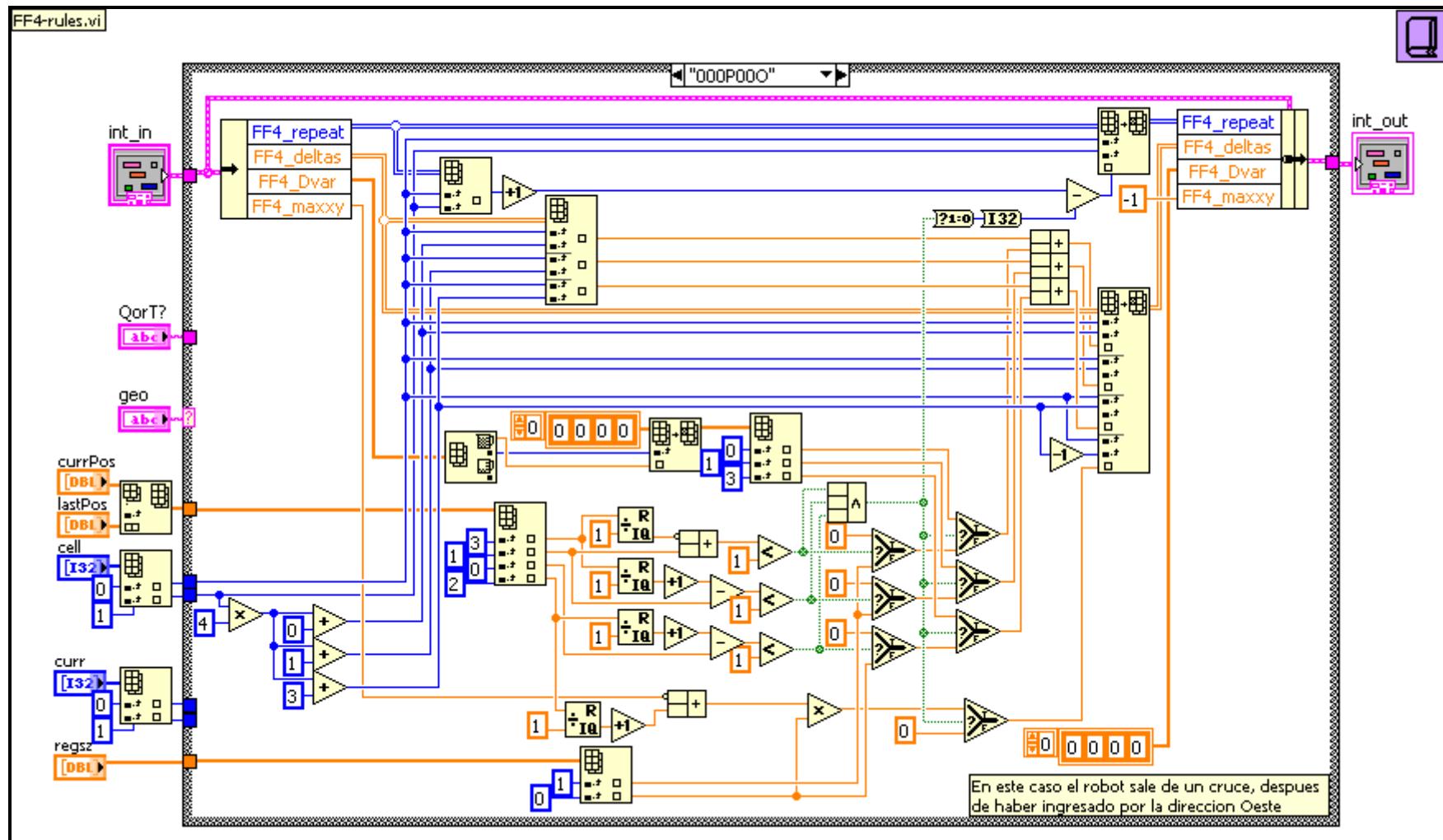
Experiencia 1: Función Fitness – FF4 (subVIs)



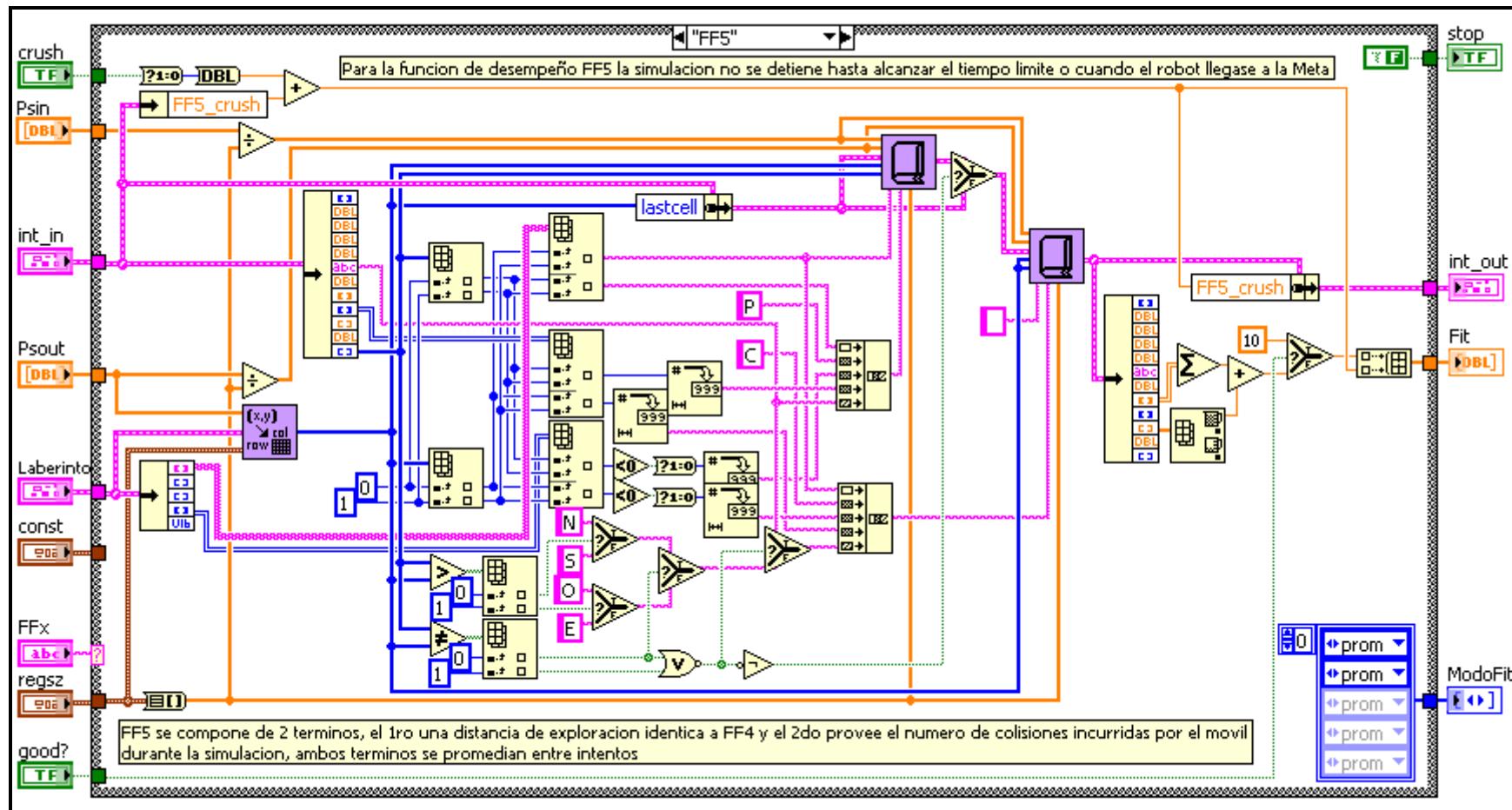
Experiencia 1: Función Fitness – FF4 (subVIs)



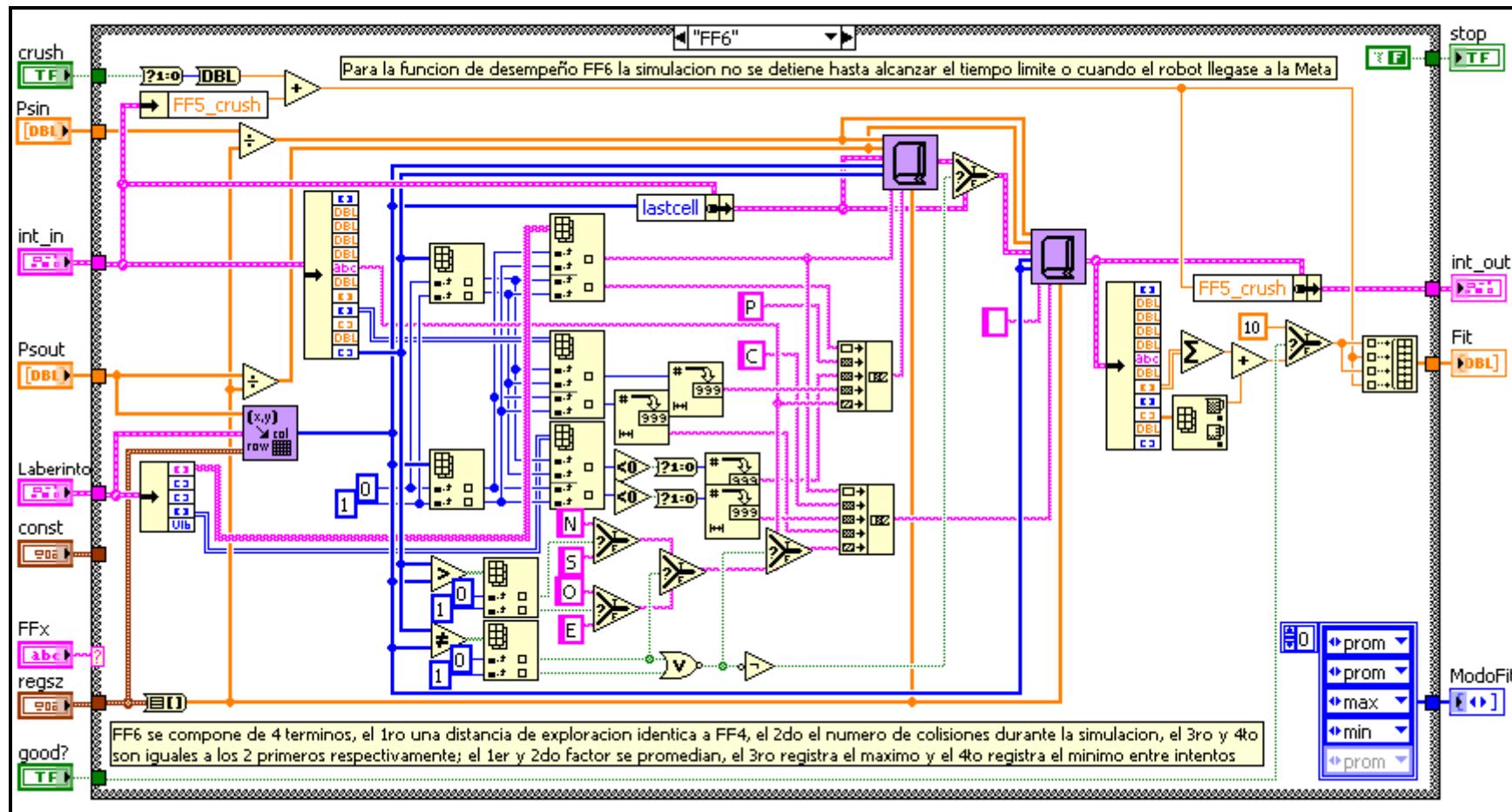
Experiencia 1: Función Fitness – FF4 (subVIs)



Experiencia 1: Función Fitness – FF5 (Diagrama de Bloques)



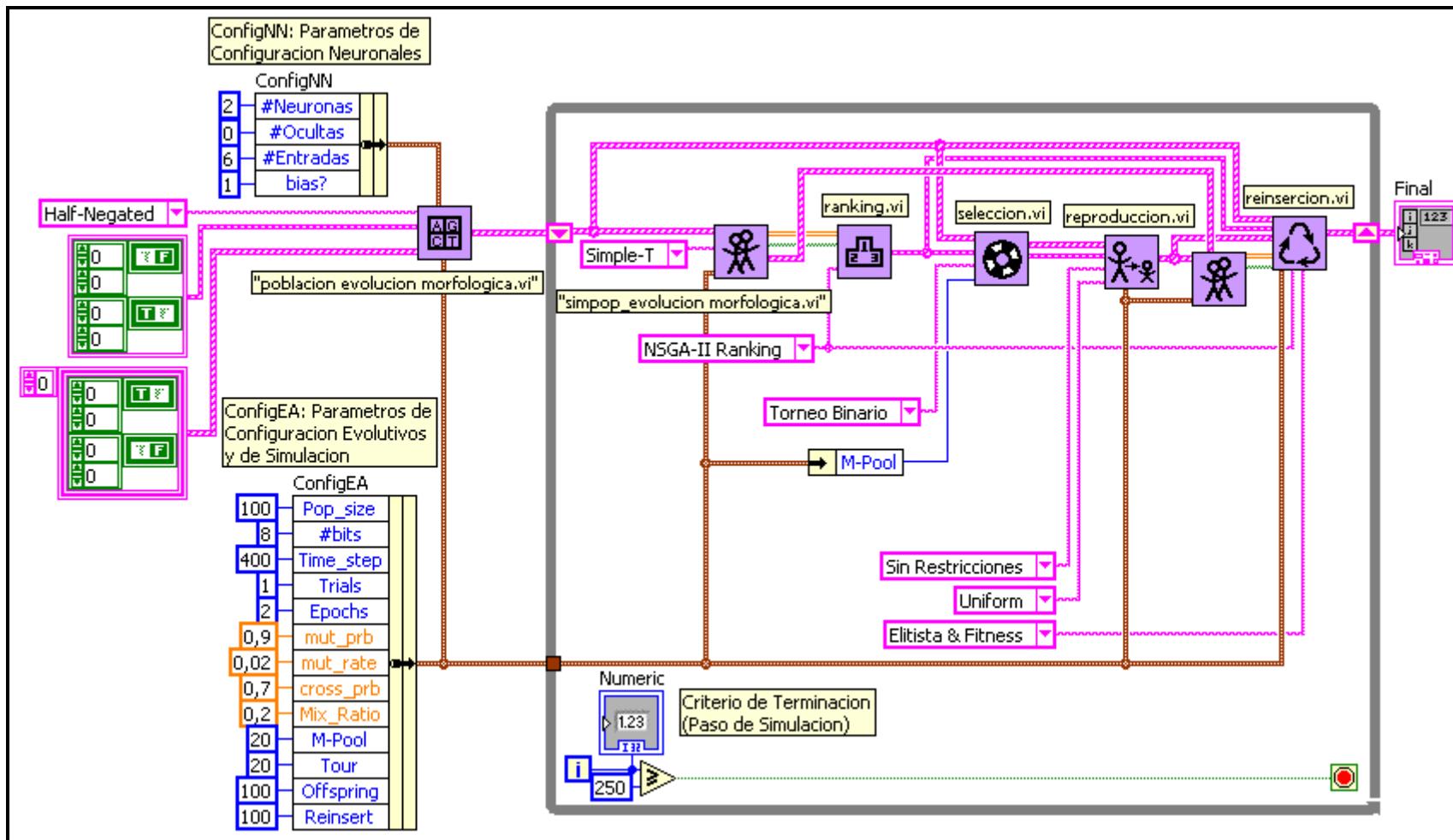
Experiencia 1: Función Fitness – FF6 (Diagrama de Bloques)



