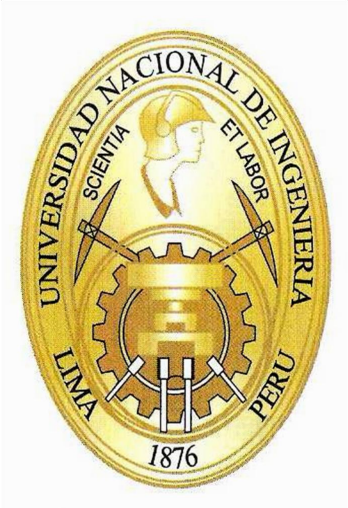
UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA MECÁNICA



PLANEAMIENTO DE MOVIMIENTOS DEL ROBOT EXPLORADOR RE1 USANDO INTELIGENCIA COMPUTACIONAL IMPLEMENTADA EN UN FPGA

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE:
INGENIERO MECATRÓNICO

JOSÉ LUIS RIVERA GUTIÉRREZ

PROMOCIÓN 2008-II

LIMA - PERÚ

2010

Dedico esta tesis a mis padres

José Rivera y Ana Lupe Gutiérrez,

a mi hermana

Fiorella,

a mi abuela

Elba Serina,

a mi asesor

Ricardo Rodríguez Bustinza

y a todas las personas que han contribuido

en mi formación personal y profesional

TABLA DE CONTENIDOS

LISTA DE FIGURAS

LISTA DE CUADROS

PRÓLOGO

Los algoritmos de inteligencia computacional permiten a un robot resolver problemas de manera semejante a los sistemas biológicos, una de cuyas propiedades es la adaptabilidad.

Tomando esta referencia, se busca demostrar en esta tesis que los algoritmos de inteligencia computacional permiten el planeamiento de movimientos de un robot explorador, que será llamado en adelante robot RE1, con lo que este será capaz de generar autónomamente sus pasos, de manera que su velocidad de desplazamiento sea mejorada en cada intento.

Para alcanzar este objetivo, se han desarrollado dos algoritmos para la generación de pasos que han sido implementados por separado. El primero se denomina "algoritmo para generar posiciones". En este algoritmo, el robot RE1 se traslada usando una secuencia de posiciones que ha sido aprendida en una red neuronal. La secuencia es continuamente mejorada mediante ensayos de prueba y error que consisten en realizar pequeños cambios a la red y evaluar la performance resultante. El segundo algoritmo se ha denominado "algoritmo para generar desplazamientos". En este algoritmo, el robot RE1 planea sus movimientos basándose en una función, implementada en una red neuronal, que relaciona la variación de su posición respecto a un sistema de referencia externo con la posición de un eslabón suyo respecto a otro, es decir, relaciona la variación de su desplazamiento con la variación de la posición de sus eslabones. Con esta función, el robot RE1 puede trasladarse al realizar una búsqueda, mediante algoritmos genéticos,

de la posición siguiente de las articulaciones que maximice el desplazamiento de él. Hay que resaltar que ambos algoritmos se caracterizan por ser variables ya que se van adaptando en cada iteración o ciclo de operación al entorno hasta optimizarse. Además, ambos algoritmos han sido implementados en el dispositivo FPGA (arreglo programable de compuertas lógicas) debido a su versatilidad para realizar operaciones en forma simultánea y a su velocidad de procesamiento.

Por otro lado, esta tesis tiene el propósito de exponer dos algoritmos en tiempo real que, por un lado, permiten a un robot explorador articulado generar autónomamente pasos y que, por el otro lado, han sido implementados en un controlador digital dispuesto en el mismo robot. Además, estos algoritmos también tienen la intención de contribuir con investigaciones en la Universidad Nacional de Ingeniería en temas de inteligencia computacional y ramas afines.

De esta forma, la tesis será desarrollada en capítulos y a continuación se describen brevemente cada uno de ellos.

En el **Capítulo 1**, se dan a conocer las investigaciones o trabajos semejantes realizados anteriormente, la justificación, el planteamiento del problema, los objetivos tanto el general como los específicos, los alcances y las limitaciones de la investigación.

En el **Capítulo 2**, se desarrollan dos algoritmos para el planeamiento del movimiento del robot: el "algoritmo para generar posiciones" y el "algoritmo para generar desplazamientos". Igualmente, se exponen los fundamentos en los cuales están basados.

En el **Capítulo 3**, se describen los algoritmos de funcionamiento de los sensores y los actuadores. Además, se detalla la función que cumplen en el robot RE1, el principio de funcionamiento y los correspondientes diagramas de flujo de cada sensor y actuador.

En el **Capítulo 4**, se especifican las operaciones complementarias para el desarrollo de los algoritmos de planeamiento de movimientos. Estas operaciones

tienen en cuenta el funcionamiento y las restricciones del FPGA SPARTAN-3E.

En el **Capítulo 5**, se presentan las simulaciones y los resultados obtenidos para cada algoritmo de planeamiento de movimientos.

Finalmente, se expresan las conclusiones, la bibliografía, los apéndices, con información específica del desarrollo de la tesis, y los anexos, con detalles técnicos de los sensores y actuadores implementados en el robot RE1.

CAPÍTULO 1
INTRODUCCIÓN

1.1. ANTECEDENTES

Mediante el planeamiento de movimientos, los robots exploradores logran autónomamente generar pasos y así conseguir trasladarse. Ejemplos de estos robots son los trabajos realizados por Clune [1], Bongard [2], Hornby [3], Zhang [4], Gallagher [5] y Wettergreen [6].

El primer trabajo es el de Clune [1], en el cual se desarrolla una codificación generativa llamada HyperNEAT que evoluciona redes neuronales, de modo que estas puedan generar pasos para un robot cuadrúpedo.

En este trabajo, el robot utilizado ha sido desarrollado en el simulador físico ODE[1] como se observa en la figura 1.1. El robot tiene un torso rectangular y cuatro extremidades, cada una con tres cilindros y tres articulaciones tipo bisagra. El primer cilindro funciona como un hueso de cadera, el segundo como un muslo y el último como una pierna. Respecto a las extremidades, estas tienen tres articulaciones: dos en la cadera y una en la rodilla. Una articulación de la cadera (HipFB) permite a las extremidades girar 180° hacia adelante y hacia atrás, y la otra (HipIO) les permite girar hacia dentro y hacia fuera. La articulación de la rodilla rota hacia delante y hacia atrás. Tanto la articulación HipIO como la de la rodilla no tienen restricciones de movimiento [1].

[1]Open Dynamics Engine

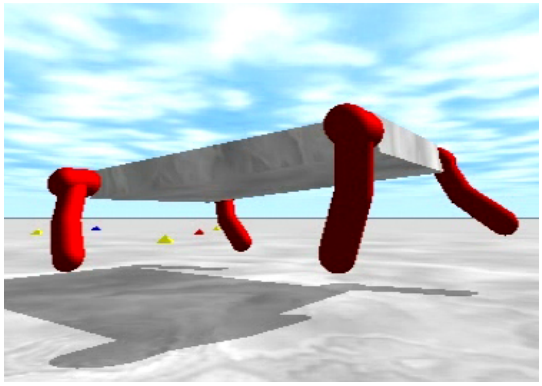


Figura 1.1: Robot cuadrúpedo simulado [1].

Respecto al planeamiento de movimientos, una red neuronal no recurrente y configurada por substratos fue usada. Esta configuración por substratos consiste de tres cuadrículas cartesianas de 5x4 que representan a las capas de entrada, oculta y de salida de la red neuronal (véase figura 1.2). Por otro lado, las entradas de la red neuronal son los ángulos de las doce articulaciones del robot, los cuatro sensores de contacto que devuelven "1" si la pierna está en contacto con el suelo o "0" en caso contrario, el cabeceo, la guiñada y el alabeo del torso y una onda sinusoidal modificada πsen(120°) que facilitó la generación de comportamientos periódicos. Asimismo, las salidas de la red neuronal son los ángulos deseados para cada articulación, los cuales son ingresados a un controlador PID que simula un servomotor [1].

Para el entrenamiento de la red neuronal, la codificación generativa HyperNEAT evoluciona redes neuronales bajo el principio del algoritmo de "neuroevolución de topologías aumentativas" (NEAT[2]). Para esto, la HyperNEAT transforma "redes de producción de patrones composicionales" (CPPNs[3]), que a su vez pueden generar redes neuronales. En este caso, las entradas a una CPPN son una ganancia constante y las ubicaciones en un cuadrícula cartesiana del nodo origen (x_1, y_1) y del nodo destino (x_2, y_2). Entonces, la CPPN evalúa estas cinco

[2]NeuroEvolution of Augmenting Topologies.
[3]Compositional Pattern-Producing Networks

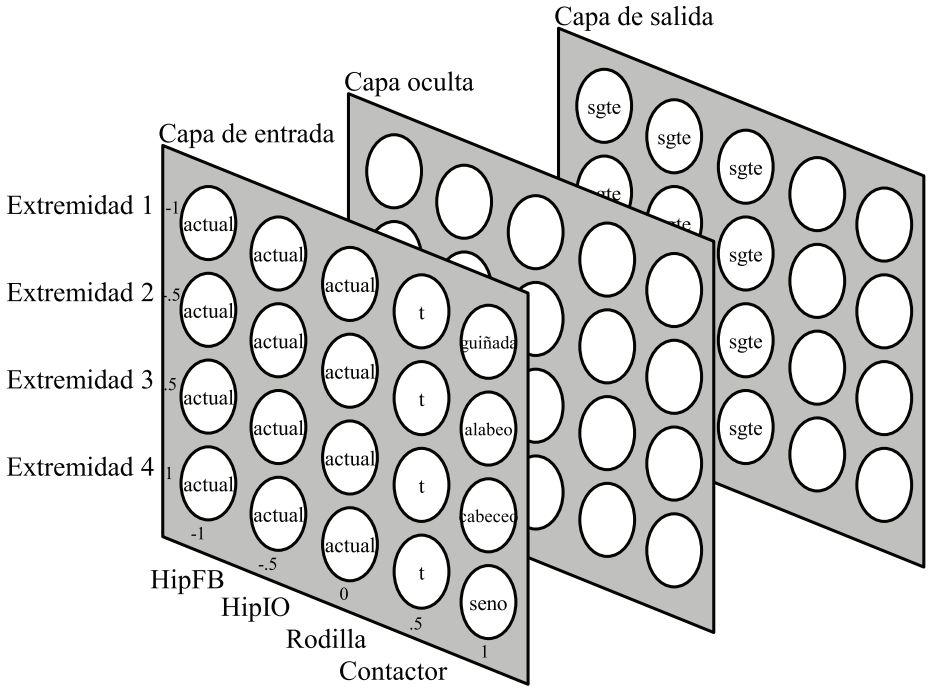


Figura 1.2: Configuración del substrato para HyperNeat y FT-Neat [1].

entradas (ganancia, x_1, y_1, x_2, y_2) y devuelve dos valores de salida. El primer valor determina el peso que vincula el nodo de la capa de entrada con el nodo respectivo de la capa oculta y el segundo valor determina el peso que vincula el nodo de la capa oculta con el nodo respectivo de la capa de salida. Todas las combinaciones entre nodos origen y destino son iterativamente ingresadas a la CPPN para determinar el valor del peso de cada conexión posible [1].

El segundo trabajo de planeamiento de movimientos es el de Bongard [2], que describe a un robot que puede recuperarse del daño en forma autónoma a través de un auto-modelamiento continuo. Este auto-modelamiento consiste en inferir indirectamente su propia estructura con el fin de generar traslación mediante la relación actuación/sensación. Asimismo, se expone que los sistemas robóticos pueden autónomamente sintetizar conductas complejas o recuperarse del daño mediante ensayos físicos de prueba y error, pero requiriendo de demasiados intentos. En cambio, se puede realizar un proceso activo de auto-modelamiento autónomo y continuo que permita a una máquina mantener su performance.

Además, un robot es capaz de indirectamente inferir su propia morfología a través de una exploración auto-dirigida y luego usar el modelo resultante para sintetizar nuevos comportamientos. Si la topología del robot inexplicablemente cambiara, el mismo proceso reestructuraría su modelo, lo cual llevaría a la generación cualitativa de comportamientos diferentes y compensatorios [2].

El robot implementado tiene cuatro extremidades, ocho uniones motorizadas, ocho sensores de ángulo, dos sensores de inclinación, cuatro sensores de contacto y un sensor de distancia para determinar la luz entre el suelo y la superficie inferior del cuerpo [7]. Este robot se muestra en la figura 1.3.