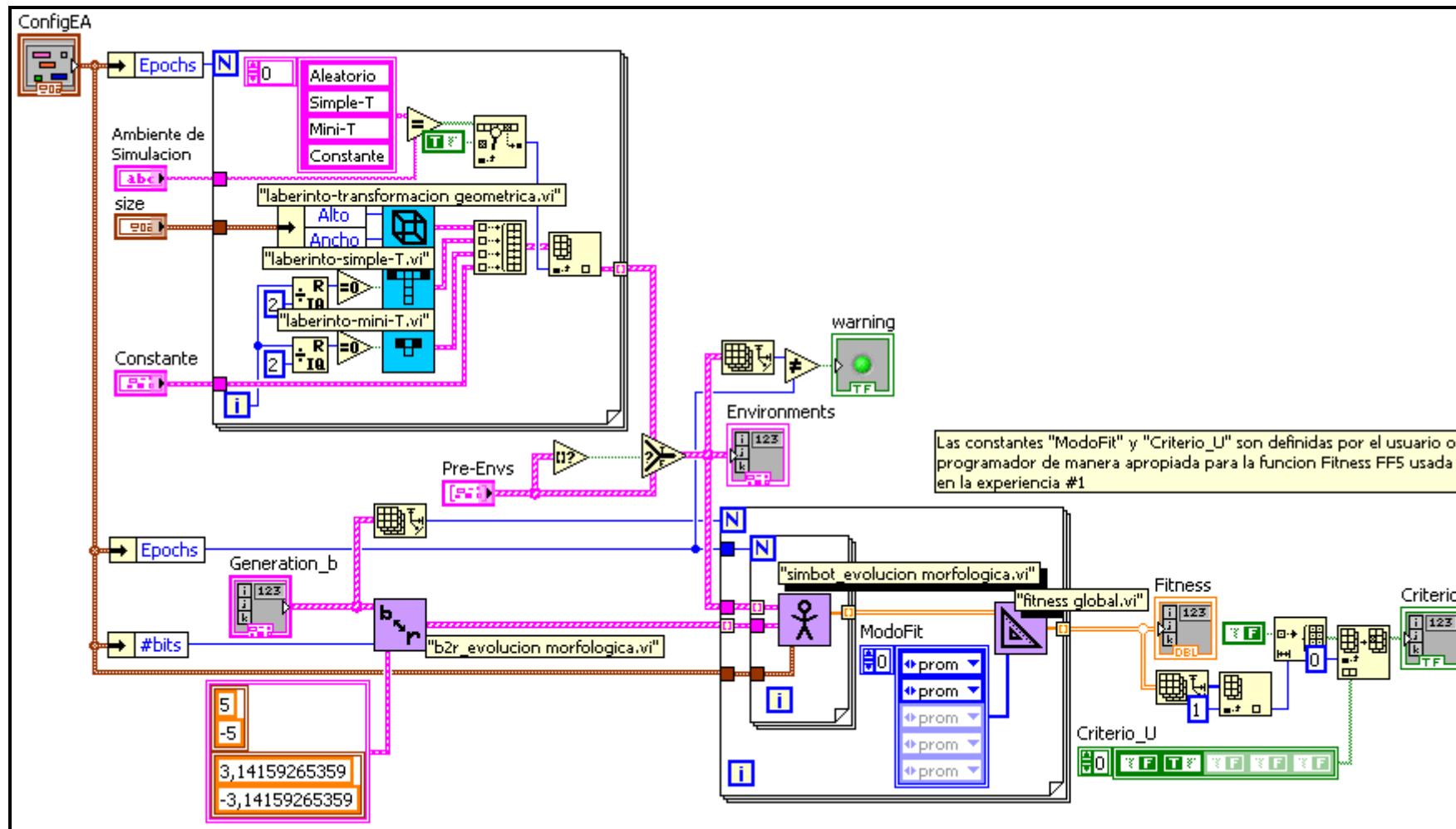
ANEXO J

DESARROLLO ALGORITMICO – EXP#2: EVOLUCION MORFOLOGICA

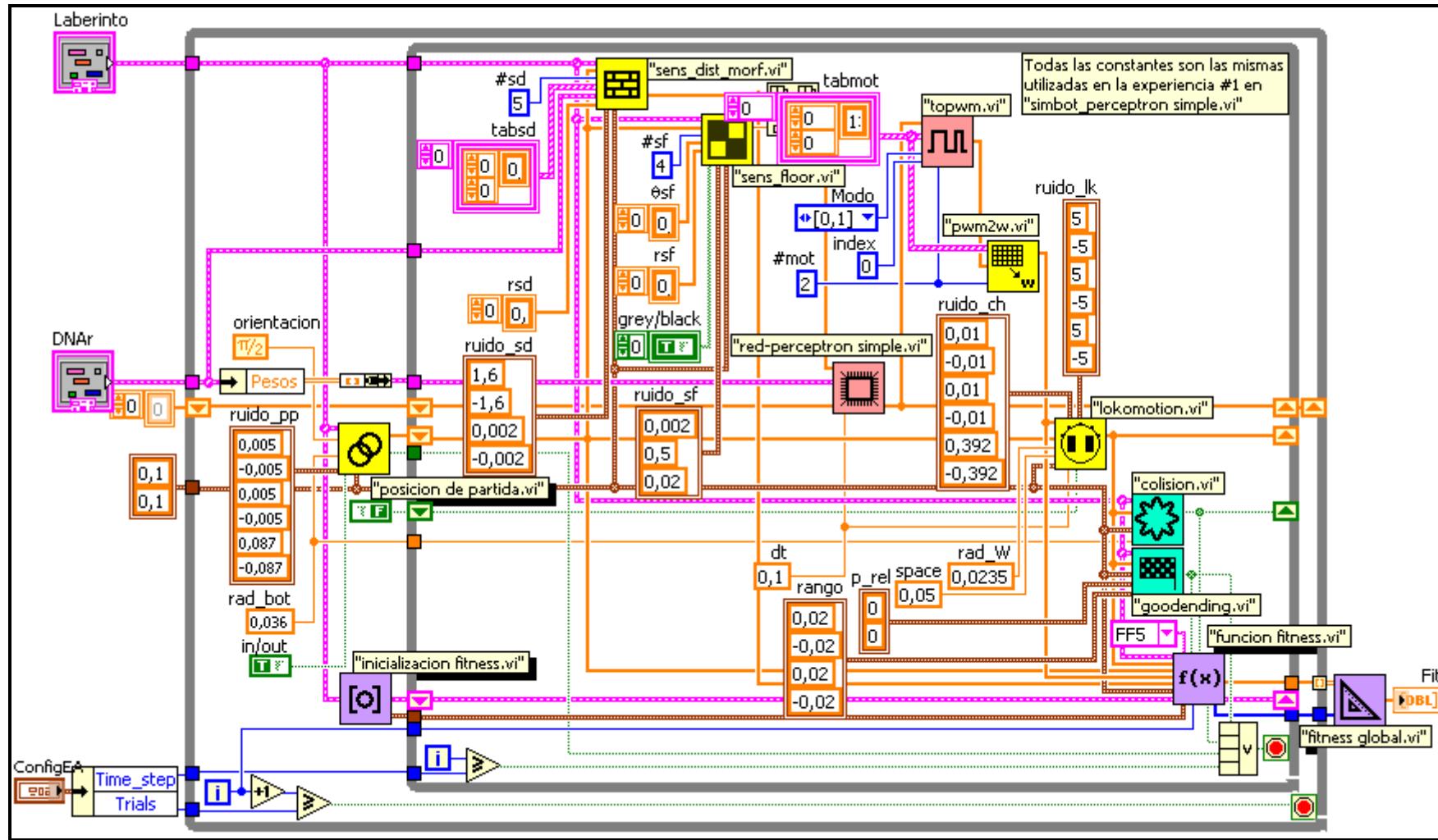
Experiencia 2: Algoritmo Evolutivo – Evolución Morfológica (Diagrama de Bloques)



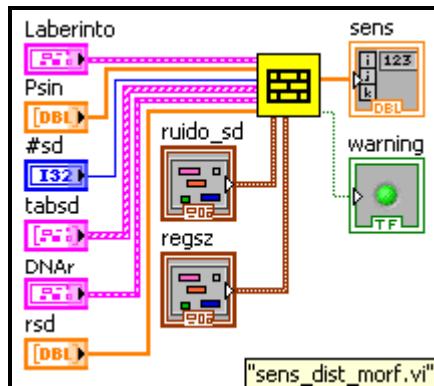
Experiencia 2: Simulación de la Población - Evolución Morfológica (Diagrama de Bloques)



Experiencia 2: Simulación de un Individuo - Evolución Morfológica (Diagrama de Bloques)

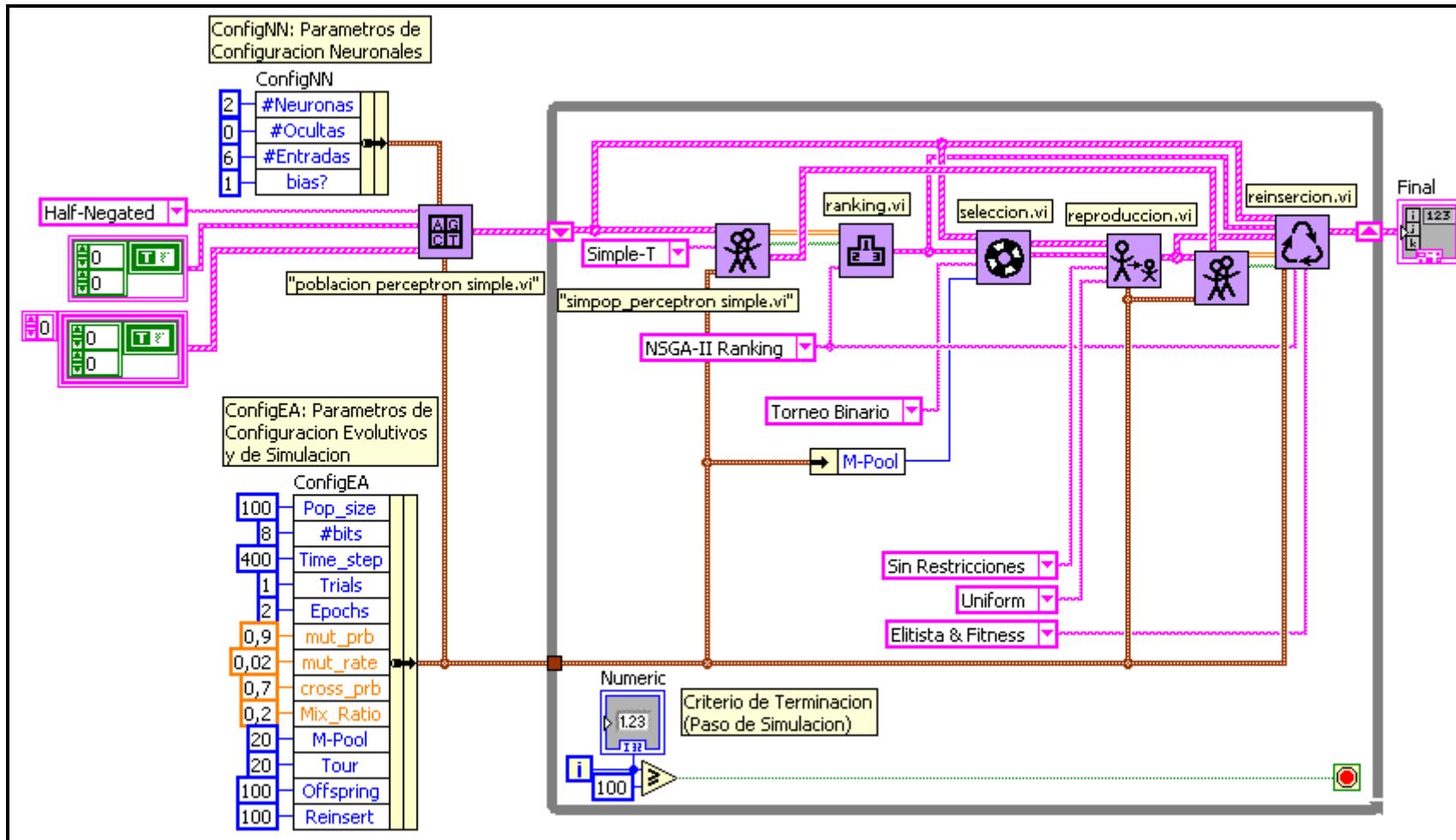


Experiencia 2: Sensores de Distancia - Evolución Morfológica

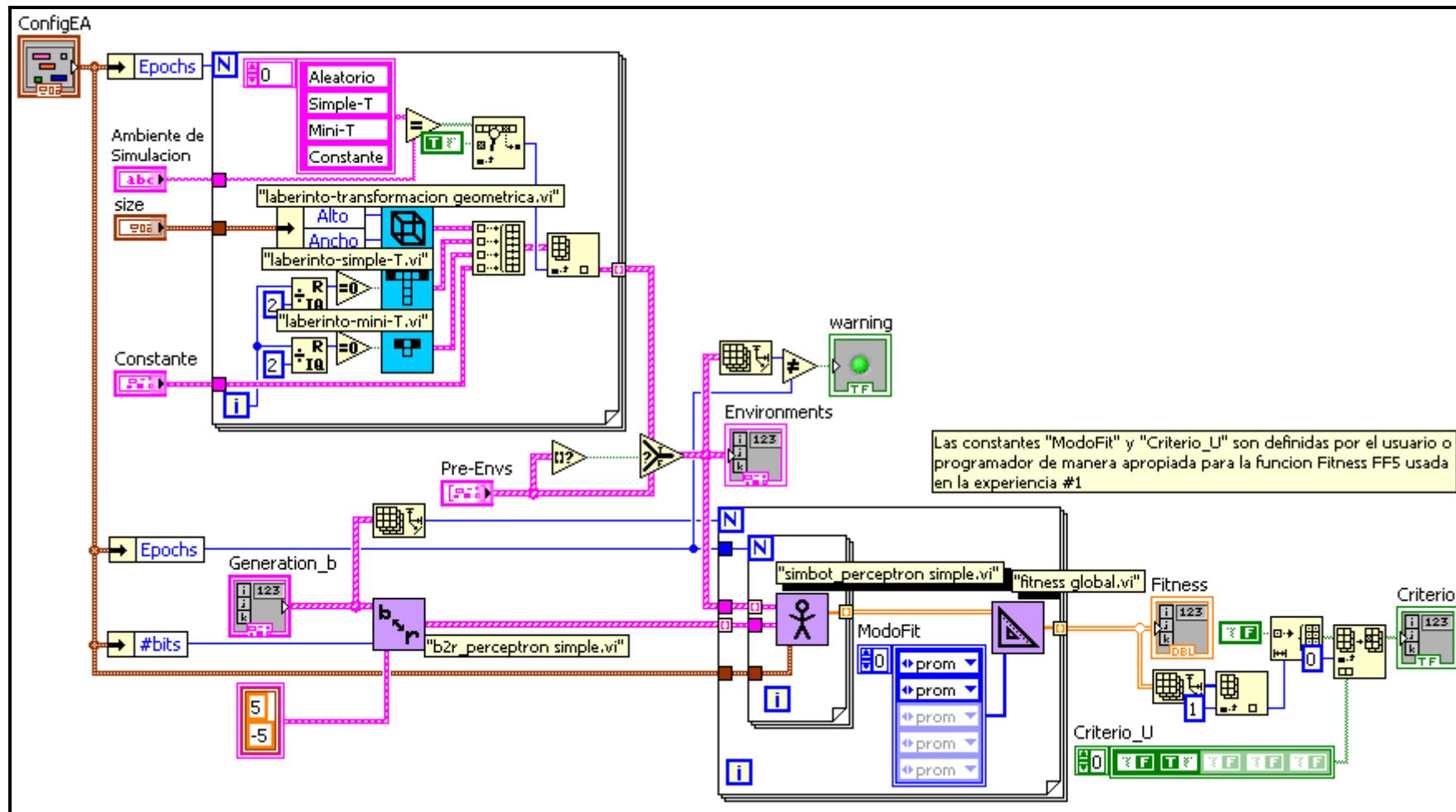


FUNCION		
Determina el estado de activación de cada sensor de distancia.		
ENTRADAS		
NOMBRE	TIPO	DESCRIPCION
Laberinto	Cluster	Define el ambiente de simulación.
Psin	Array (x, y, θ)	Define posición y orientación del robot.
#sd	Numérica (I32)	Define el número de sensores usados.
tabsd	Array de Clusters	Define para cada sensor una tabla de valores que indica su comportamiento.
DNAr	Cluster	Contiene el genotipo decodificado para ser empleado en la simulación.
rsd	Array (r1, ..., rn)	Define la distancia radial relativa de cada sensor respecto del punto (x, y) perteneciente al array "Psin".
ruido_sd	Cluster (Δang, Δdist)	Define máximos y mínimos para el ruido que es agregado a la posición angular (grados.) y distancia virtual.
regsz	Cluster (DBL)	Define dimensiones horizontal (thi) y vertical (tvi) de la cuadrícula base que compone el ambiente.
SALIDAS		
NOMBRE	TIPO	DESCRIPCION
sens	Array (sd1, ..., sdn)	Indica la activación de cada sensor.
warning	Booleana	Indica si el tamaño de los arrays de entrada corresponde al número de sensores de distancia.

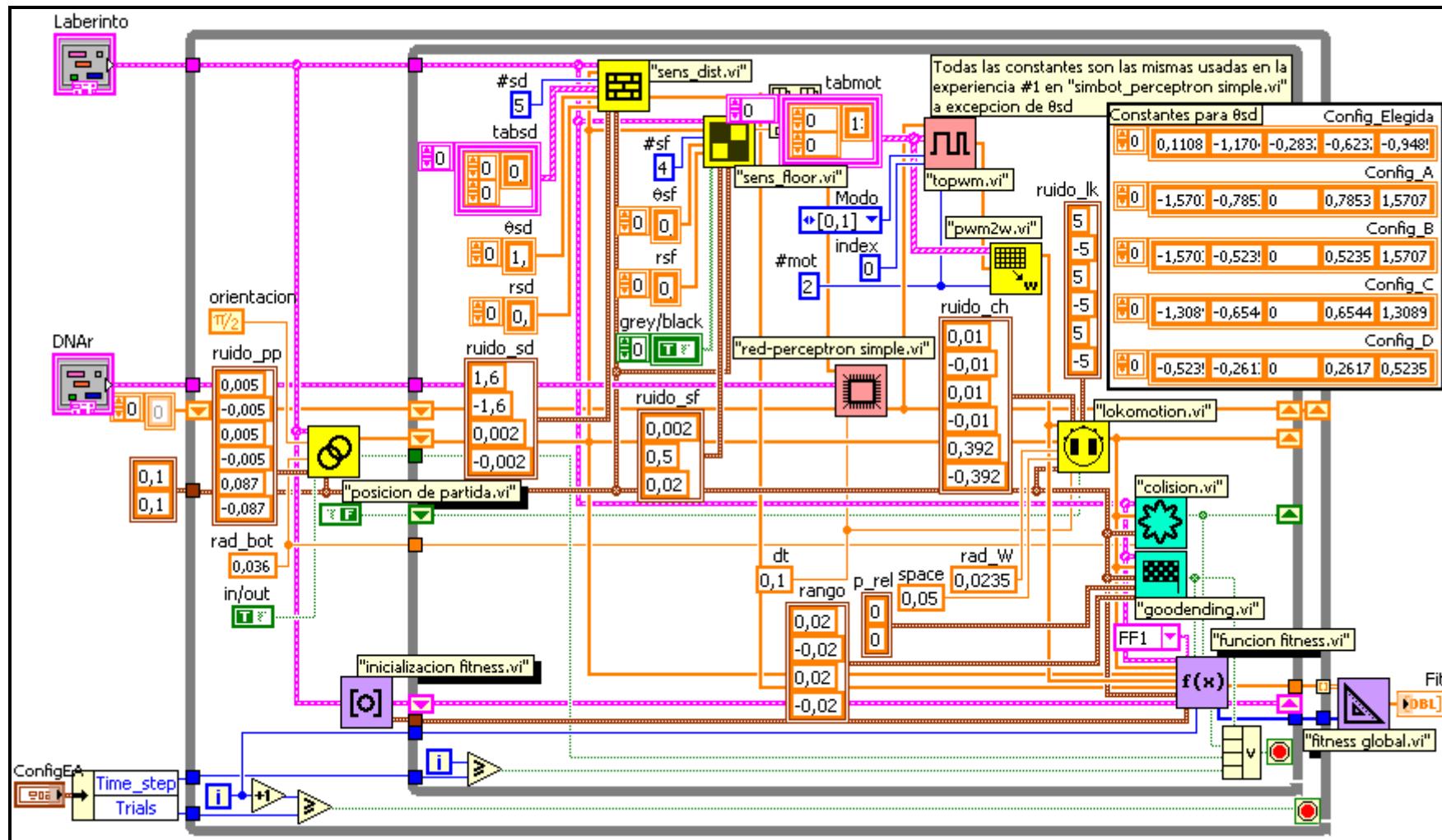
Experiencia 2: Algoritmo Evolutivo – Comparación frente a otras Configuraciones (Diagrama de Bloques)



Experiencia 2: Simulación de la Población - Comparación frente a otras Configuraciones (Diagrama de Bloques)



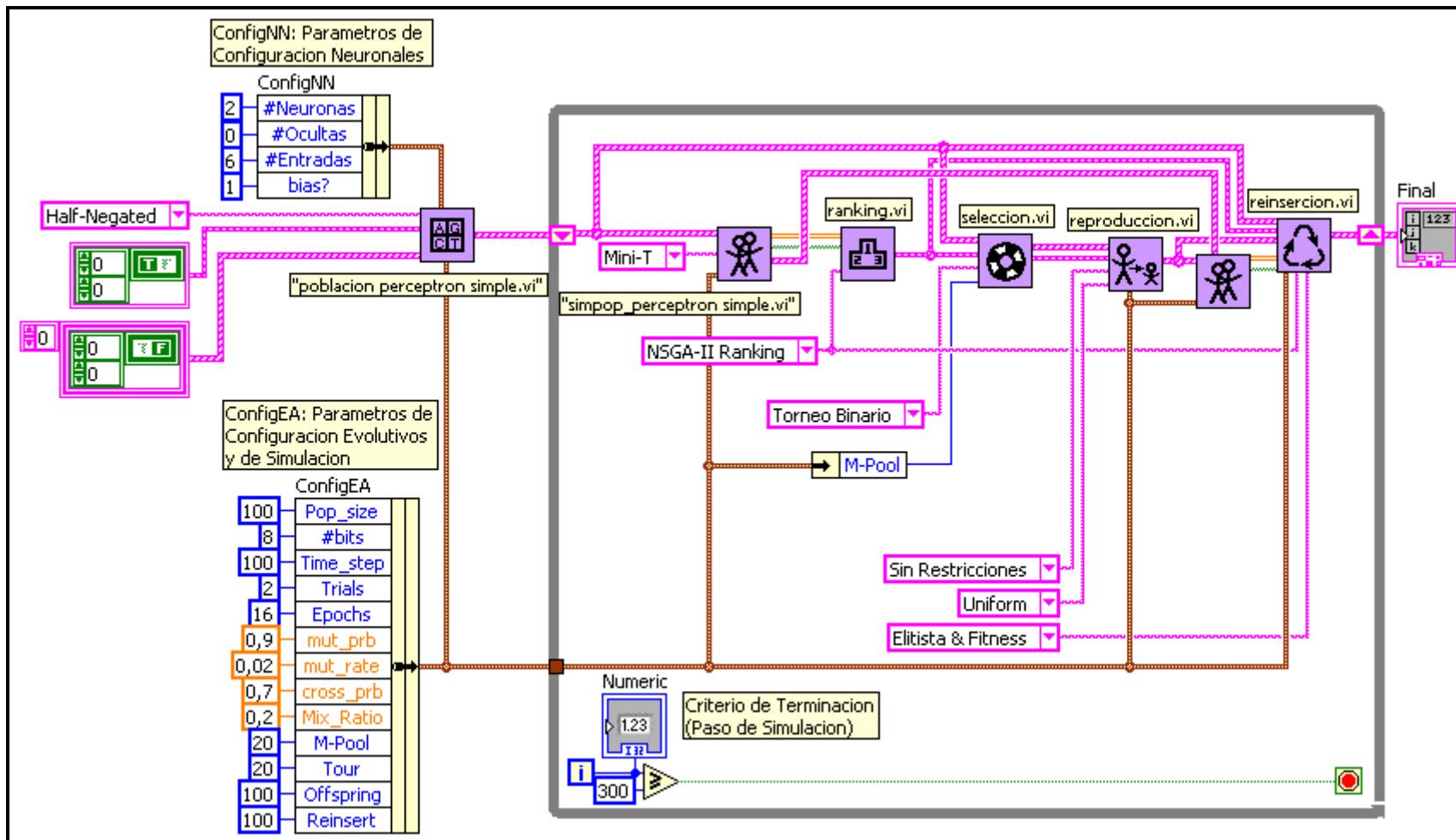
Experiencia 2: Simulación de un Individuo - Comparación frente a otras Configuraciones (Diagrama de Bloques)



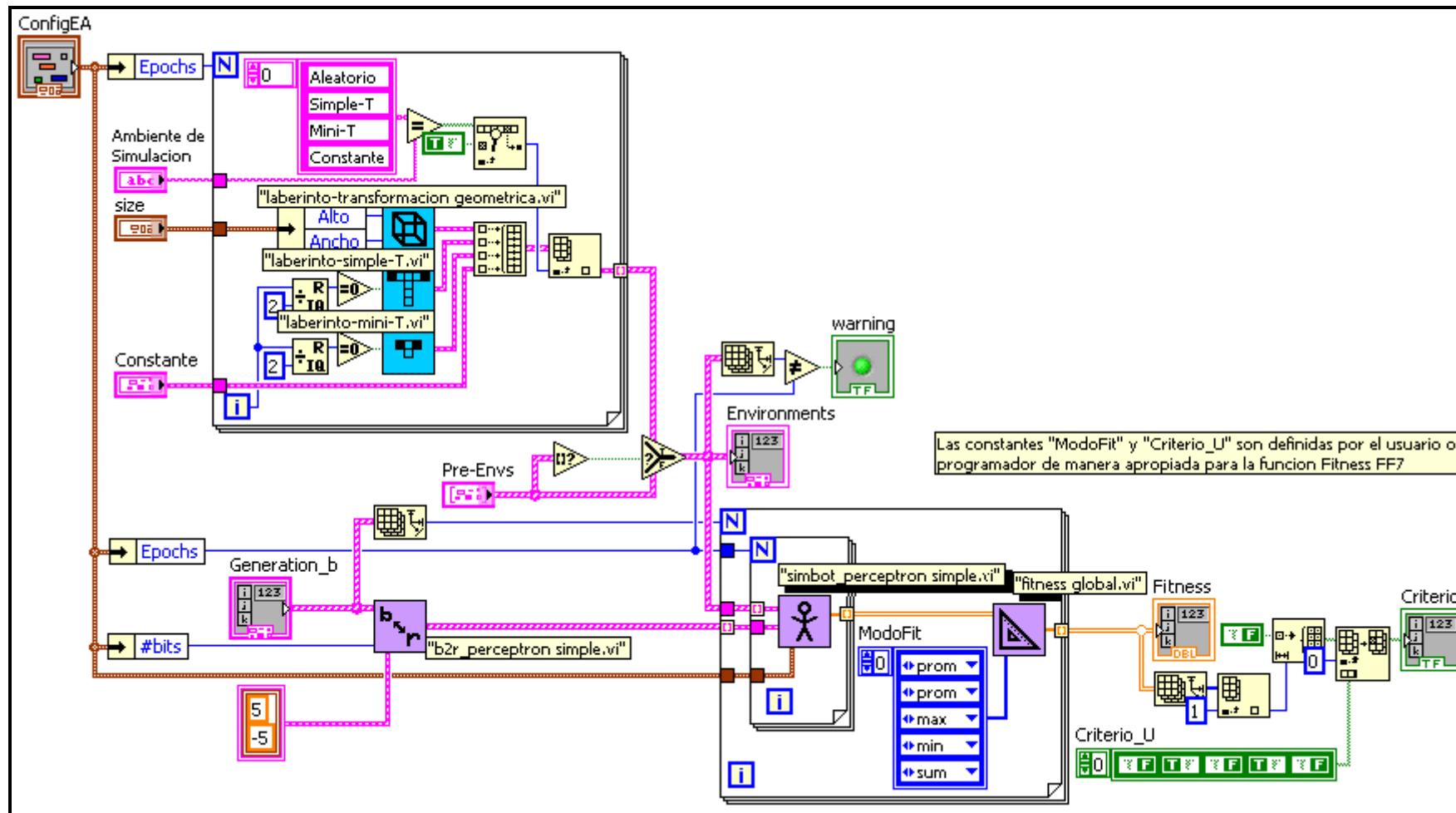
ANEXO K

DESARROLLO ALGORITMICO – EXP#3: COMPORTAMIENTO NO-REACTIVO

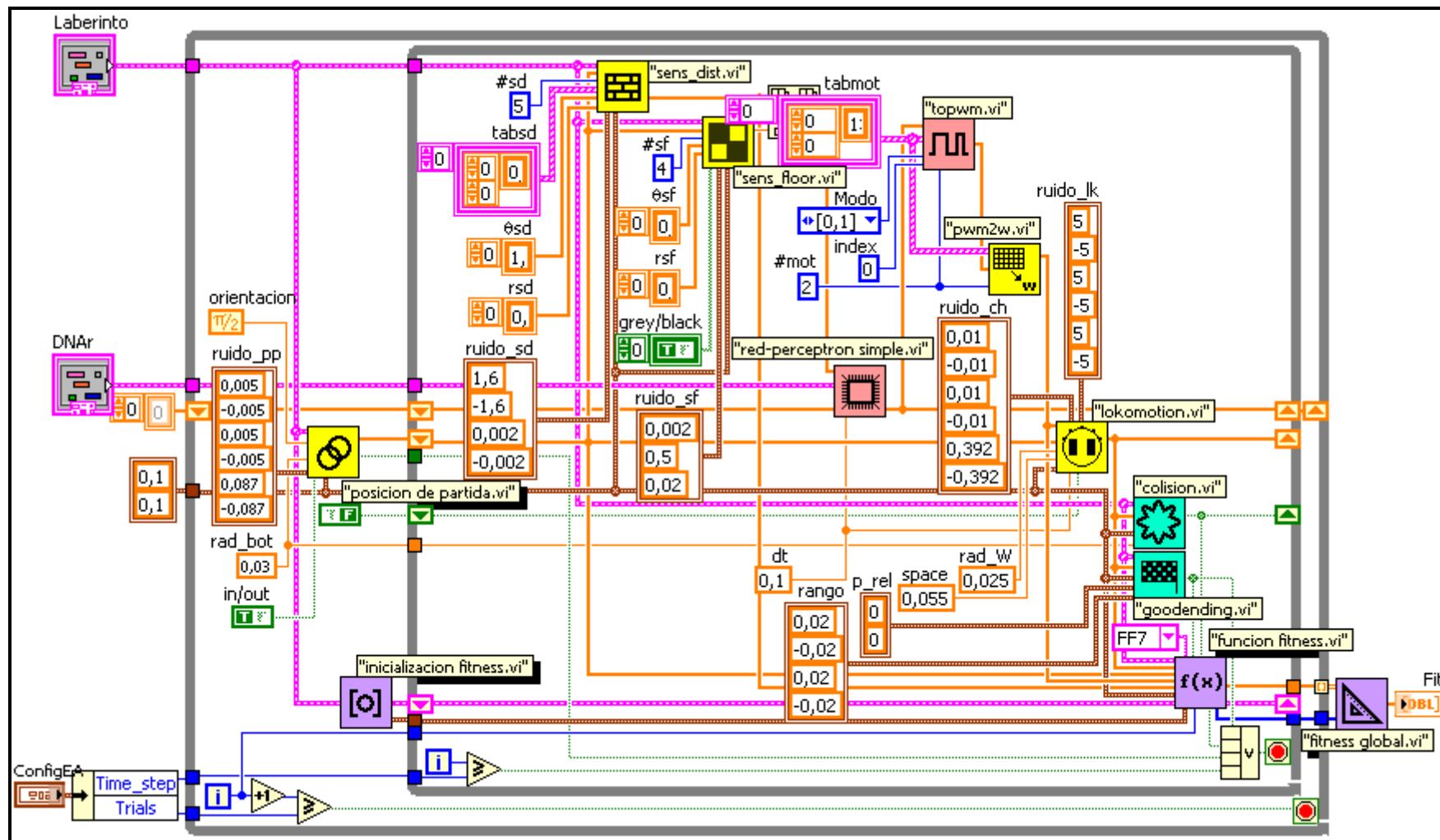
Experiencia 3: Algoritmo Evolutivo – Perceptron Simple (Diagrama de Bloques)



Experiencia 3: Simulación de la Población – Perceptron Simple (Diagrama de Bloques)



Experiencia 3: Simulación de un Individuo – Perceptron Simple (Diagrama de Bloques)



Experiencia 3: Constantes

Se emplea la misma tabla para todos los sensores

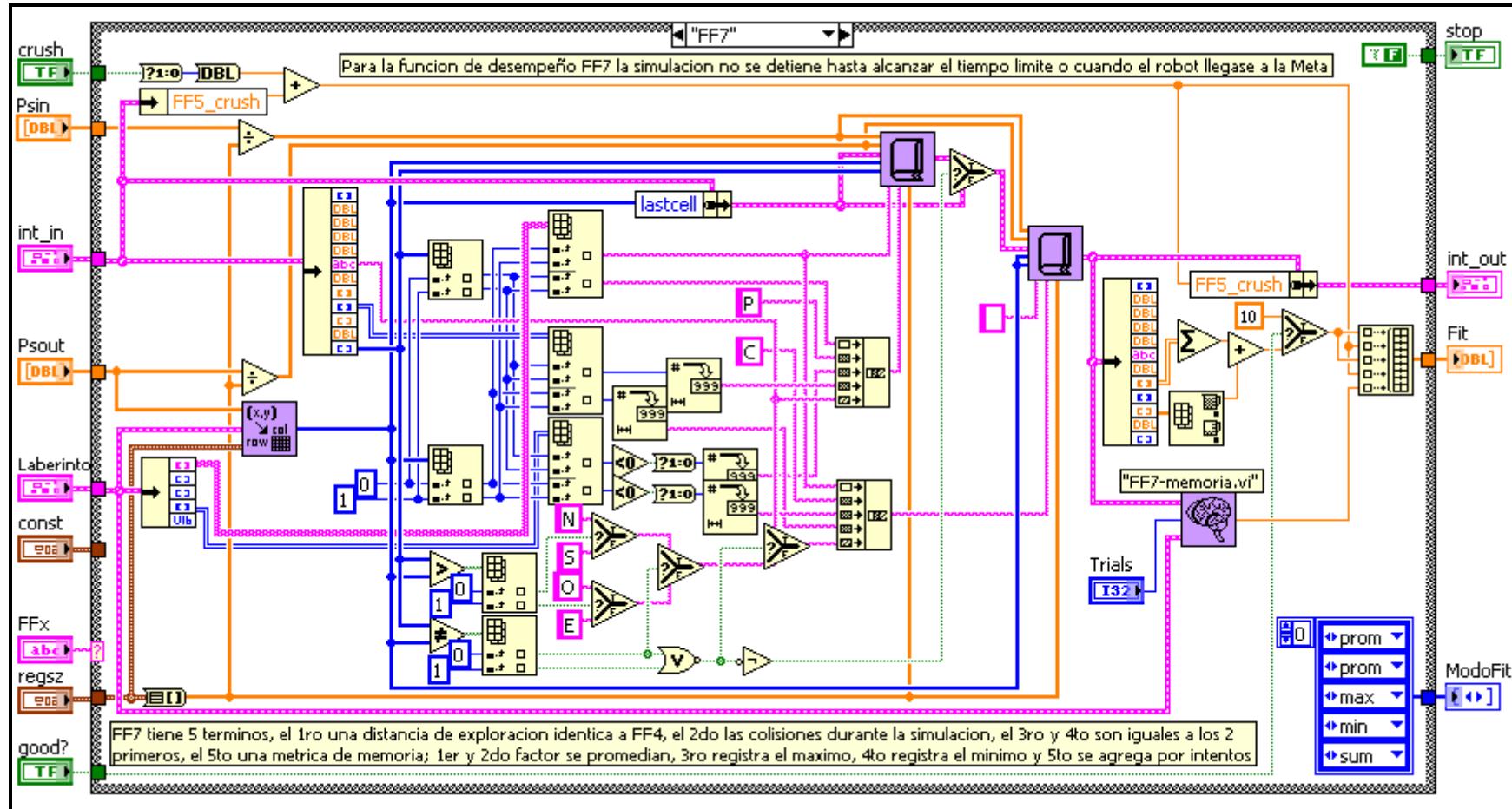
tabsd											
sensor											
0	0	-30	-22,5	-15	-7,5	0	7,5	15	22,5	30	
0	0,002	0,945	0,945	0,945	0,945	0,945	0,945	0,945	0,945	0,945	
0,007	0,584	0,657	0,693	0,729	0,675	0,720	0,711	0,684	0,629		
0,012	0,404	0,440	0,494	0,503	0,449	0,494	0,476	0,467	0,422		
0,017	0,187	0,223	0,259	0,277	0,223	0,268	0,259	0,250	0,205		
0,022	0,133	0,151	0,178	0,187	0,151	0,178	0,187	0,187	0,151		
0,027	0,079	0,097	0,142	0,151	0,106	0,142	0,151	0,133	0,115		
0,032	0,061	0,070	0,097	0,106	0,079	0,097	0,097	0,097	0,088		
0,037	0,043	0,052	0,079	0,088	0,061	0,079	0,070	0,070	0,061		
0,042	0,034	0,043	0,061	0,070	0,043	0,061	0,052	0,043	0,034		

tabmot											
motor											
0	0	1300	-60								
0	1320	-54									
1340	-48										
1360	-42										
1380	-36										
1400	-30										
1420	-24										
1440	-18										
1460	-12										
1480	-6										
1500	0										
1520	6										
1540	12										
1560	18										
1580	24										
1600	30										
1620	36										
1640	42										
1660	48										
1680	54										
1700	60										

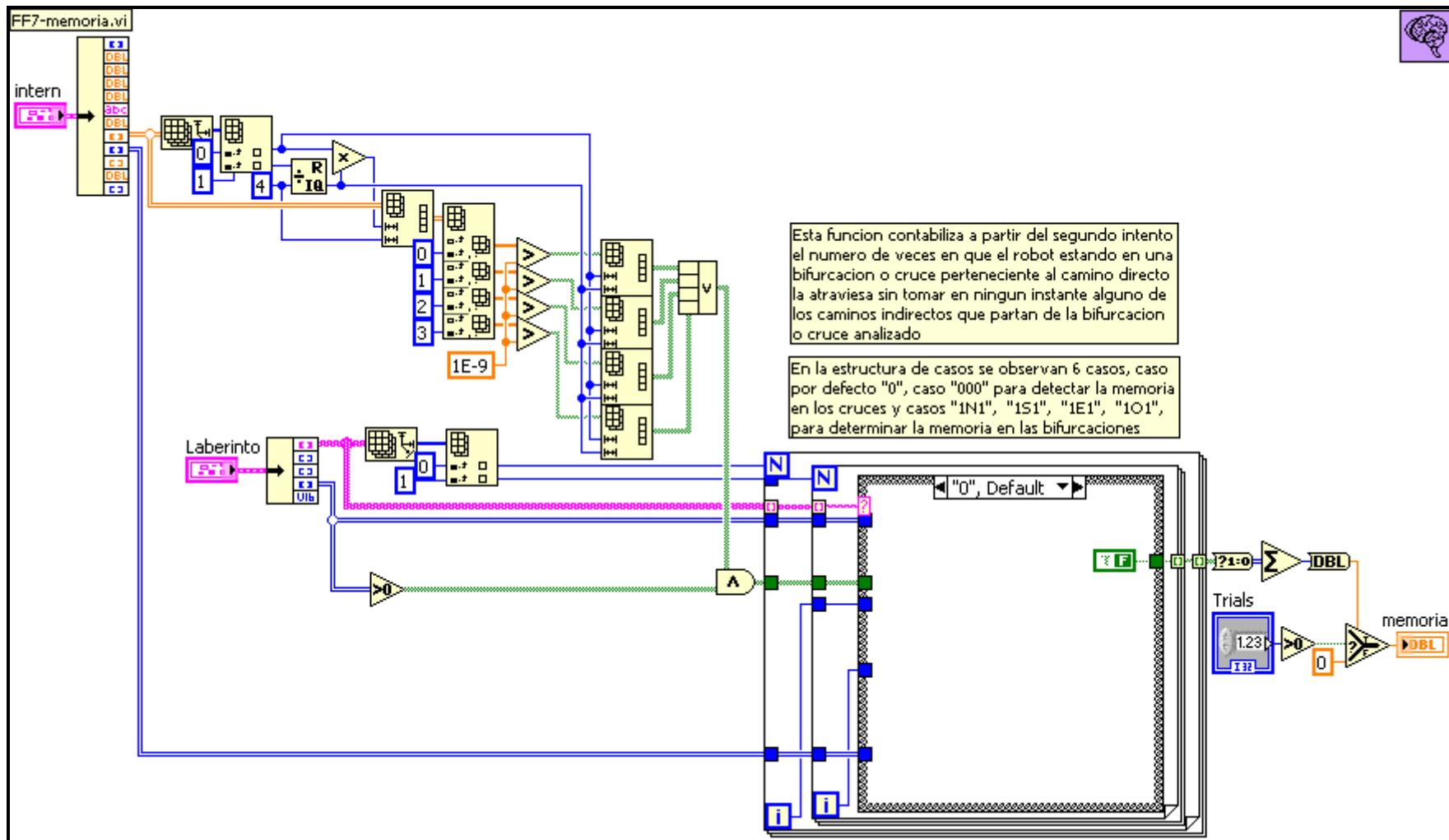
Se emplea la misma tabla para todos los motores