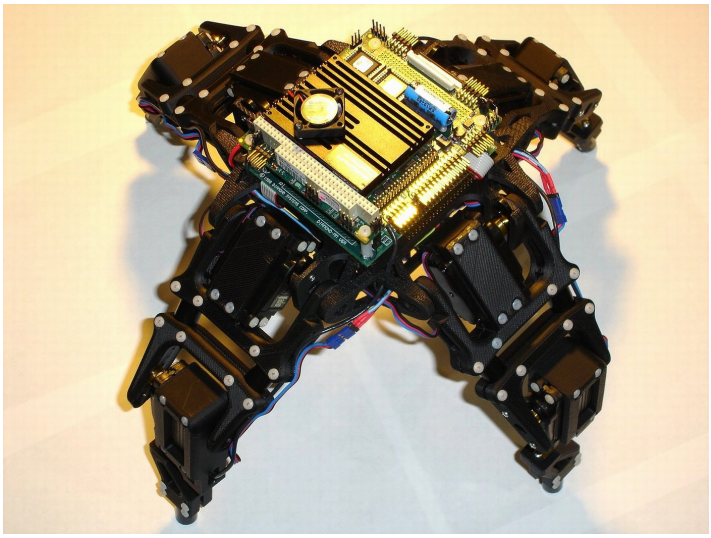


Figura 1.3: Robot cuadrúpedo [7].

En esencia, el proceso de auto-modelamiento continuo está compuesto por la modelación, el testeo y la predicción, y su funcionamiento es a través de la ejecución de ciclos continuos como se muestra en la figura 1.4. El ciclo empieza con la síntesis del auto-modelo (A y B), en la cual se grava la lectura de los sensores cuando el robot realiza un movimiento (A), que es aleatorio en un inicio y luego es la mejor acción encontrada (C). Luego, el robot genera varios modelos para coincidir con la información de los sensores mientras realiza la acción previa (B). El ciclo continua con la síntesis de la acción exploratoria (C), en la cual el robot

genera muchas posibles acciones que confrontan los modelos. Finalmente, el ciclo termina con la síntesis de la conducta objetivo (D), donde luego de varias etapas entre (A) y (C), el mejor modelo es usado para generar secuencias de traslación. En (E), la mejor secuencia de traslación es ejecutada por el dispositivo físico. En (F), el ciclo continúa en el paso (B) para refinar futuros modelos o para crear nuevos comportamientos como en el paso (D) [2].

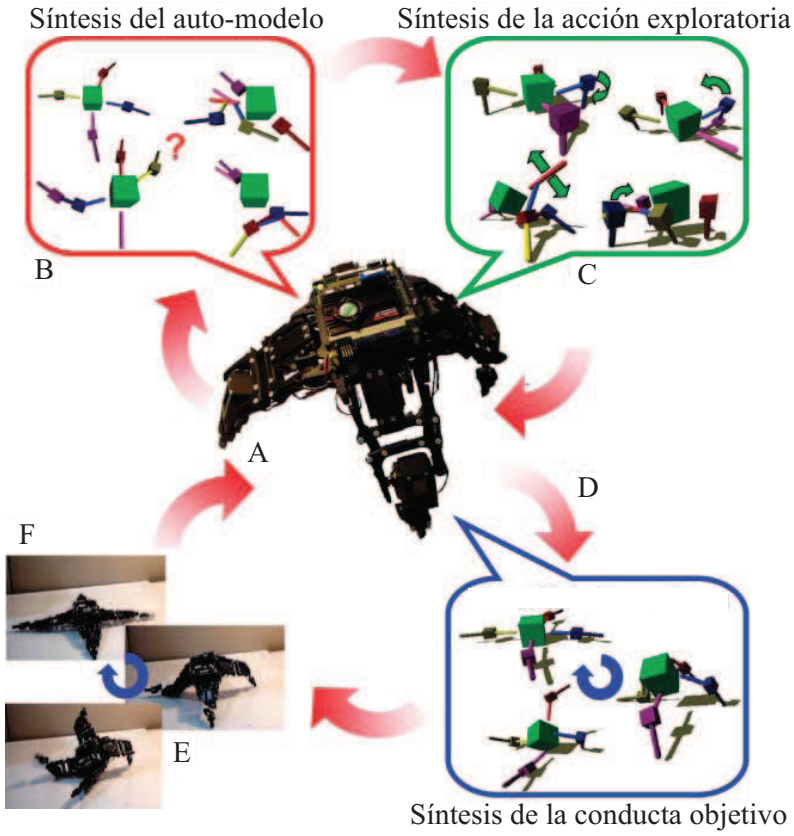


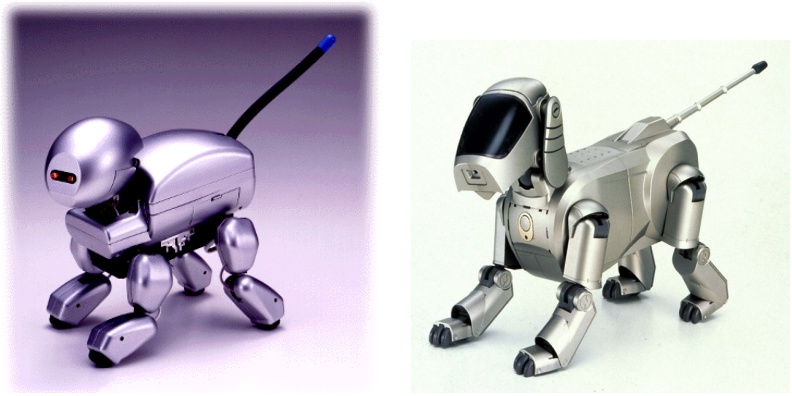
Figura 1.4: Esquema del algoritmo del robot [2].

Una característica importante del ciclo es el rol activo que juega el robot en determinar su mejor auto-modelamiento. Esto se observa cuando sufre un cambio en una de sus articulaciones. En este caso, el robot continúa con su mejor modelo anterior como referencia y, luego, el algoritmo realiza unos ajustes para determinar la nueva conducta del robot y lograr así seguir caminando [2].

Adicionalmente, en el robot se ha empleado un algoritmo de identificación

de sistemas, llamado algoritmo de estimación-exploración, para reconstruir los parámetros físicos desconocidos. Este algoritmo puede reconstruir la morfología de un robot desde el inicio, incluso a partir de los datos de los sensores [8]. Además, el algoritmo de estimación-exploración tiene por ventajas: primero, requerir de un número reducido de pruebas en el robot físico; segundo, permitir que la recuperación y la adaptación al medio ambiente sean continuas; y tercero, poder en ciertos casos recuperarse de daños o fallas compuestas o imprevistas [9].

El tercer trabajo que desarrolla el planeamiento de movimientos para un robot móvil articulado es el de Hornby [3]. En su investigación se describe un sistema de obtención autónoma de pasos para las dos versiones de robots de entretenimiento de Sony, prototipo OPEN-R y ERS-110 (primer robot AIBO[4]), que se muestran en la figura 1.5.



(a) pototipo OPEN-R (b) ERS-110

Figura 1.5: Robots de entretenimiento de Sony [3].