θsd		rsd		esf		rsf		grey/black	
0	1,047197551197	0	0,03	0	0,7853981633974	0	0,01	T	F
	0,5235987755983		0,03		2,356194490192		0,01	T	F
0		0,03		3,926990816987		0,01		T	F
	-0,5235987755983	0,03		5,497787143782		0,01		T	F
	-1,047197551197	0,03							

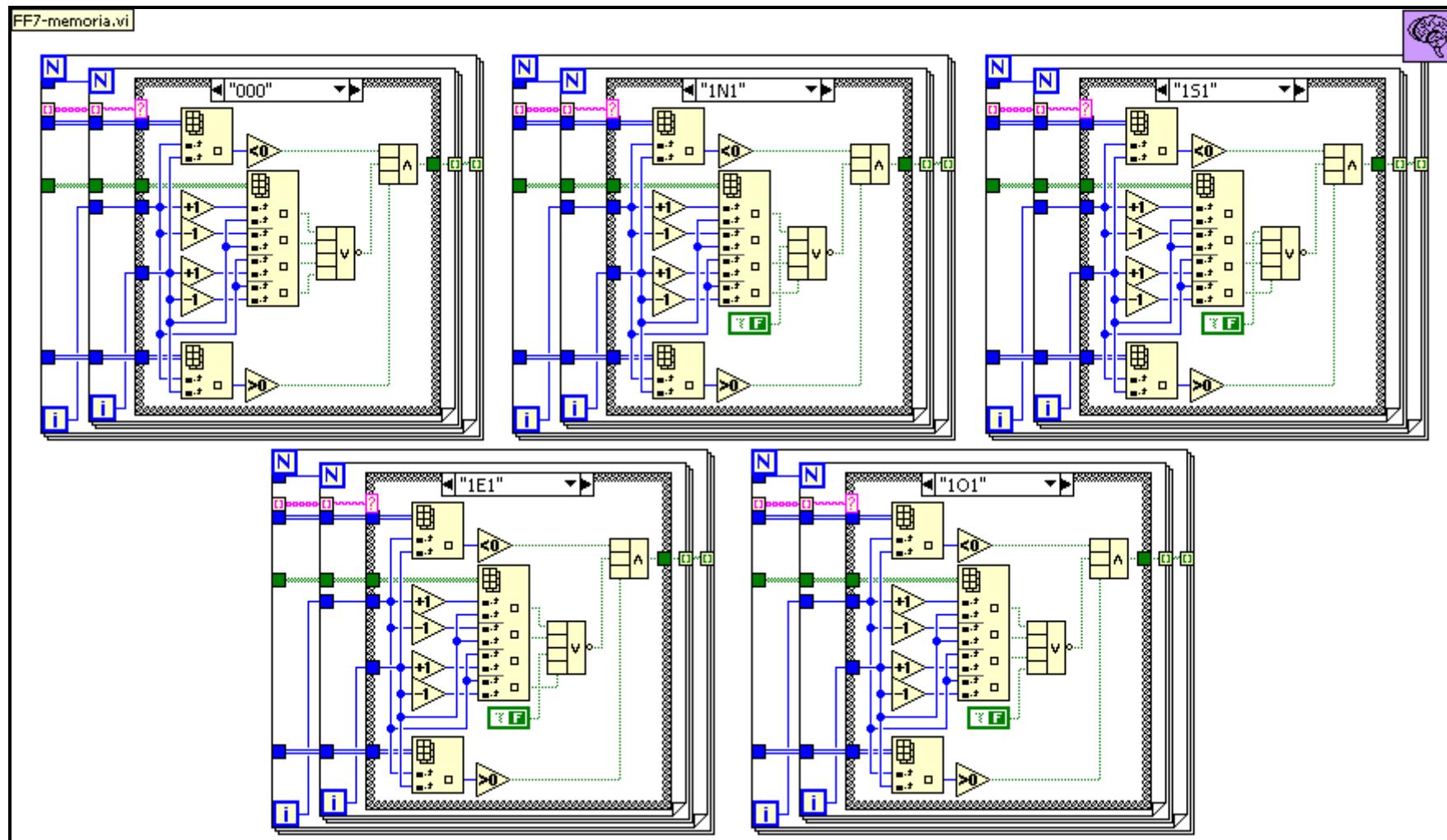
Experiencia 3: Función Fitness – FF7 (Diagrama de Bloques)



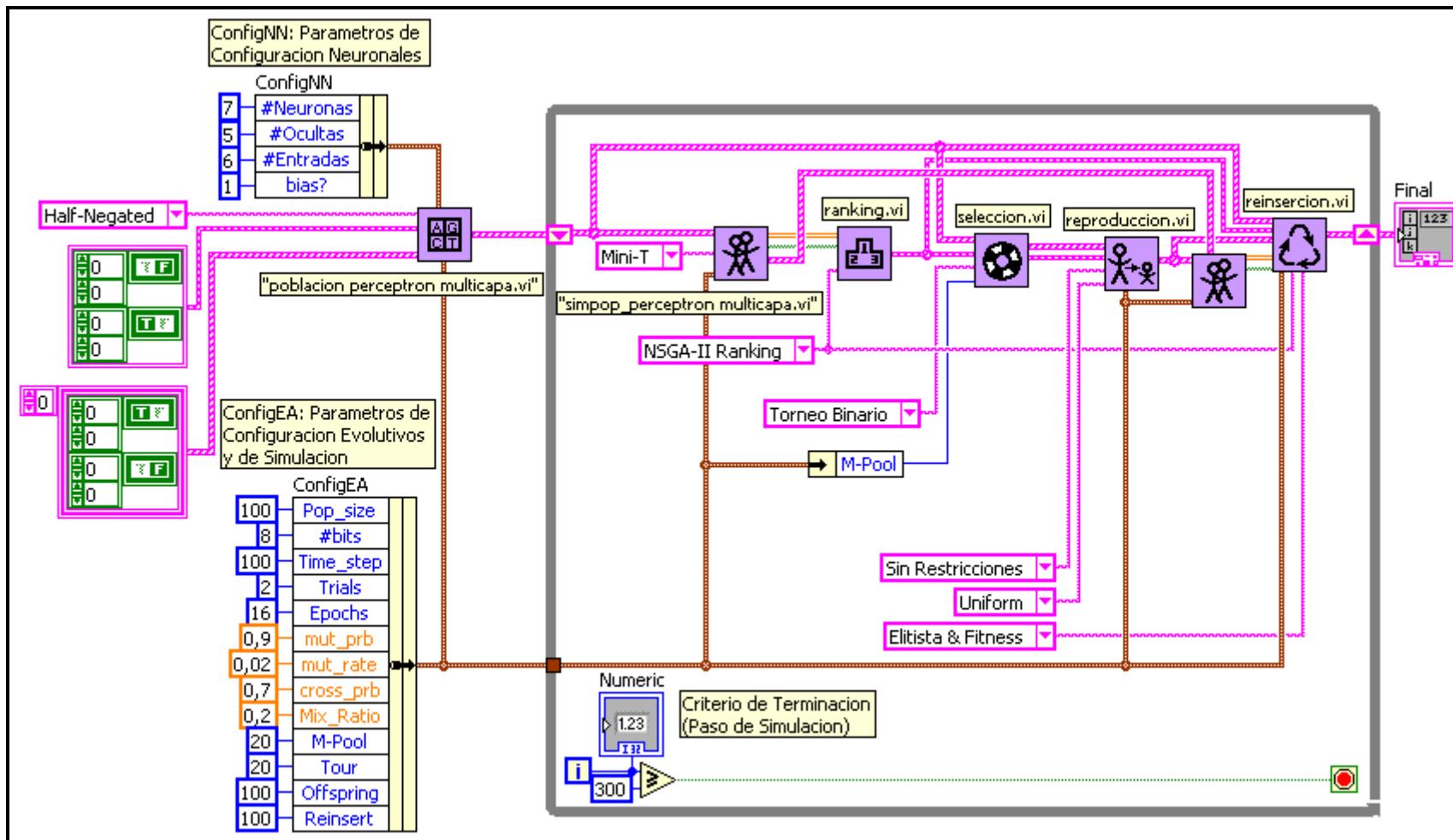
Experiencia 3: Función Fitness – FF7 (subVIs)



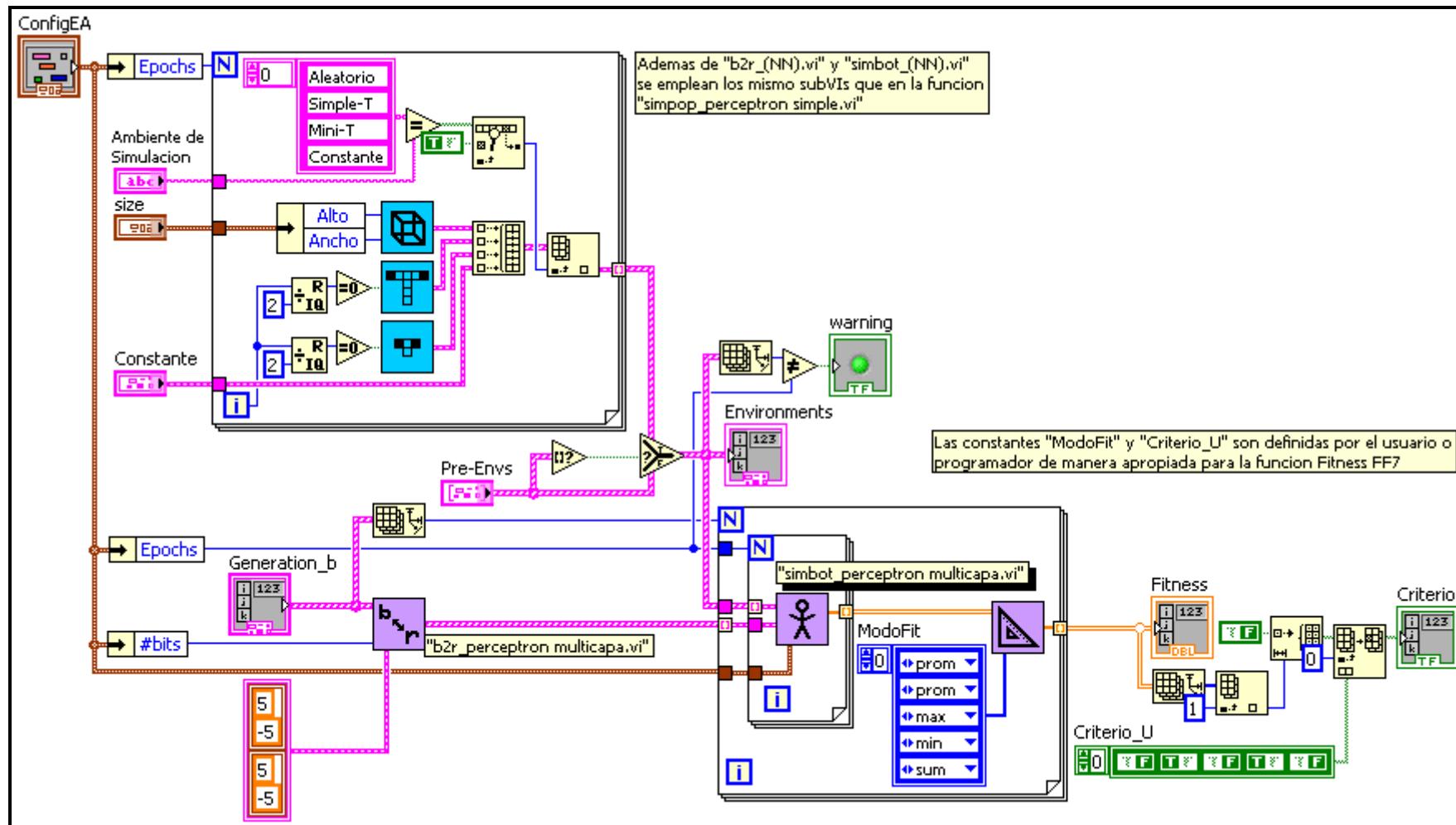
Experiencia 3: Función Fitness – FF7 (subVIs)



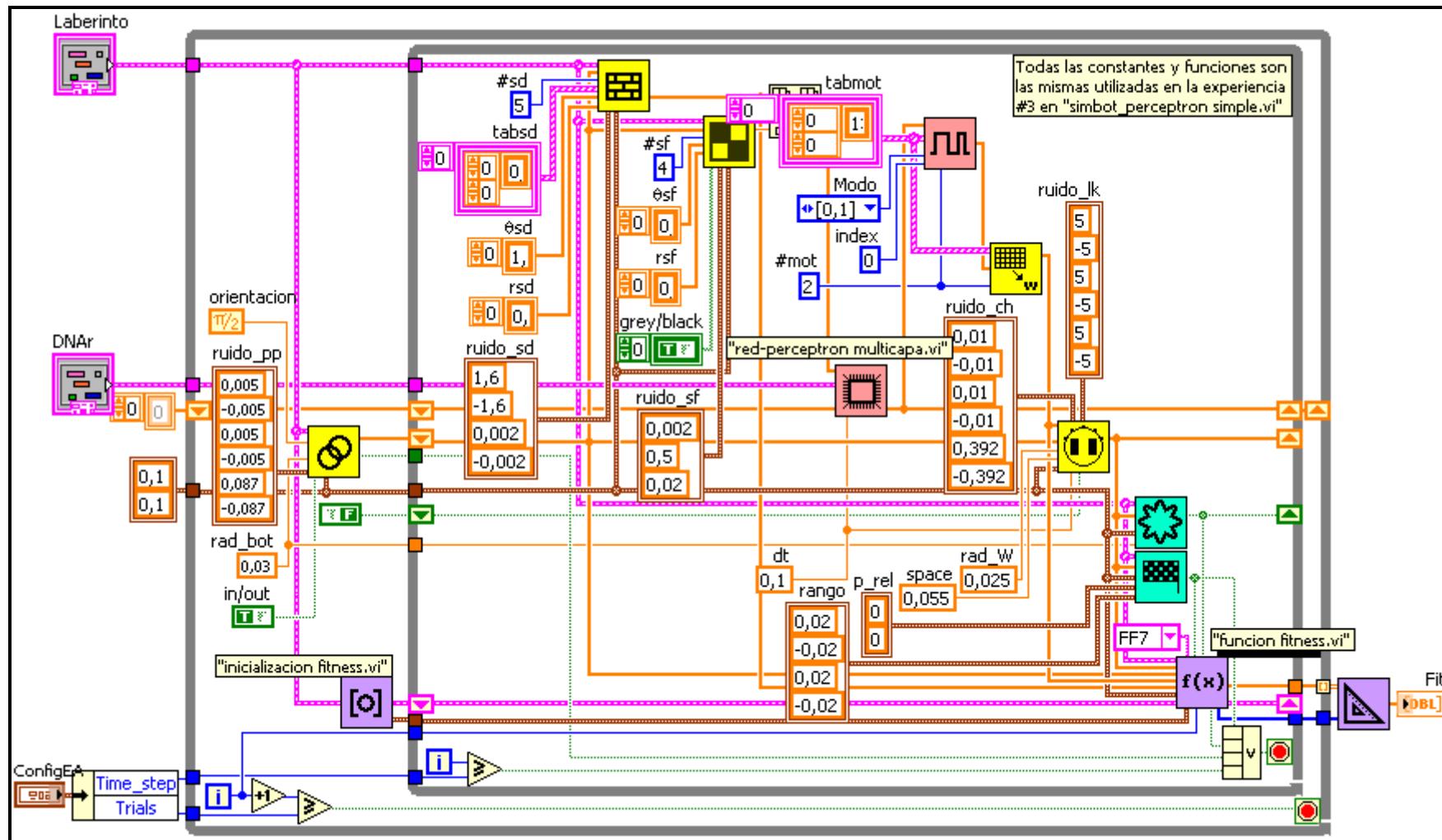
Experiencia 3: Algoritmo Evolutivo – Perceptron Multicapa (Diagrama de Bloques)



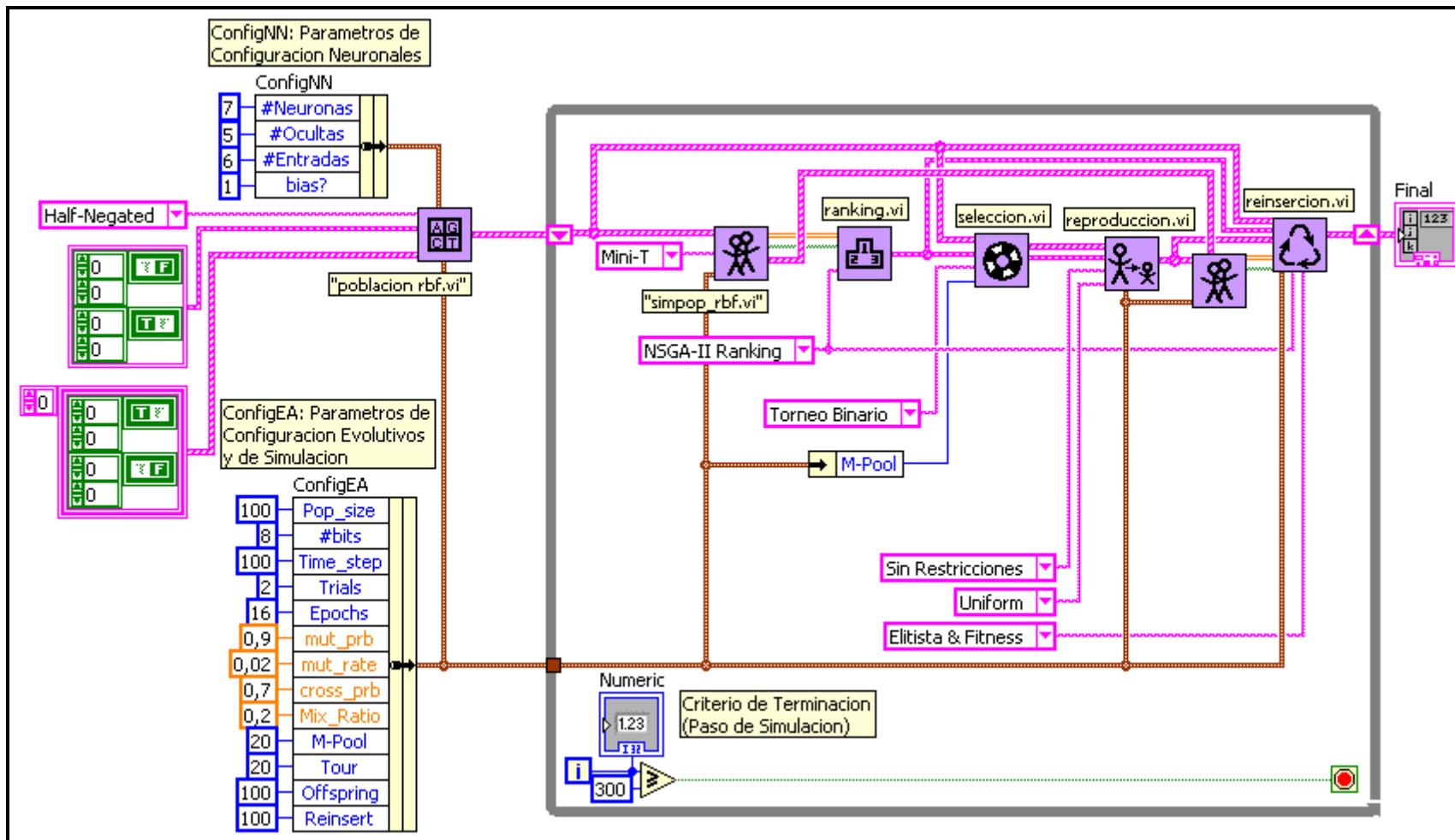
Experiencia 3: Simulación de la Población – Perceptron Multicapa (Diagrama de Bloques)