Tanto el prototipo OPEN-R como el ERS-110 tienen quince grados de libertad, que corresponden a los tres actuadores en la cabeza y a las cuatro extremidades, cada cual con tres grados de libertad. El prototipo OPEN-R tiene además una cola con un grado de libertad, lo que le da un total de dieciséis grados de libertad; y, por el otro lado, el ERS-110 tiene dos grados de libertad en la cola y dos orejas actuadas de un grado de libertad cada una, lo que le da un total de

[4]Artificial Intelligence roBOt

diecinueve grados de libertad. Además, ambos robots tienen una micro cámara, un micrófono estéreo, un dispositivo de posición y sensores de contacto ubicados sobre la cabeza y en la parte inferior de cada extremidad; así como, una CPU, una batería, un giroscopio y un acelerómetro albergados en el cuerpo [3].

Los pasos en ambos robots son llevados a cabo por el módulo de traslación, que controla el movimiento de las extremidades. Un paso es definido por un vector de parámetros que es usado por una función matemática de senos y cosenos para obtener un movimiento cíclico de la parte inferior de cada extremidad. Para el prototipo OPEN-R, el módulo de traslación usa veinte parámetros, de los cuales diecisiete parámetros son para especificar las trayectorias de cada extremidad y tres para suavizar el movimiento del robot al variar la ganancia de los motores de cada extremidad durante un ciclo de movimiento. En cambio, tomando como antecedente la experiencia con el prototipo OPEN-R, el módulo de traslación para el ERS-110 usa sesenta y un parámetros para definir un paso con el propósito de permitir a las extremidades delanteras y traseras seguir diferentes trayectorias, así como de añadir control a la postura del cuerpo [3].

Para autónomamente generar pasos, un algoritmo evolutivo optimiza los parámetros de cada paso al crear nuevos conjuntos de parámetros en base a estos de forma tal que maximicen la performance resultante del paso. Para hallar la performance, cada conjunto de parámetros es evaluado en el módulo de traslación al implementarlo en el robot. Este proceso ocurre dentro de un área rodeada de paredes con una marca en el centro de dos paredes opuestas. La primera parte de la prueba de evaluación consiste en centrar al robot en dirección a la marca. El robot se centra al rotar su cuerpo en una dirección fija en busca de la marca y, una vez detectada, continua girando hasta que la posición horizontal promedio de la marca esté dentro de los $\pm 6^{\circ}$ del centro de la imagen por un período de dos segundos. En la segunda parte de la prueba, el robot determina qué tan lejos está de la marca y empieza a correr hacia ella por siete segundos, usando un

conjunto de parámetros de traslación, o hasta que detecta que está a menos de 20 cm de la pared. En la tercera parte de la prueba, que empieza después que el robot se ha parado, este usa sus sensores para determinar la rectitud de su movimiento y la distancia que viajó. Todo el procesamiento es ejecutado por el controlador implementado en el robot [3].

Como resultado de la implementación, el algoritmo evolutivo fue capaz de evolucionar tanto una marcha como un trote. Para el prototipo OPEN-R, el algoritmo produjo marchas que eran más rápidas (10.2 m/min) que los trotes (6.5 m/min). Esto puede deberse, teniendo en cuenta al robot, a que las extremidades que se mueven juntas son las del mismo lado del cuerpo en la marcha, mientras que en el trote son las diagonalmente opuestas. Además, sin un torso que pueda girar en el medio, es más difícil levantar las dos extremidades lo suficientemente alto tanto en el trote como en la marcha. Por otro lado, para el ERS-110, la optimización de los parámetros fue realizada en un ambiente con varillas de plástico en el suelo. Esto se debe a que, según Hornby [3], un paso evolucionado depende del robot y del entorno en el que se desarrolló, con lo que podría trabajar en entornos más fáciles, pero no necesariamente en más difíciles. Hay que resaltar que cambiar de robots es una forma de cambiar de entorno debido a las diferencias en la fabricación y la calibración. Al final del experimento, se evolucionaron los parámetros de manera tal que los pasos eran lo suficientemente altos como para pasar los obstáculos, a diferencia de los parámetros evolucionados en ambientes sin obstáculos. La velocidad alcanzada del paso fue de 9 m/min [3].

El cuarto trabajo es el de Zhang [4] que ahonda en el planeamiento de movimientos para el robot AIBO de Sony. En esta investigación se ha utilizado el robot ERS-7, mostrado en la figura 1.6, que tiene en total 20 grados de libertad.

De igual forma que el prototipo OPEN-R y el ERS-110, el robot ERS-7 también tiene un módulo de traslación. El módulo de traslación determina los ángulos requeridos y los valores de PID para cada una de las articulaciones de acuerdo

Figura 1.6: Robot Sony AIBO ERS-7 [4].

con el comportamiento buscado, como por ejemplo caminar en una dirección y velocidad específica. Para esto, un paso es definido mediante un conjunto de 31 parámetros que permite calcular la trayectoria del cuerpo, el tiempo de transición y la trayectoria de las extremidades [4].

En relación a la generación del paso, esta es efectuada al optimizar los parámetros del módulo de traslación mediante un algoritmo genético. Esta optimización además ha empleado, por un lado, una analogía al método de la gradiente como proceso de aprendizaje, y por el otro, una evolución Lamarckiana, basada en el principio de que un cambio en el ambiente produce cambios en las necesidades de los organismos que viven en ese entorno, que a su vez causa cambios en sus comportamientos. Los resultados de la generación mostraron una velocidad máxima de 24.6 m/min con solamente el algoritmo de evolución y de 26.4 m/min incluyendo la evolución Lamarckiana y el proceso de aprendizaje [4].

El quinto trabajo de planeamiento de movimientos es el de Gallagher [5] donde muestra un controlador para la traslación de un robot hexápodo. Este controlador ha sido primero simulado y luego probado en el robot.

Para la simulación, se utilizó un agente insecto como se muestra en la figura 1.7. Cada extremidad del insecto tiene un pie (que puede subir o bajar), tres actuadores y un sensor (que mide su posición angular). De los tres actuadores,

el primero gobierna el estado del pie y los otros dos generan torques en sentido horario y antihorario respectivamente sobre la única articulación de la extremidad. Los torques sobre cada articulación son sumados y dependiendo del estado de cada pie trasladarán al cuerpo o rotarán la extremidad sobre su articulación [5].

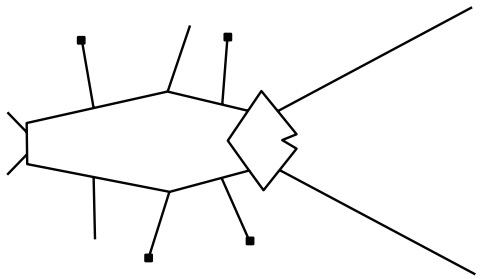


Figura 1.7: Cuerpo del insecto simulado [5].

Para el controlador de traslación, se emplearon redes dinámicas con neuronas de Hopfield. Cada extremidad es controlada por una red de cinco neuronas totalmente interconectadas, de las cuales tres son para los motores que gobiernan el estado de los torques de avance o de retroceso de las articulaciones y el estado del pie, y las dos restantes son interneuronas con un rol no específico. Asimismo, cada neurona tiene por entrada a la medición ponderada de los sensores de ángulo [5].

Estos controladores de traslación fueron probados y los resultados mostraron que eran capaces de dirigir adecuadamente el comportamiento del insecto bajo tres condiciones diferentes según la realimentación de los sensores: realimentación (1) siempre disponible, (2) nunca disponible o (3) disponible la mitad del tiempo. Los controladores evolucionados sin acceso a la información de los sensores funcionaron, pero no eran capaces de tomar ventaja de la información cuando esta estaba disponible. En cambio, los controladores evolucionados con acceso parcial a la realimentación fueron capaces de operar con y sin sensores, siendo la conducta para caminar generalmente mejor empleándolos [5].

Habiendo obtenido resultados satisfactorios en la simulación, la segunda parte de la investigación consistió en probar los controladores en un entorno real como el robot hexápodo de la figura 1.8.

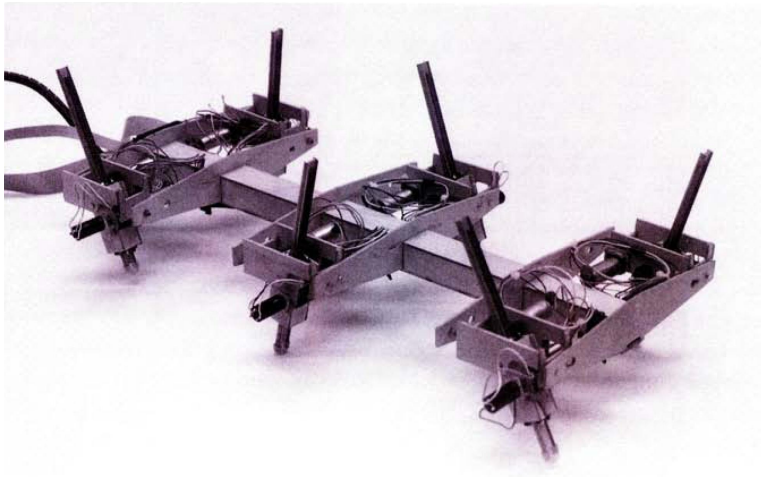


Figura 1.8: Robot hexápodo [5].

El robot hexápodo tiene seis extremidades y cada extremidad tiene dos grados de libertad: rotación y extensión. Ambos grados de libertad son realizados por dos motores DC y, adicionalmente, el movimiento radial es realizado por medio de una transmisión de cremallera y piñón. Por otro lado, el robot hexápodo tiene sensores de ángulo en cada extremidad, los cuales han sido implementados usando potenciómetros montados en paralelo con los motores. Además, los controladores de traslación fueron ejecutados en una computadora y comunicados con el robot mediante convertidores ADC y DAC de 8 bits [5].

El resultado de la implementación en el robot hexápodo muestra que la conducta para caminar del robot es consistente como la simulación predijo. Con la información disponible de los sensores, el robot caminó suavemente y, sin ella, el robot aún caminó pero con un paso marcado que tenía notables pausas, en las cuales el robot se detenía por completo [5].

Por último, el sexto trabajo que trata sobre el planeamiento de movimientos de un robot móvil articular es el de Wettergreen y Thorpe [6]. En su investi-

gación, ellos han desarrollado un robot articular construido para atravesar con alta fiabilidad terrenos extremos tal como la superficie de Marte. Este robot es el Ambler de la figura 1.9.

Figura 1.9: El Ambler [6].

El mecanismo del Ambler tiene, como se observa en la figura 1.10, seis extremidades con tres articulaciones: una rotacional y una prismática en el plano horizontal y otra prismática en el plano ortogonal, la cual proporciona el contacto con el suelo. Los sensores que tiene el Ambler son un telémetro láser y sensores de fuerza-torque montados en los pies. El telémetro es utilizado para generar mapas del terreno, y los sensores de fuerza-torque proporcionan información de apoyo sobre el terreno y la estabilidad de la posición actual [6].

Para la generación de pasos en terrenos extremos, Wettergreen y Thorpe [6] exponen que un robot debe caminar utilizando el mínimo número de pasos que produzcan el máximo progreso con la precaución adecuada. Entonces, para caminar autónomamente en un terreno áspero, no es suficiente adaptar un paso

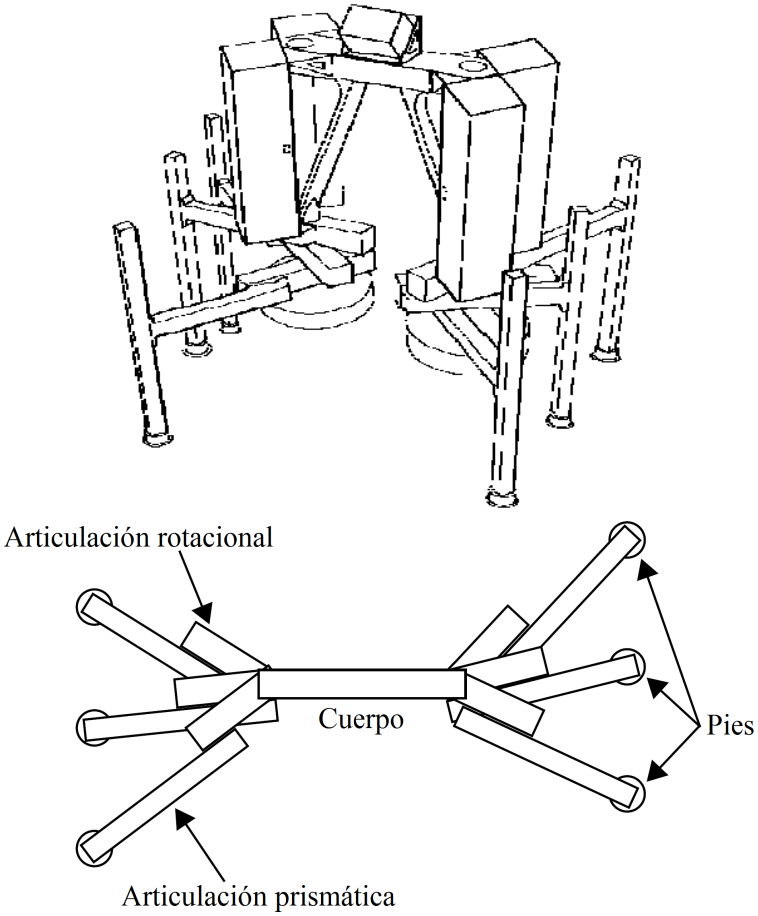


Figura 1.10: Vista en perspectiva y superior del Ambler [6].

fijo, sino que el robot debe operar dentro de los límites de su capacidad para maximizar su rendimiento y lograr fiabilidad y eficiencia. Es así que el problema se reduce a la generación de pasos libres de una postura por vez; por lo que, el mejor movimiento corporal va a ser seguido del mejor movimiento de extremidades [6].