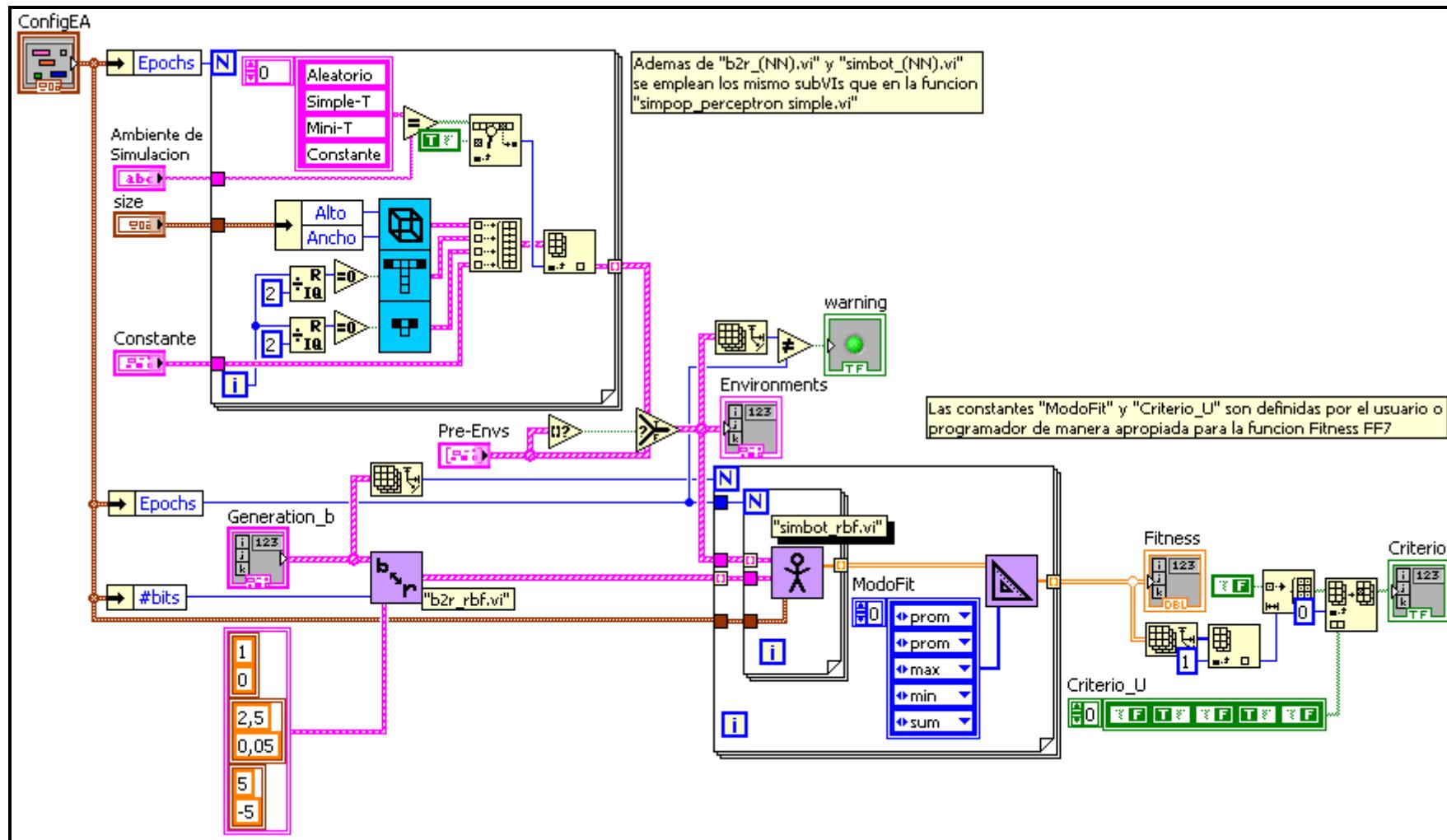
Experiencia 3: Simulación de un Individuo – Perceptron Multicapa (Diagrama de Bloques)



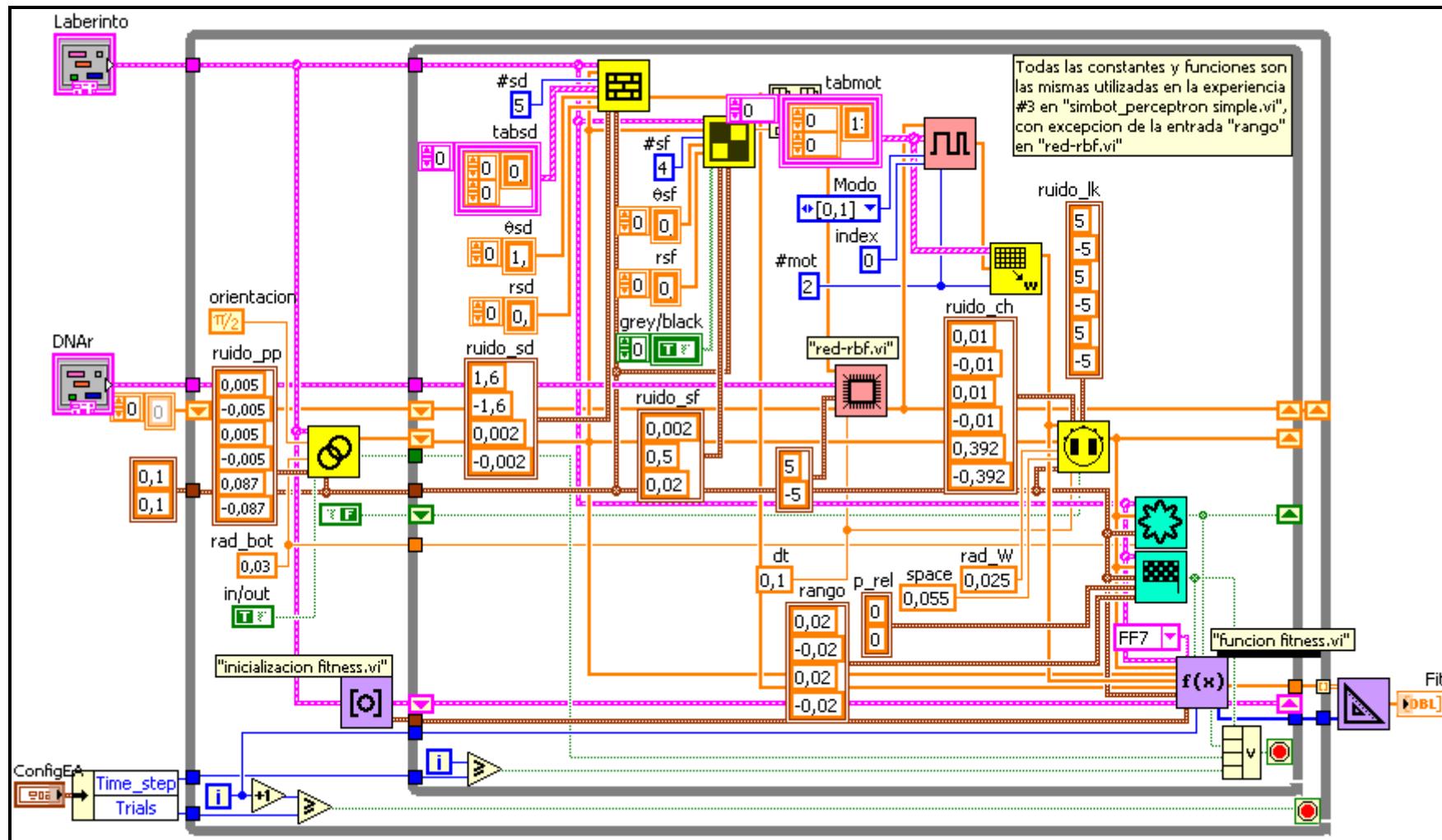
Experiencia 3: Algoritmo Evolutivo – RBF (Diagrama de Bloques)



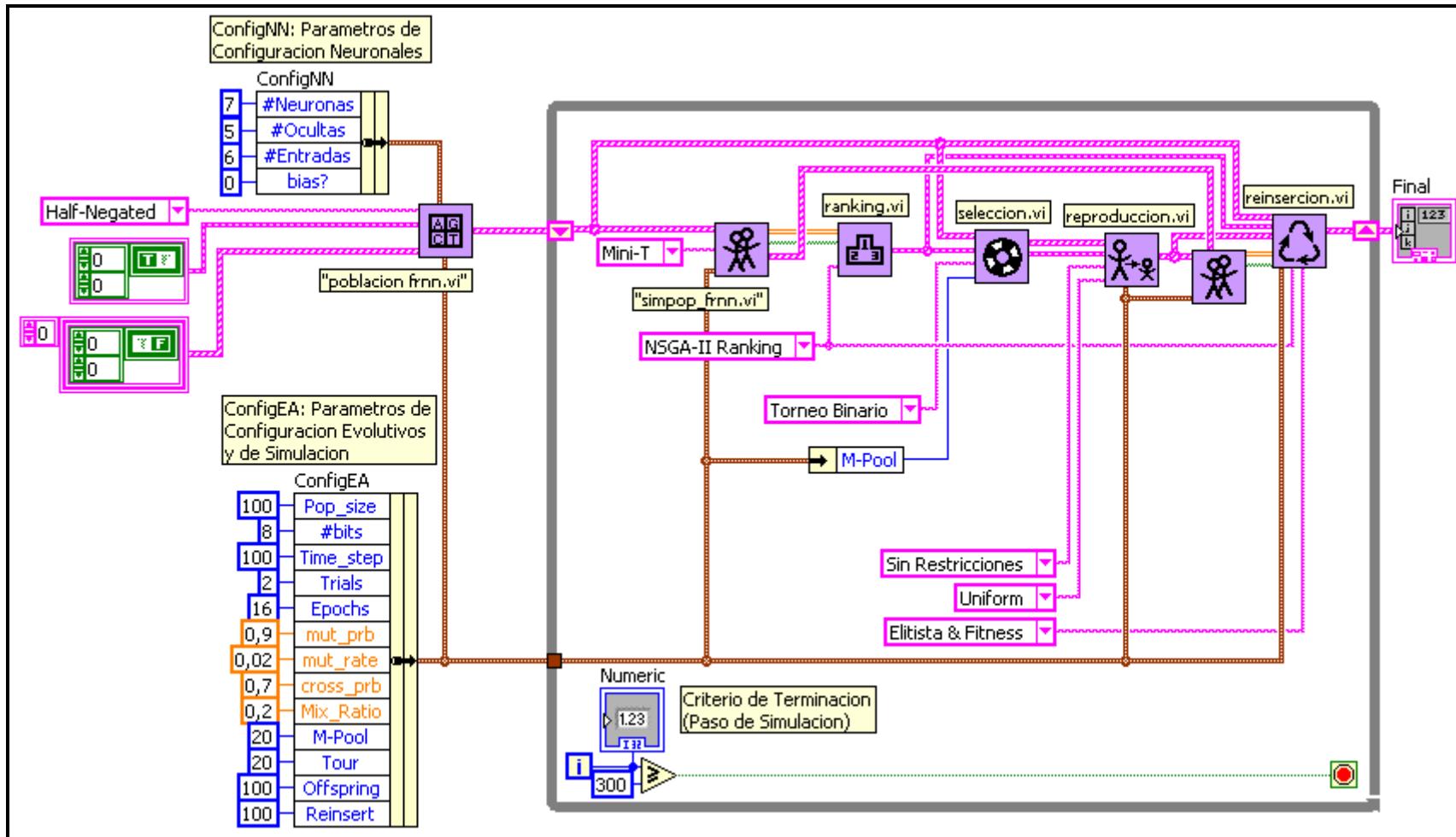
Experiencia 3: Simulación de la Población – RBF (Diagrama de Bloques)



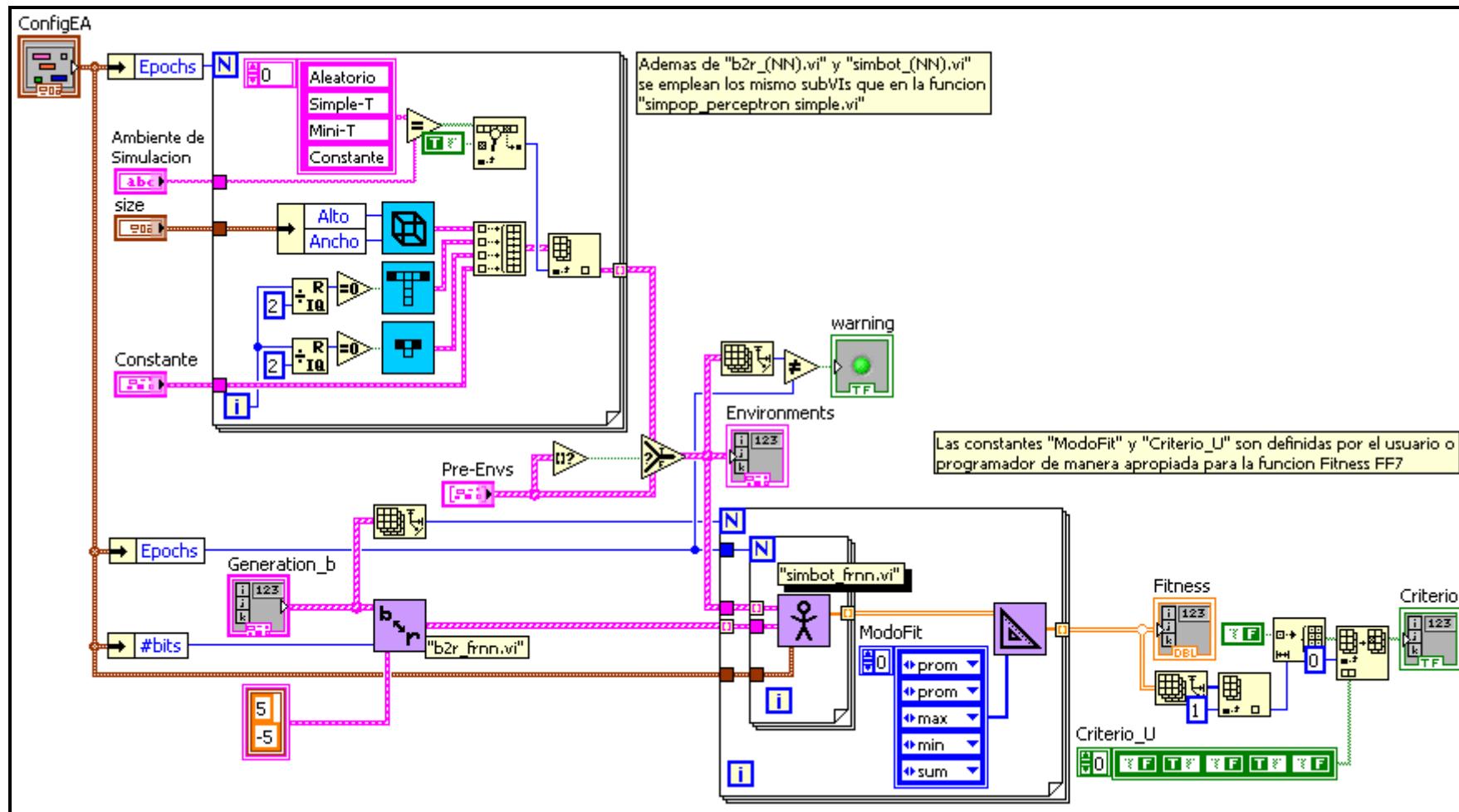
Experiencia 3: Simulación de un Individuo – RBF (Diagrama de Bloques)