Para esto, el generador de pasos del Ambler requiere información acerca de la configuración inicial del mecanismo, del terreno circundante (elevación) y de la trayectoria deseada. Una vez que ha sido otorgada esta información, el conjunto inicial de movimientos posibles del cuerpo es comparado con el mapa de profundidad del terreno para eliminar aquellos movimientos que estén en contacto con el terreno. Luego, para identificar límites cinemáticos y condiciones de colisión, la

trayectoria entre posturas es dividida en incrementos iguales, a los cuales se comprueba su viabilidad. Una vez que el rango de movimientos aceptables del cuerpo ha sido determinado, todos los movimientos que cumplan o superen un nivel específico de estabilidad, que puede variar según el entorno y la configuración del robot, son comparados y el máximo es seleccionado. En general, los movimientos de las extremidades que cumplen con las restricciones tanto del cuerpo como de sí mismas, que permiten un mayor avance del cuerpo en la dirección deseada y que aumentan la estabilidad de la postura son elegidos. Este funcionamiento permite al generador producir movimientos localmente óptimos a expensas de la performance global [6].

1.2. JUSTIFICACIÓN

Los sistemas exploradores con la habilidad de tomar información del medio para generar movimientos tienen por ventaja sólo necesitar modificaciones en su conducta para adaptarse ante cambios inesperados.

Dentro de las aplicaciones en la industria de los sistemas exploradores, se contemplan la exploración en lugares confinados como las inspecciones de tuberías. Ejemplos de estos robots son la familia de vehículos Theseus para tuberías de 25mm, 50mm y 150mm de diámetro [10], y el robot Kursk que se mueve usando bolas y utiliza la vibración como principio de traslación [11]. Otro campo es en atmósferas contaminadas, como la familia de robots ROBICEN para la inspección de plantas nucleares. Estos robots, cuyo movimiento está basado en cuatro ventosas, usan cuatro cámaras de video y un sensor ultrasónico para medir el espesor de la pared [12]. Asimismo, los robots exploradores también pueden ser empleados en la detección de minas, tal como el robot hexápodo COMET-I [13]. Otra aplicación es en la exploración de terrenos remotos, como los robots móviles con ruedas Spirit en el cráter de Gusev y Opportunity en la meseta de Meridiani

del planeta Marte [14, 15]. Finalmente, los robots exploradores pueden también ser utilizados en operaciones tácticas, como los "Man Portable Robotic Systems" (MPRS) para operaciones en áreas enemigas [16] o los robots vehiculares no tripulados Pioneer-AT y Urbie con navegación autónoma en interiores y percepción de ambientes urbanos para misiones militares [17, 18].

Estos ejemplos muestran la importancia de una unidad artificial que pueda adaptarse al medio como el robot RE1 (véase apéndice A). Para esto, en los dos algoritmos de planeamiento de movimientos implementados en el robot, se han usado redes neuronales. Las redes neuronales permiten al robot RE1 aprender cómo se relaciona cada posición de sus eslabones con la anterior e ir modificando esta relación al interactuar el robot con su entorno. Adicionalmente, se han empleado algoritmos genéticos en uno de los algoritmos de planeamiento de movimientos para realizar una búsqueda de la posición que maximice el desplazamiento. Los algoritmos genéticos han sido empleados porque pueden implementarse en un controlador netamente digital como el FPGA SPARTAN-3E y porque pueden operarse en funciones no explícitas como las redes neuronales. Por otro lado, los algoritmos de planeamiento de movimientos han sido implementados en el FPGA SPARTAN-3E por su propiedad de ser un dispositivo reconfigurable, por su versatilidad para realizar operaciones en forma simultánea (útil en la red neuronal), y por su velocidad de procesamiento; a pesar de que el uso de sus recursos no es tan eficiente en procesos netamente secuenciales como los algoritmos genéticos. Sin embargo, también pudo usarse otros controladores como el DSP siempre y cuando los recursos disponibles del controlador fueran suficientes y la velocidad real de procesamiento fuera capaz de cumplir con los tiempos de ejecución de los algoritmos.

Además, se ha introducido en esta tesis un nuevo método en tiempo real para el planeamiento de movimientos en un dispositivo FPGA de robots móviles articulares mediante la generación de secuencias de posiciones o mediante la relación en-

tre el desplazamiento del robot y su posición; en comparación con los antecedentes en donde la traslación del robot es generada (1) al simular una codificación generativa llamada HyperNEAT que evoluciona una red neuronal vinculada con la configuración del robot [1], (2) al inferir indirectamente la estructura del robot mediante la relación actuación/sensación [2], (3) al definir parámetros que son usados en funciones matemáticas de senos y cosenos para obtener un movimiento cíclico de cada extremidad [3], (4) al optimizar un conjunto de parámetros que definen el movimiento del robot mediante un algoritmo genético, un proceso de aprendizaje basado en el método de la gradiente y una evolución Lamarckiana [4], (5) al emplear una red neuronal para cada extremidad del robot que controle el torque de los motores [5], y (6) al dividir la trayectoria entre posturas en incrementos iguales y optimizarlas a expensas del desempeño global [6].

1.3. PLANTEAMIENTO DEL PROBLEMA

La tesis busca demostrar que los dos algoritmos propuestos de inteligencia computacional ("algoritmo para generar posiciones" y "algoritmo para generar desplazamientos") permitirán al robot RE1 actuar autónomamente en el planeamiento de sus movimientos de modo que este pueda desplazarse.

Para lograr esto, el robot RE1 está implementado con sensores que le permiten determinar su posición respecto a su medio y la posición de cada una de sus articulaciones. Con esta información el robot tiene una idea de cómo debe comportarse para poder mejorar su desplazamiento en cada intento.