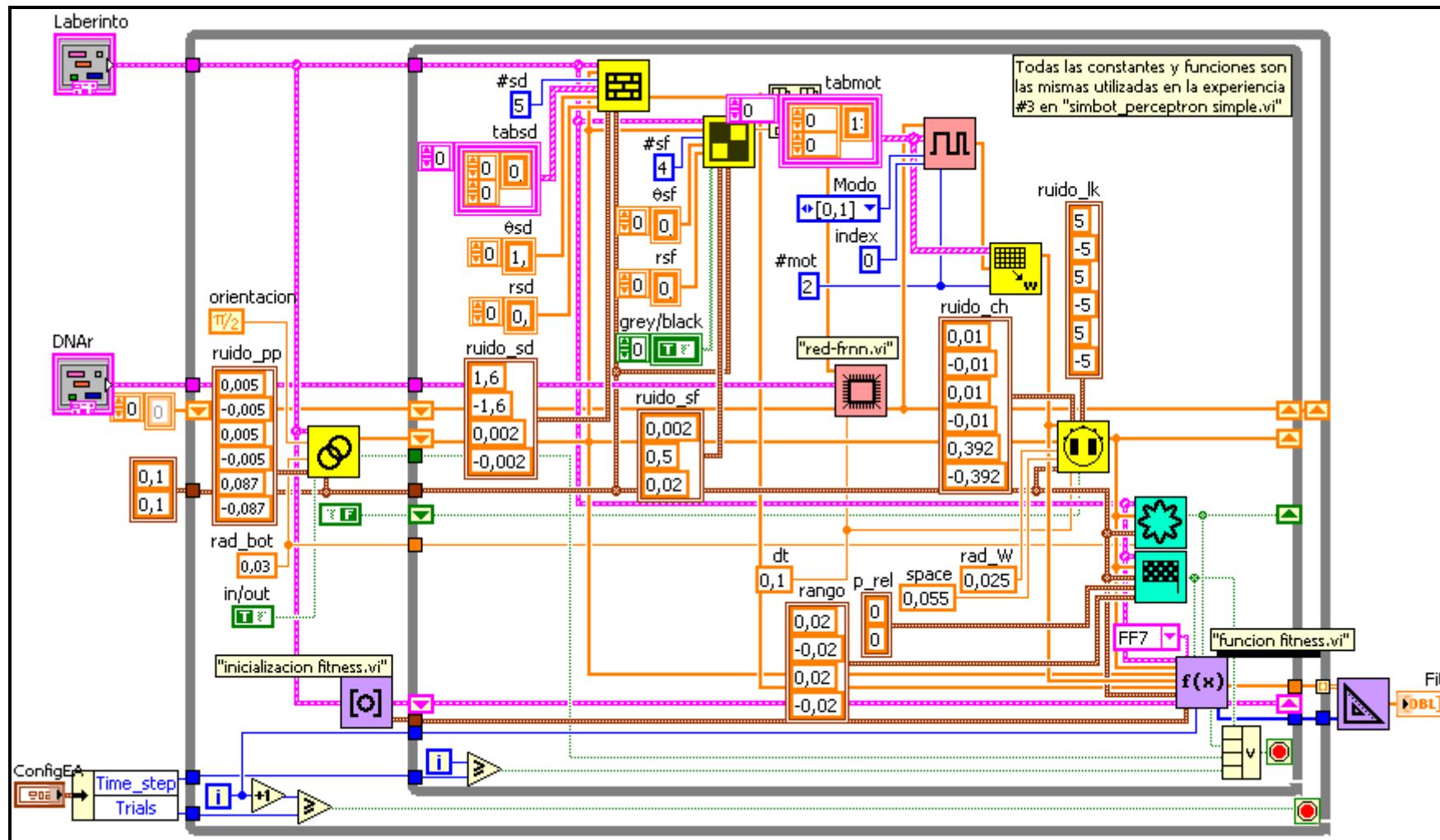
Experiencia 3: Algoritmo Evolutivo – FRNN (Diagrama de Bloques)



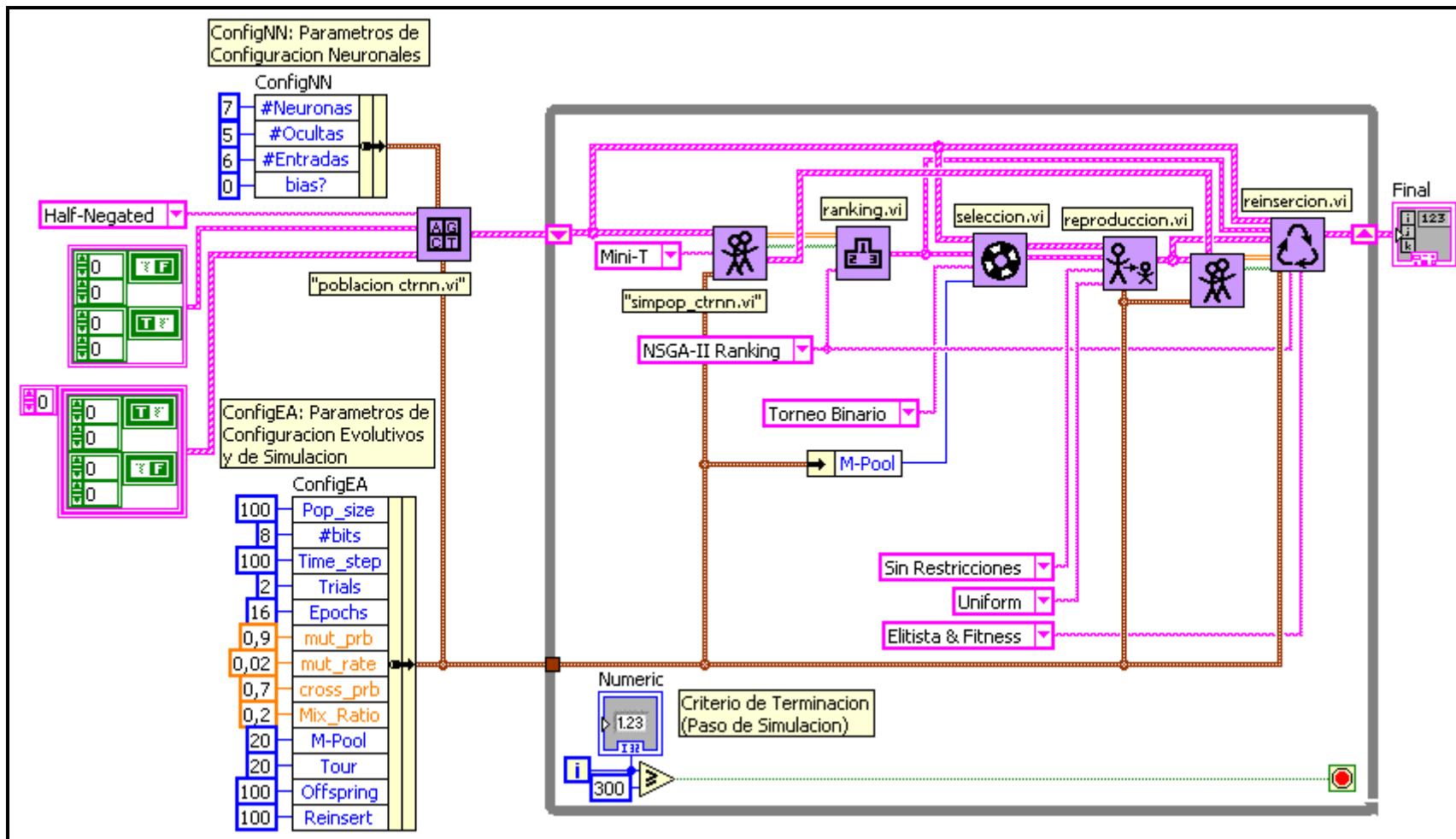
Experiencia 3: Simulación de la Población – FRNN (Diagrama de Bloques)



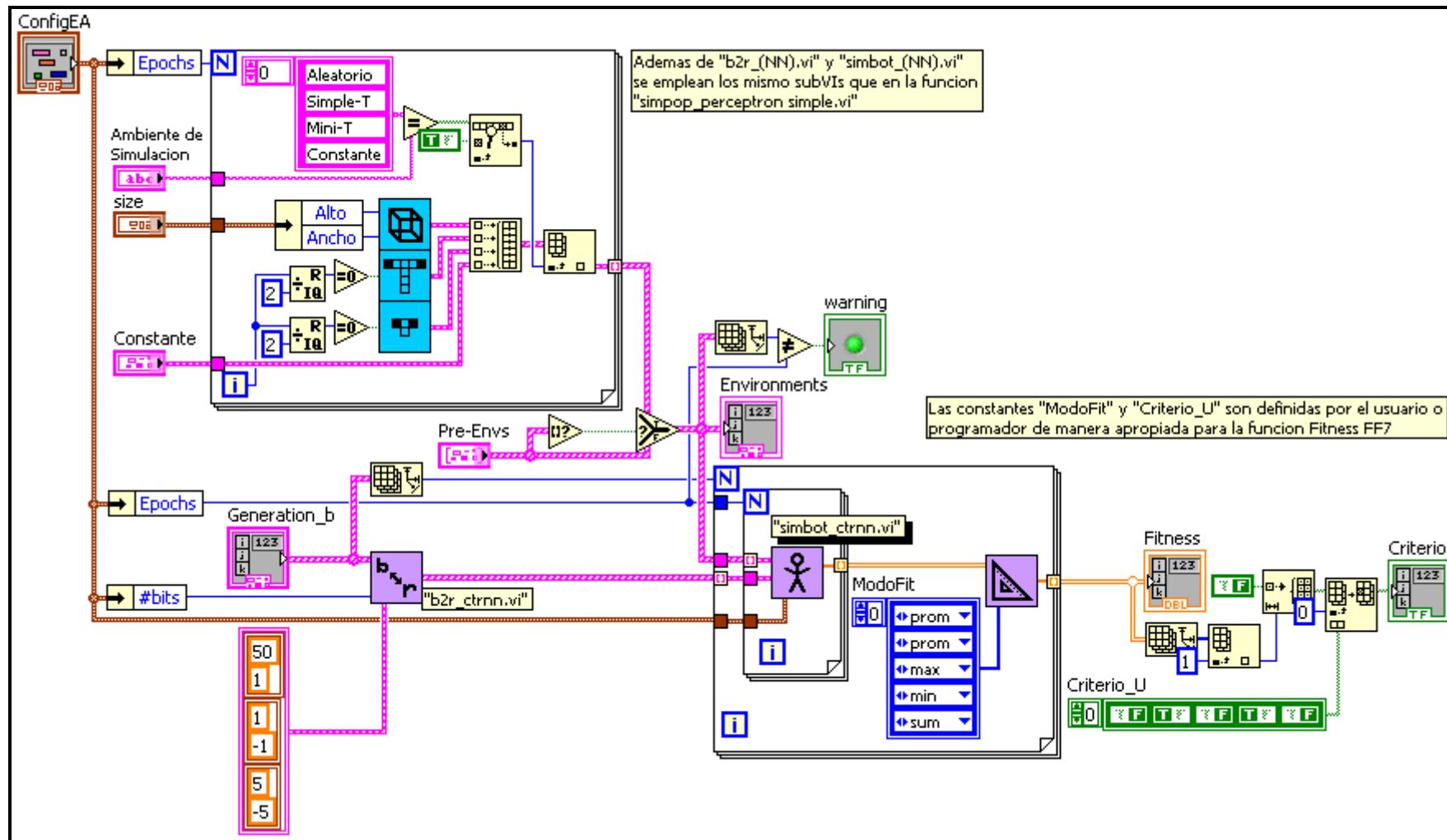
Experiencia 3: Simulación de un Individuo – FRNN (Diagrama de Bloques)



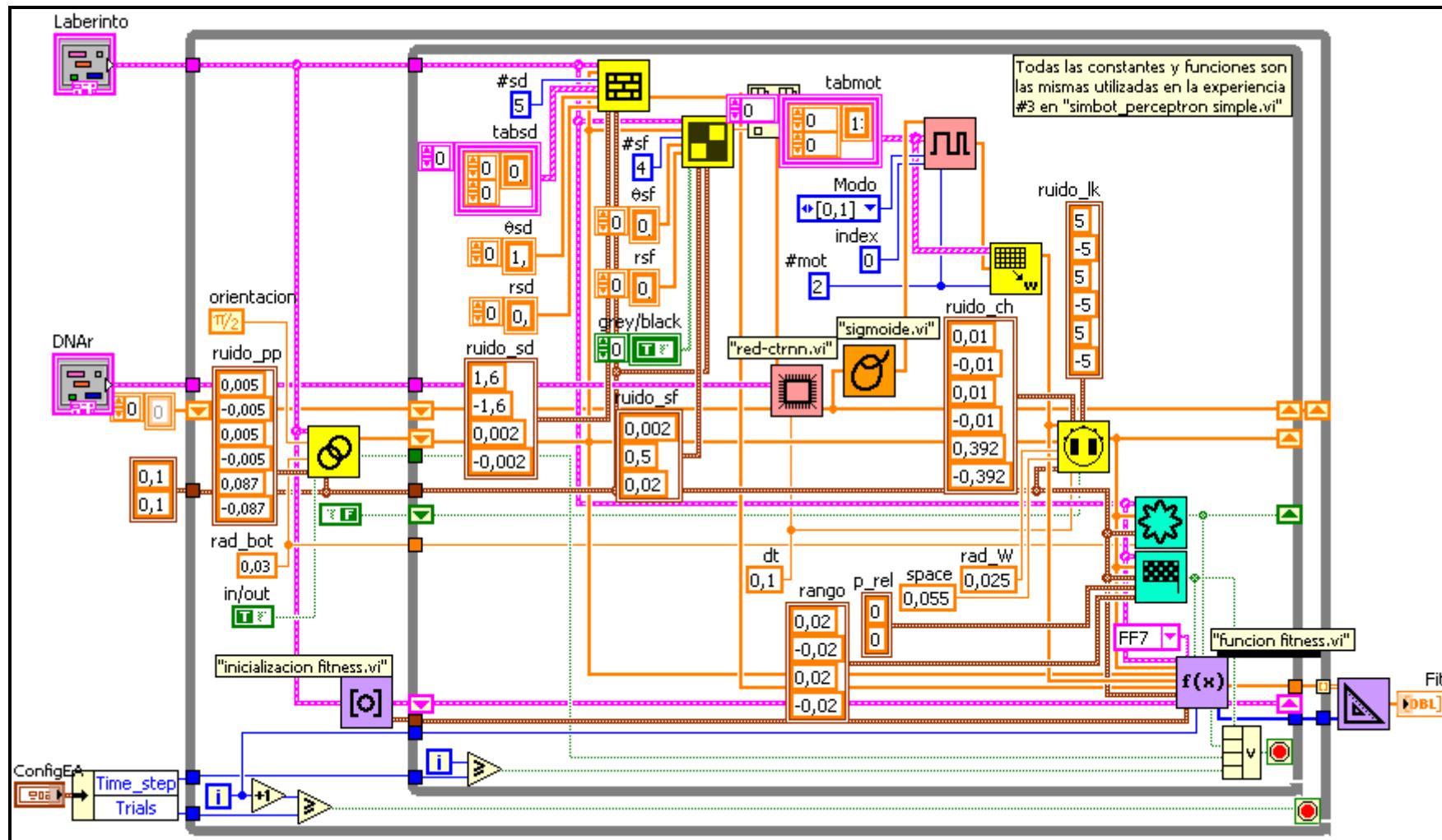
Experiencia 3: Algoritmo Evolutivo – CTRNN (Diagrama de Bloques)



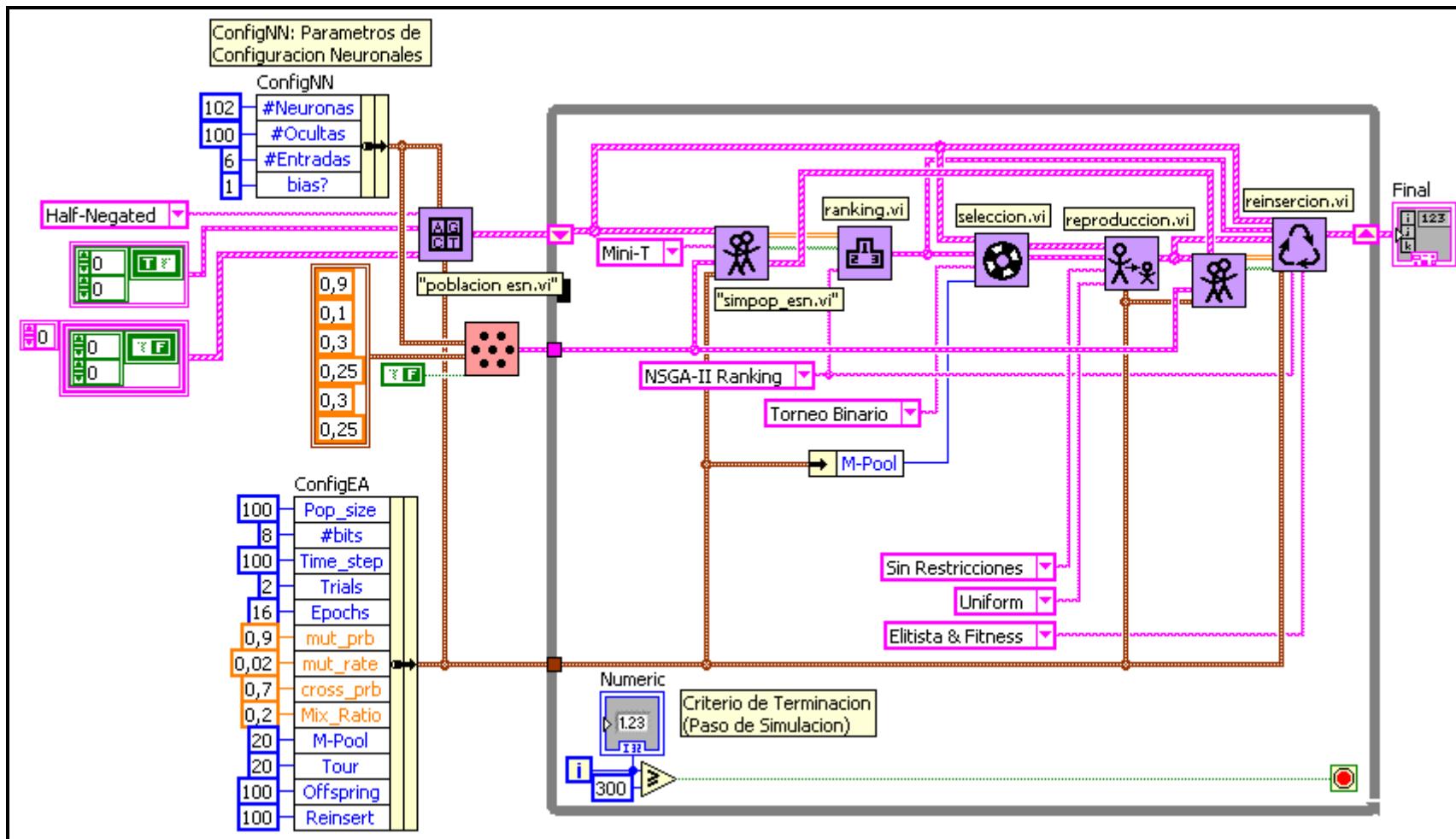
Experiencia 3: Simulación de la Población – CTRNN (Diagrama de Bloques)



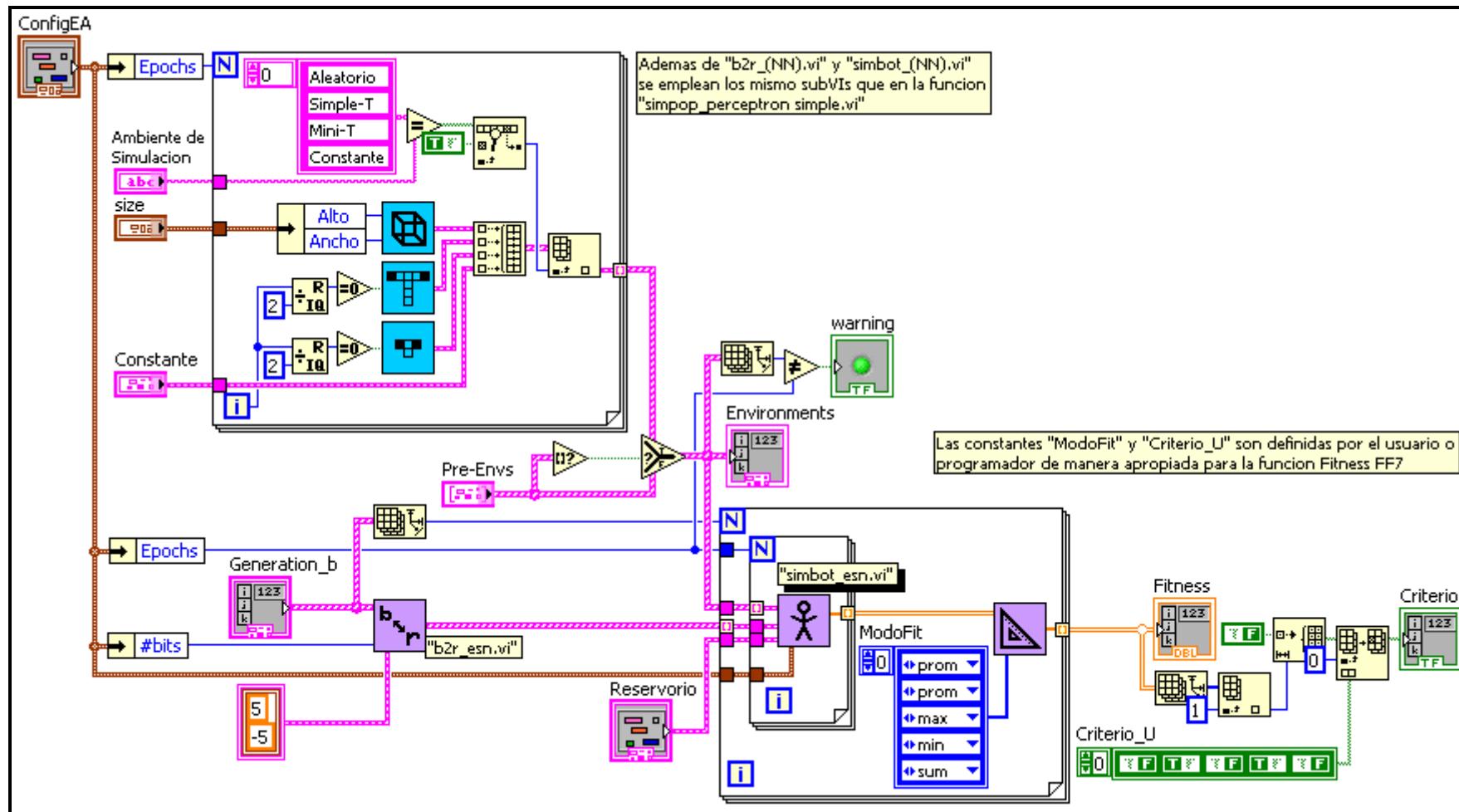
Experiencia 3: Simulación de un Individuo – CTRNN (Diagrama de Bloques)



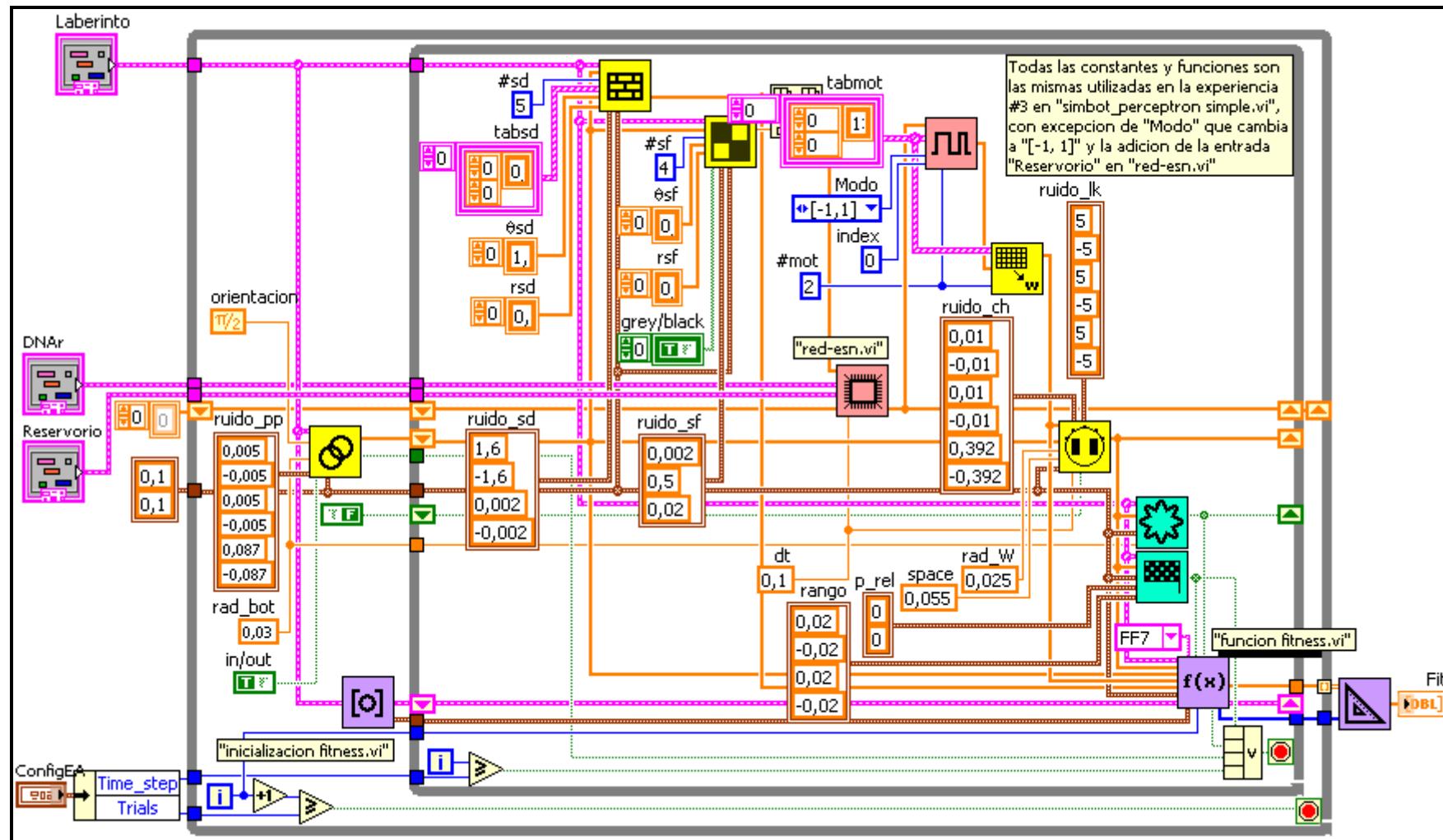
Experiencia 3: Algoritmo Evolutivo – ESN (Diagrama de Bloques)



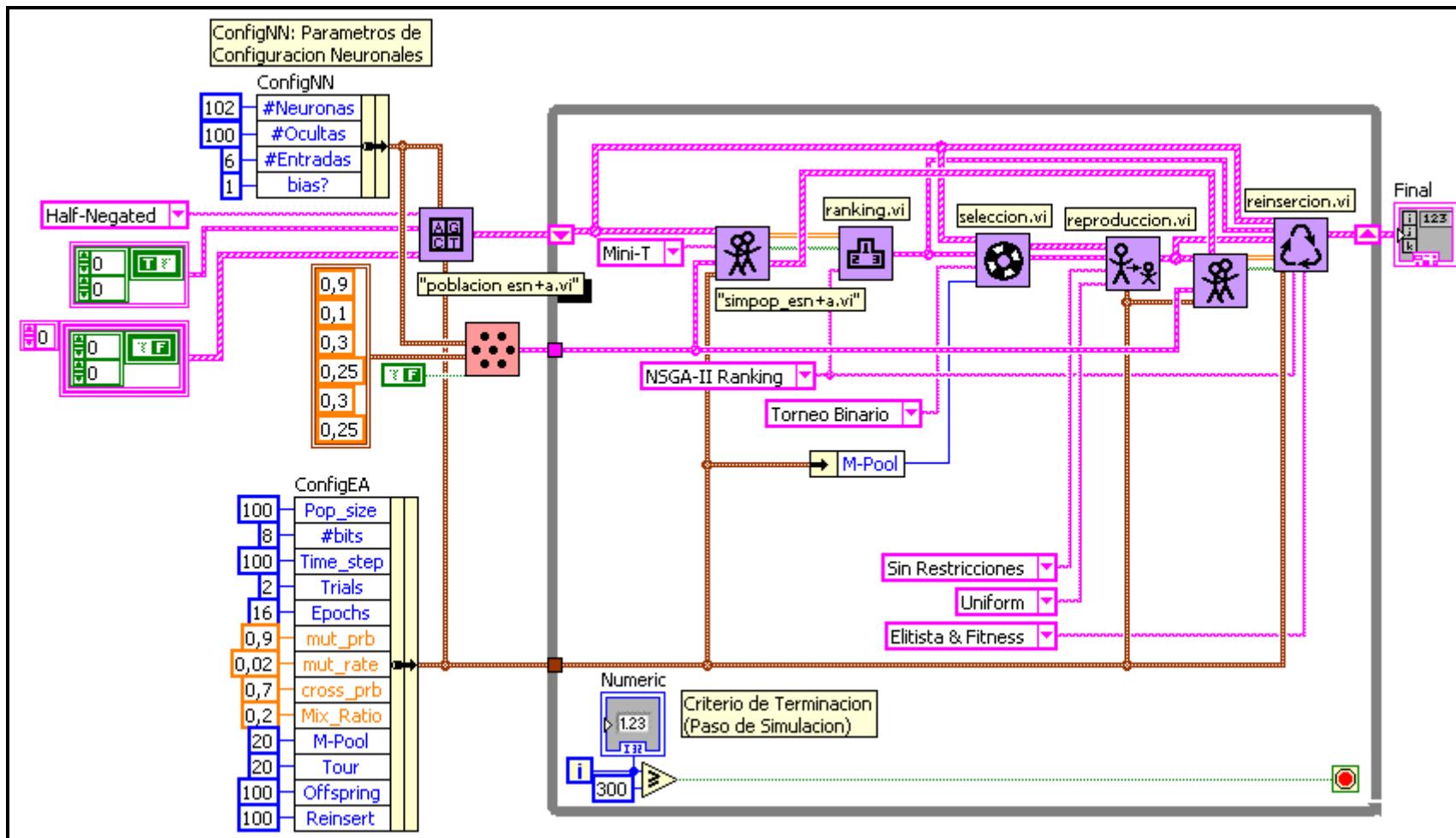
Experiencia 3: Simulación de la Población – ESN (Diagrama de Bloques)



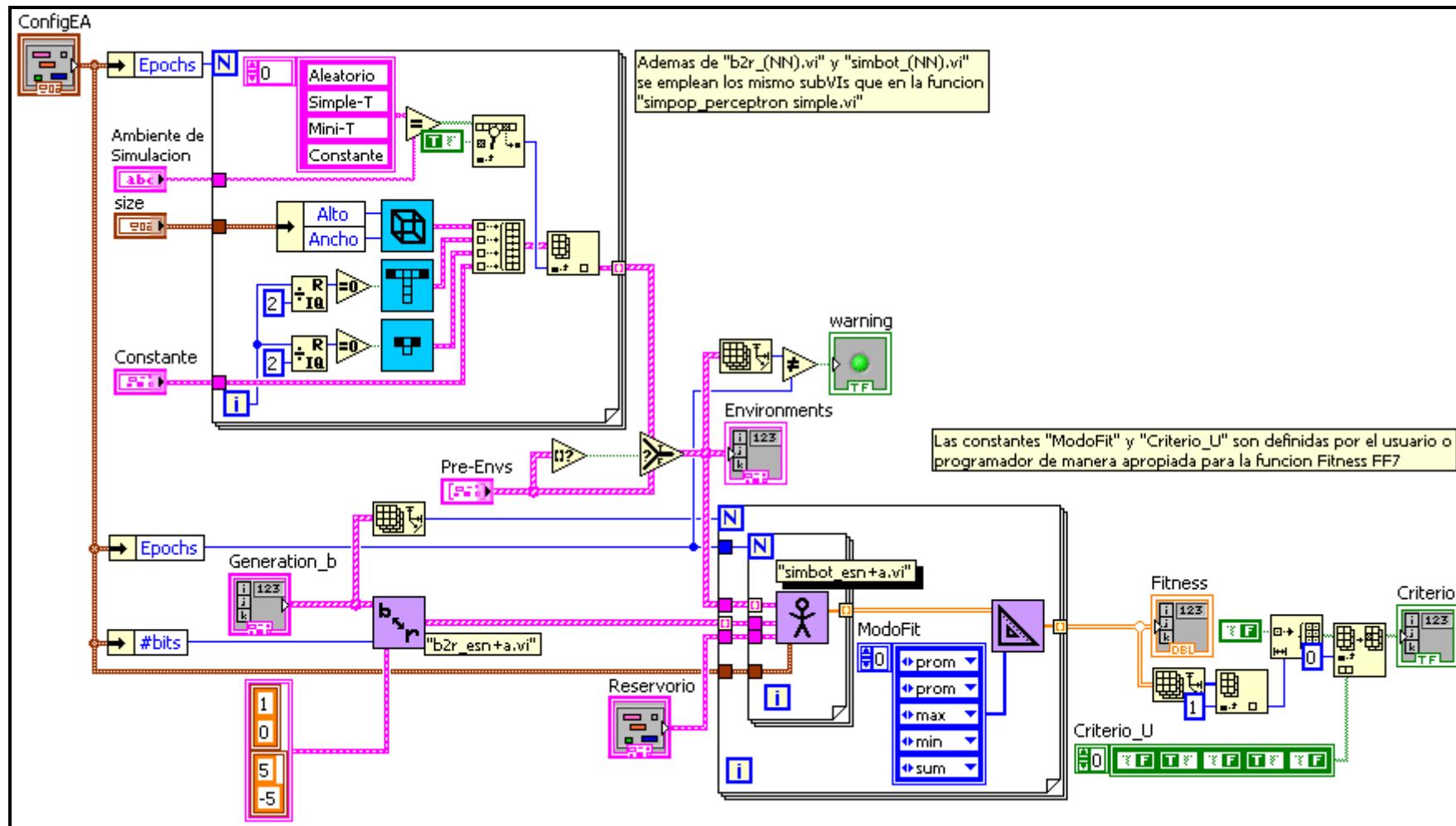
Experiencia 3: Simulación de un Individuo – ESN (Diagrama de Bloques)



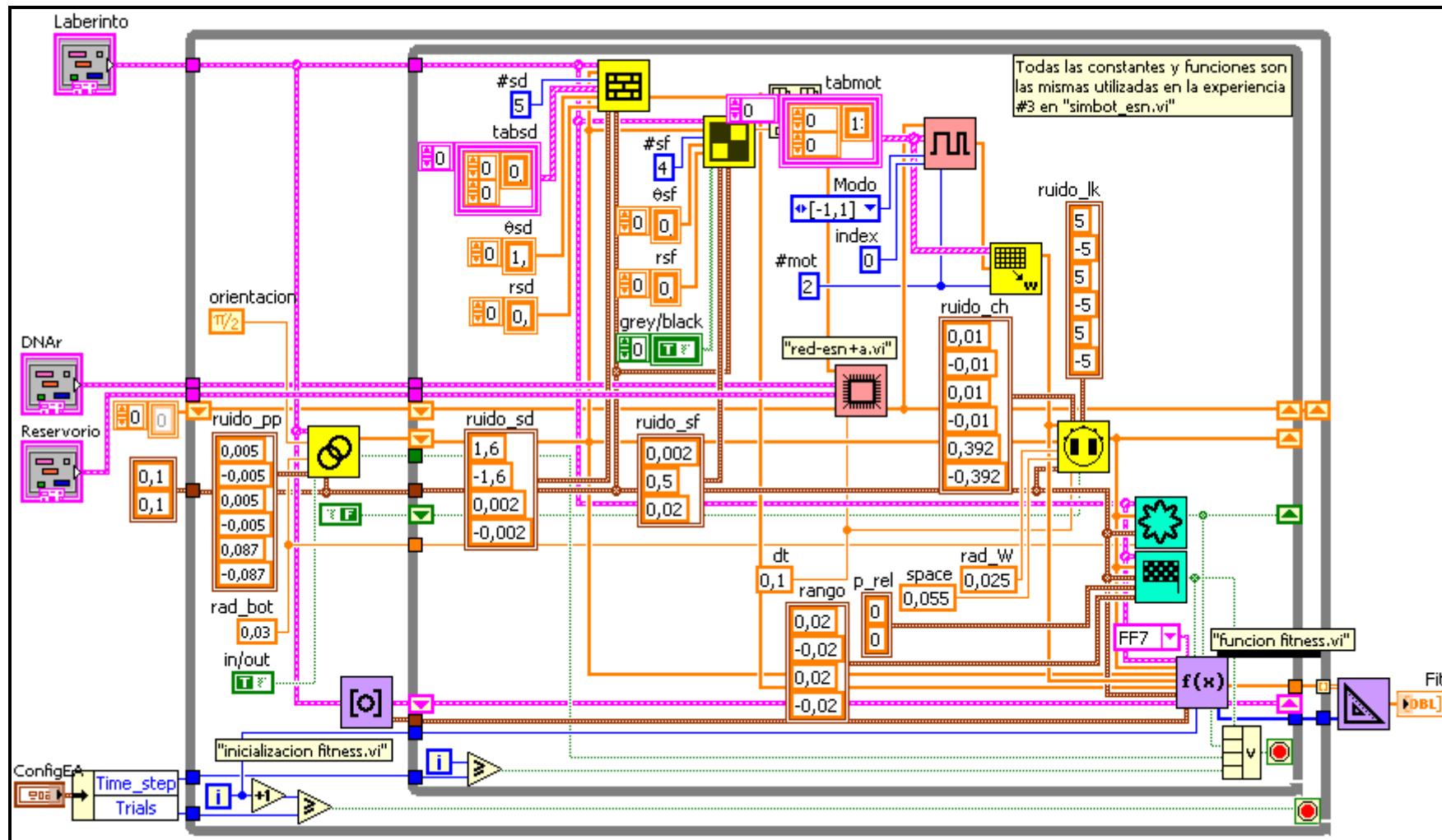
Experiencia 3: Algoritmo Evolutivo – ESN+ α (Diagrama de Bloques)