1.4. OBJETIVOS

1.4.1. Objetivo General

El objetivo general de la tesis es:

Diseñar e implementar en la tarjeta de desarrollo FPGA SPARTAN-3E algoritmos basados en redes neuronales y algoritmos genéticos para el planeamiento de movimientos del robot explorador RE1, de forma tal que la velocidad de desplazamiento sea mejorada en cada intento de generación autónoma de movimientos.

1.4.2. Objetivos Específicos

En la tesis, se establecieron los siguientes objetivos específicos:

1. Desarrollar el "algoritmo para generar posiciones", basado en redes neuronales, que ejecuta el planeamiento de movimientos del robot RE1.
2. Desarrollar el "algoritmo para generar desplazamientos", basado en redes neuronales y algoritmos genéticos, que efectúa el planeamiento de movimientos del robot RE1.
3. Realizar los algoritmos de funcionamiento de los sensores y los actuadores del robot RE1 en el controlador FPGA SPARTAN-3E.
4. Resolver las operaciones matemáticas para la implementación de los algoritmos en el FPGA SPARTAN-3E.
5. Implementar los dos algoritmos de inteligencia computacional en el controlador FPGA SPARTAN-3E.
6. Demostrar la eficacia de la implementación de los dos algoritmos de inteligencia computacional.

1.5. ALCANCES Y LIMITACIONES

Tanto el robot RE1 como los dos algoritmos de planeamiento del movimiento son partes de una investigación. Debido a esto, el diseño del robot RE1 es un prototipo orientado a demostrar la generación de pasos mediante los algoritmos de inteligencia computacional.

Además, la implementación del robot y de las tarjetas electrónicas no están comprendidas en la tesis. De igual modo, los casos de falla o de daño al robot RE1 no califican en los escenarios en que se da el planeamiento o la generación de pasos.

Respecto a las limitaciones, el robot siempre tiene que tener en frente una pared que le sirva como superficie de referencia al sensor de distancia. Esta no puede estar separada a una distancia mayor de 3 metros ya que se estaría trabajando fuera del rango de operación del sensor. Además, el sensor siempre tiene que estar apuntando a la pared para que la onda de sonido no se pierda.

Asimismo, el robot RE1 no puede saltar, correr o hacer otras actividades que requieran de un conjunto de movimientos complejos. Sus acciones se restringen a desplazarse dado que el objetivo de los algoritmos desarrollados es planear la traslación; y además, porque todos los estados del sistema han sido tomados discretos ya que el tiempo de establecimiento para pasar de un estado a otro es menor al tiempo de ejecución entre órdenes dadas al robot RE1.

Por otro lado, no hay obstáculos que tenga que superar el robot RE1 durante la generación de pasos. El suelo en donde va a estar parado es plano, horizontal y liso al tacto.

Finalmente, el robot RE1 es un módulo de pruebas que necesita ser alimentado externamente mediante una fuente de voltaje.

CAPÍTULO 2
ALGORITMOS DE PLANEAMIENTO DE MOVIMIENTOS

2.1. FUNDAMENTOS

2.1.1. Redes Neuronales Artificiales

Las redes neuronales artificiales son un paradigma de procesamiento de información inspirado en el funcionamiento del cerebro y están compuestas por un número de elementos de procesamiento o neuronas que trabajan al unísono para resolver un problema específico [19].

Además, las redes neuronales artificiales tienen ciertas características de desempeño en común con las redes neuronales biológicas. Las redes neuronales artificiales han sido desarrolladas como generalización de modelos matemáticos de la cognición del humano o la biología neuronal y están basadas en los supuestos que:

1. El procesamiento de información ocurre en varios elementos simples llamados neuronas.
2. Las señales son propagadas entre neuronas a través de conexiones.
3. Cada conexión en una red neuronal típica tiene asociado un peso, que multiplica la señal transmitida.

4. Cada neurona ejecuta una función de activación usualmente no lineal a sus entradas (suma ponderada de las señales de entrada) para determinar sus señales de salida.

Es importante notar que una neurona sólo puede enviar una señal a la vez, aunque esta señal es trasmitida a varias otras neuronas [20].

Unidades Lógicas de Umbral[1]

Las unidades lógicas de umbral (threshold logic unit, TLU) o perceptrón son neuronas cuya estructura se encuentra en la figura 2.1 y cuya salida es 1 ó 0 dependiendo de si la suma ponderada de sus entradas es mayor o igual que el valor umbral, θ.

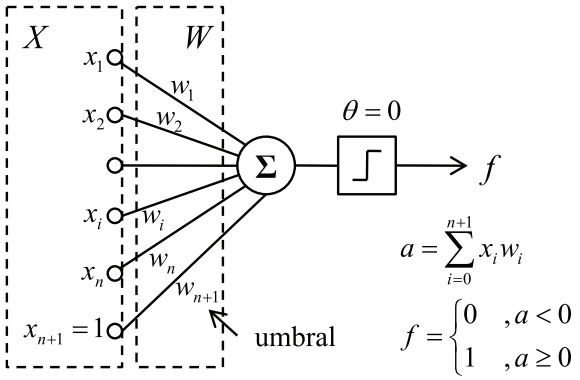


Figura 2.1: Unidad lógica de umbral [21].

El vector de entrada o la característica n-dimensional de la TLU se denota por $X = (x_1, ..., x_n)$ y sus pesos están representados por un vector n-dimensional $W = (w_i, ..., w_n)$, cuyas componentes son números reales o enteros. Además, la TLU tiene por salida 1 si $\sum_{i=1}^{n} x_i w_i \geq \theta$; de otro modo, su salida es 0. Esta operación también puede ser calculada por un producto punto $X \cdot W$. Así mismo, el umbral θ es fijado normalmente a 0; en ese caso, umbrales arbitrarios son determinados usando vectores aumentados

[1] cf. [21], p.39-41

$(n+1)$-dimensionales, Y y V, cuyas n primeras componentes son los mismos que los de X y W respectivamente. En consecuencia, la componente $(n+1)$-ésimo, x_{n+1}, del vector aumentado de características Y siempre tiene valor 1, mientras que la componente $(n+1)$-ésimo, w_{n+1}, del vector aumentado de pesos V se le asigna el negativo del valor umbral deseado. Entonces, la TLU tiene una salida de 1 si $Y \cdot V \geq 0$; de lo contrario, es 0.