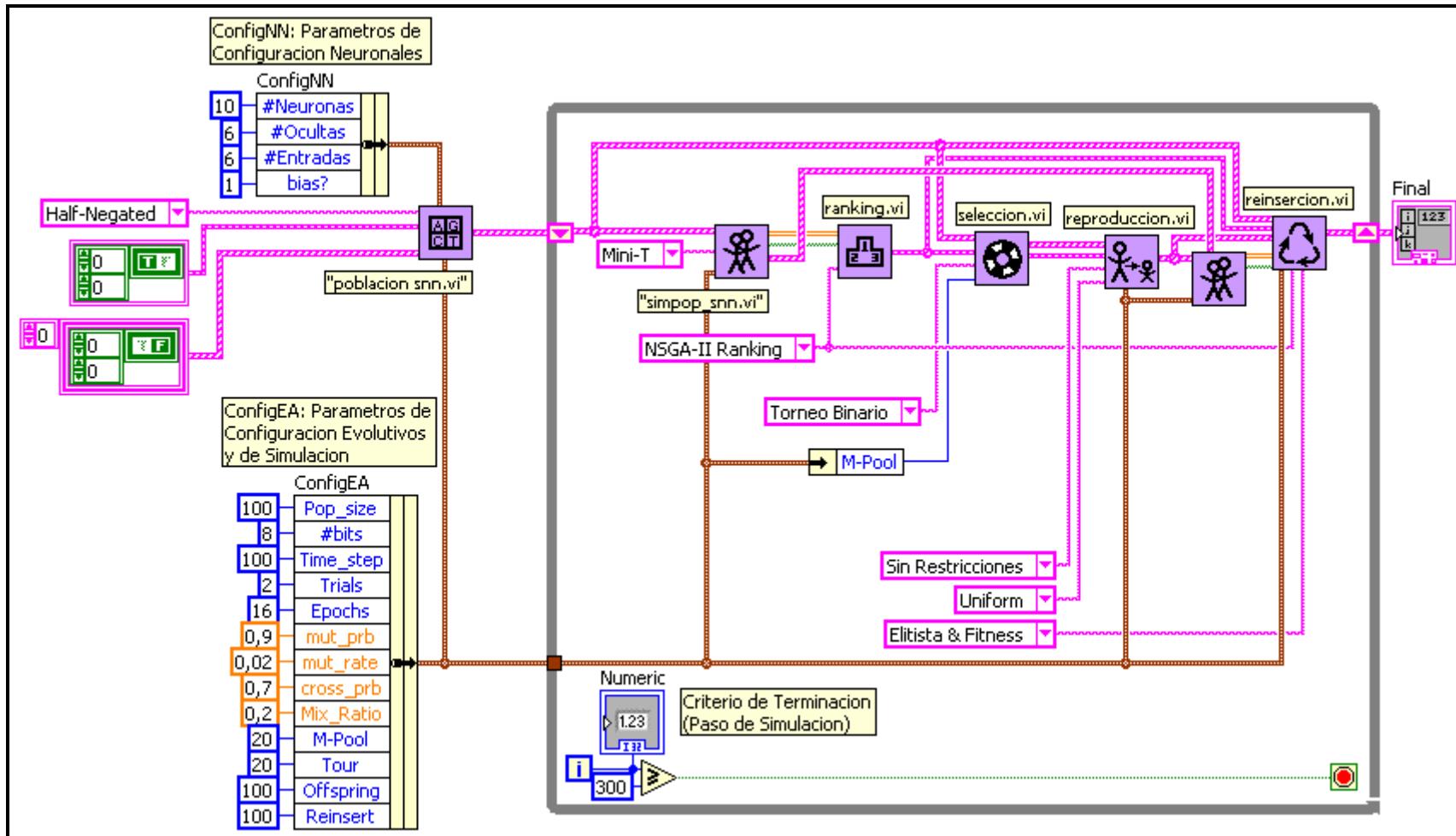
Experiencia 3: Simulación de la Población – ESN+ α (Diagrama de Bloques)



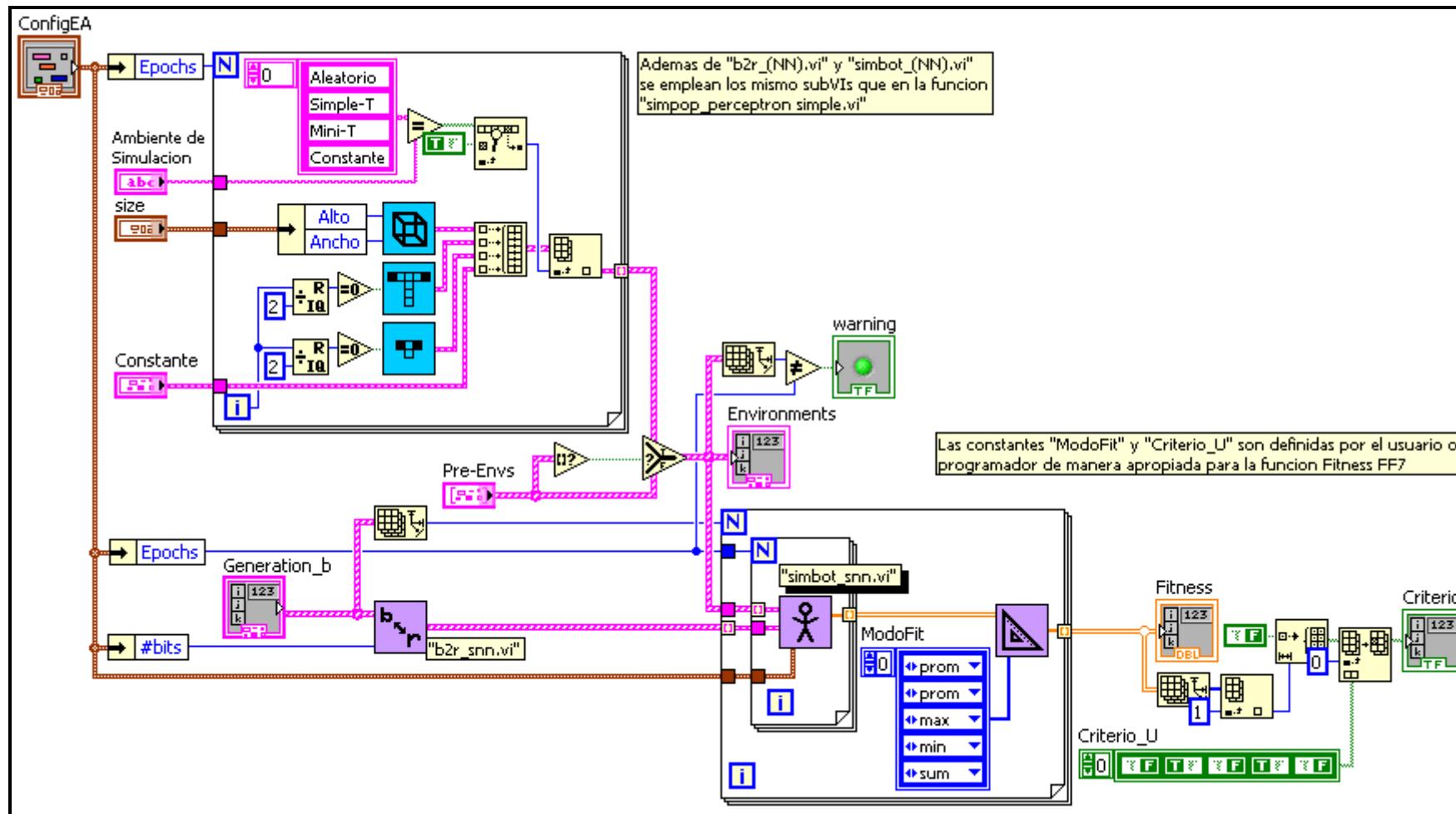
Experiencia 3: Simulación de un Individuo – ESN+ α (Diagrama de Bloques)



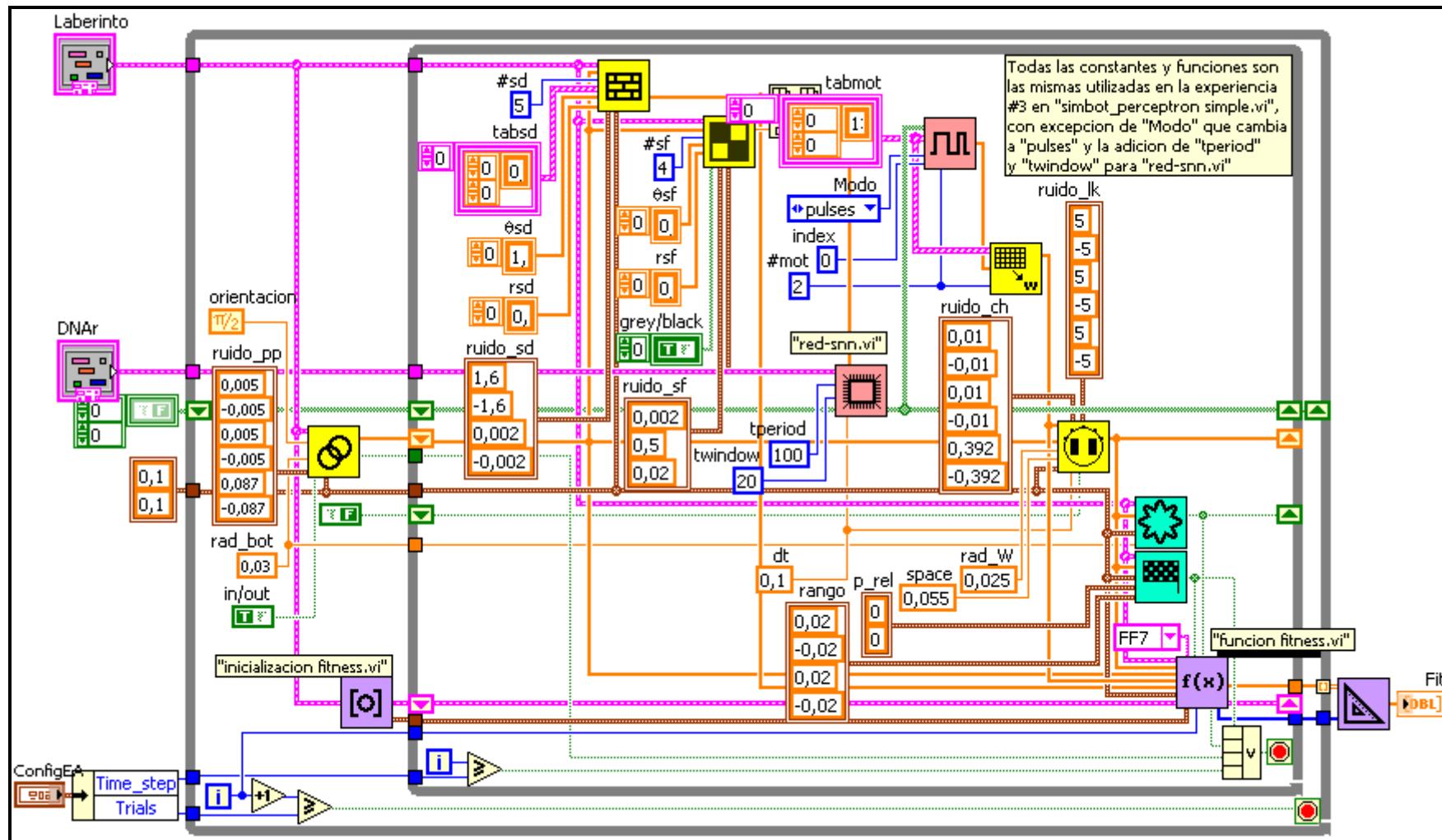
Experiencia 3: Algoritmo Evolutivo – SNN (Diagrama de Bloques)



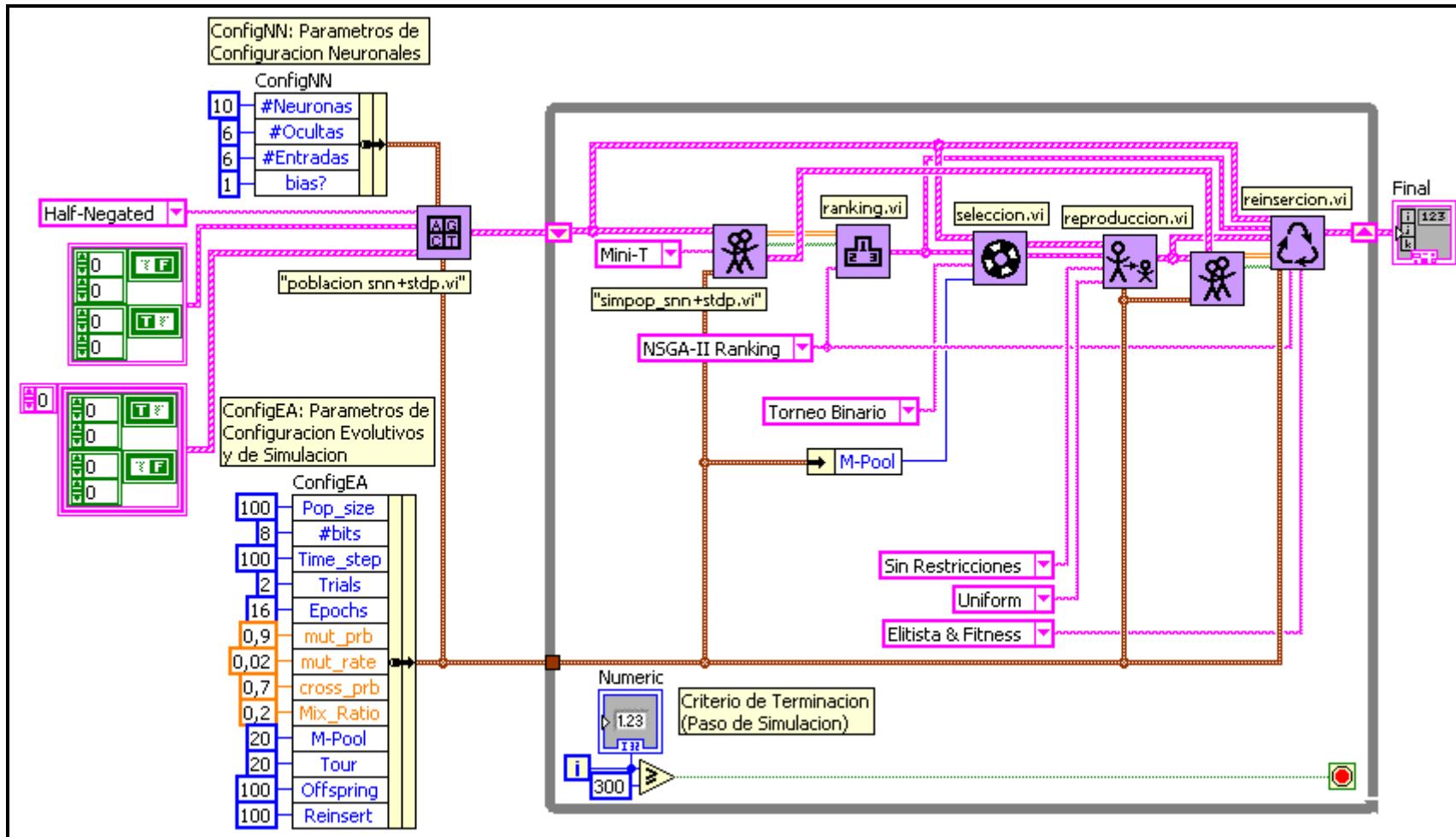
Experiencia 3: Simulación de la Población – SNN (Diagrama de Bloques)



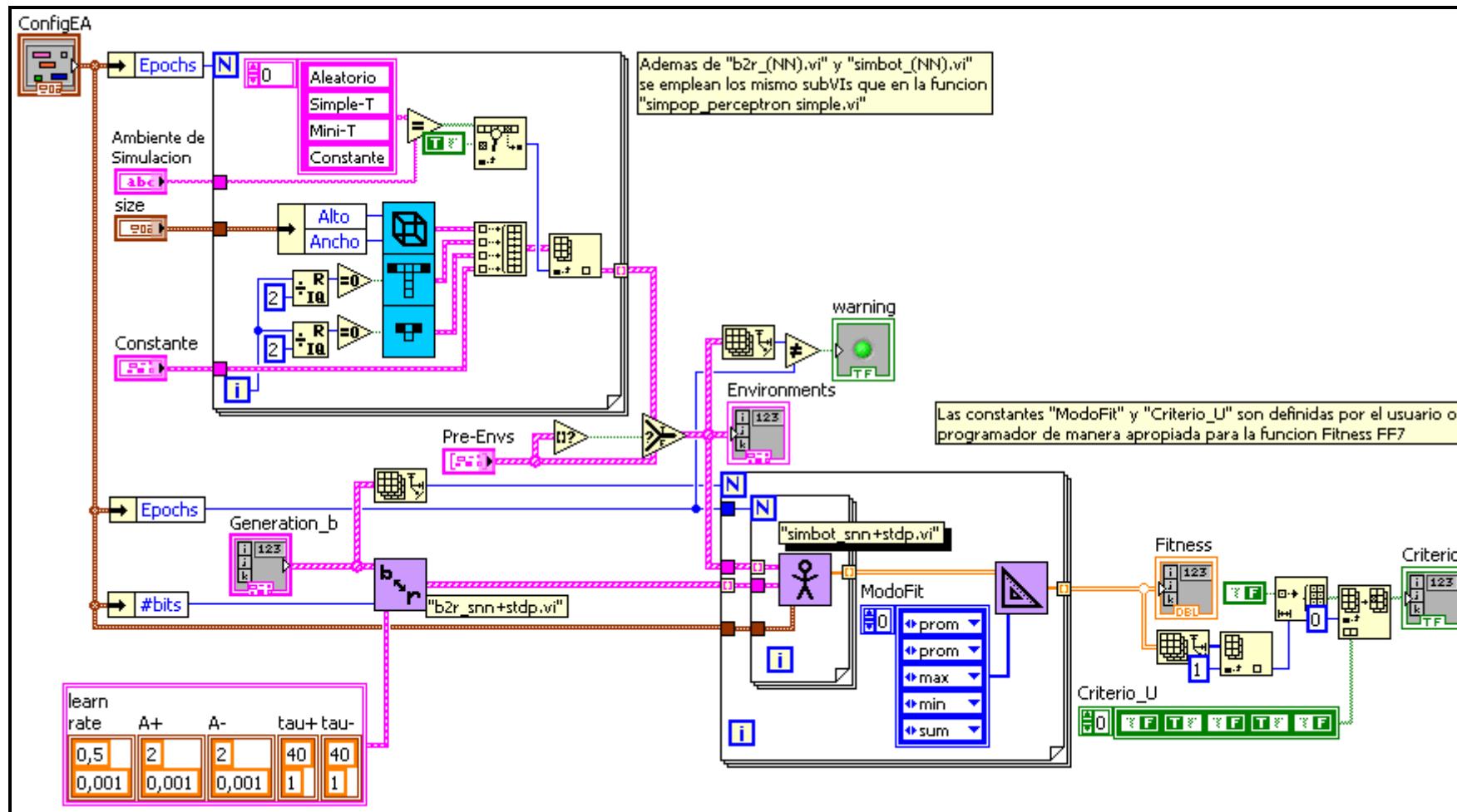
Experiencia 3: Simulación de un Individuo – SNN (Diagrama de Bloques)



Experiencia 3: Algoritmo Evolutivo – SNN+STDP (Diagrama de Bloques)



Experiencia 3: Simulación de la Población – SNN+STDP (Diagrama de Bloques)



Experiencia 3: Simulación de un Individuo – SNN+STDP (Diagrama de Bloques)