Se puede dar a la TLU una descripción geométrica. Una TLU divide el espacio de entrada por un hiperplano como se muestra en la figura 2.2. El hiperplano está limitado entre los patrones donde $X \cdot W + w_{n+1} > 0$ y $X \cdot W + w_{n+1} < 0$ (estos patrones son llamados regiones de decisión [20]). Entonces, la ecuación del hiperplano es $X \cdot W + w_{n+1} = 0$. El vector unitario que es normal al hiperplano es $n = \dfrac{W}{|W|}$, donde $|W| = \sqrt{(w_1^2 + ... + w_n^2)}$ es la longitud del vector W. La distancia del hiperplano al origen es $\dfrac{w_{n+1}}{|W|}$ y la distancia desde un punto arbitrario X al hiperplano es $\dfrac{X \cdot W + w_{n+1}}{|W|}$.

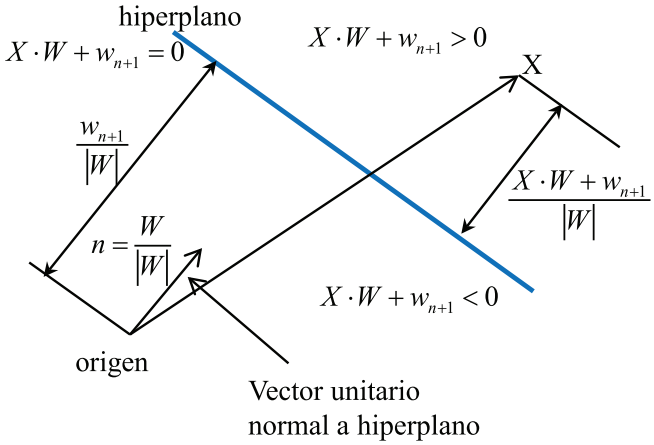


Figura 2.2: Hiperplano formado por la TLU [21].

Ajustar el vector de pesos W hace cambiar la orientación del hiperplano y ajustar w_{n+1} cambia la posición del hiperplano relativa al origen. De esta forma, el hiperplano puede ser movido para que la TLU implemente diferentes funciones linealmente separables.

Redes de TLUs[2]

Una neurona puede aprender solamente problemas linealmente separables [20]. Para clasificar correctamente todos los patrones en un set de entrenamiento linealmente no separable se requiere superficies más complejas que un hiperplano. Una forma de lograr superficies más complejas es con redes de TLUs.

En una red feedforward ninguna entrada de las TLUs depende de alguna salida de la red; mientras que en las recurrentes sí. Si las TLUs de una red feedforward están arregladas en capas con los elementos de la capa j recibiendo entradas sólo de las TLUs en la capa $(j-1)$, se dice que la red es una red feedforward multicapa. La red feedforward multicapa tiene la estructura mostrada en la figura 2.3. Todas las capas excepto la última (de salida) son capas ocultas.

Cabe resaltar que agregar capas adicionales no compensa una primera capa inadecuada de TLUs. La primera capa de TLUs particiona el espacio de características con el propósito de que no exista dos vectores diferentes en la misma región (esto es, que no exista dos vectores que circunscriban el mismo set de salidas de la primera unidad de capas). Si la primera capa no separa el espacio de características de esta forma; entonces, sin importar lo que las subsecuentes capas hagan, la salida final no será consistente con el set del entrenamiento.

Propiedades Básicas de las Redes Neuronales

Cada red neuronal tiene las siguientes propiedades básicas [22]:

1. **Características de las Neuronas:** Se tiene que definir en una red neuronal artificial la forma en qué cada elemento de procesamiento o

[2]cf. [21], p.51-54

neurona suma sus entradas, cómo estas entradas son transformadas en un nivel de activación y cómo el nivel de activación es actualizado y transformado en una salida.

2. **Conectividad:** Se tiene que describir qué neuronas están conectadas a cuáles y en qué dirección. Las direcciones pueden ser feedforward o recurrente.

3. **Regla de Propagación:** Se debe detallar cómo una activación determinada es propagada a través de la red neuronal. La regla de propagación tiene que ser explícita porque, en modelos neuronales más realísticos, se debe tener en cuenta las propiedades temporales del proceso de propagación, tales como los retardos.

4. **Regla de Aprendizaje:** Se tiene que especificar cómo los pesos de las conexiones entre neuronas cambian en el tiempo. En general, los pesos son actualizados según la ecuación 2.1 donde t representa el tiempo.

$$W(t+1) = W(t) + \Delta W \tag{2.1}$$

Tipos de Aprendizajes

- **Aprendizaje No-supervisado:** El aprendizaje no-supervisado consiste en la agrupación (o la detección de similitudes) de patrones de un conjunto dado de entrenamiento. La idea es optimizar (maximizar o minimizar) algún criterio o función de desempeño definido en términos de la salida de las unidades de la red neuronal. En este caso, se espera por lo general que los pesos y las salidas de la red converjan en representaciones que capturen las regularidades estadísticas de los datos de entrada [23].

- **Aprendizaje Supervisado:** En el aprendizaje supervisado, cada patrón de entrada es asociado con un patrón objetivo específico y deseado [23]. Para esto, la red neuronal debe aprender a relacionar el patrón de entrada con la salida al variar sus pesos. Estos pesos varían dependiendo de la magnitud del error que produce la red en la capa de salida: cuanto mayor sea el error, es decir, cuanto mayor sea la discrepancia entre la salida que produce la red y el valor correcto de salida, mayor será el cambio de los pesos [22].
- **Aprendizaje por Refuerzo:** El aprendizaje por refuerzo se utiliza en el caso que un comportamiento particular se quiere reforzar. Por lo general, se recibe una señal de refuerzo positivo si el resultado que se ha producido fue exitoso; en caso contrario, no se recibe refuerzo o se recibe una señal de refuerzo negativo [22].

Algoritmo de Entrenamiento Backpropagation[3]

El método de Backpropagation proporciona un método computacionalmente eficiente para modificar los pesos en una red neuronal feedforward, con funciones de activación diferenciable, con el propósito de aprender un conjunto de entrenamiento de entradas y salidas [23].

Si en la ejecución de la red se hace un error en un patrón, hay muchas formas diferentes en las cuales el error puede ser corregido. Sin embargo, es difícil asignar la "culpa" del error a una TLU particular en la red. Para esto, el método de la gradiente descendiente de Widrow Hoff ha sido generalizado para redes multicapa.

Tomando como referencia a la red de la figura 2.3. Cada una de las capas de TLUs tendrán salidas que se tomarán como componentes de un

[3]cf. [21], p.58-66

vector, tal como las características de entrada son componentes del vector de entrada. La j-ésima capa de TLUs ($1 \leq j < k$) tendrá como sus salidas al vector $X^{(j)}$. La característica de entrada se denota por $X^{(0)}$ y la salida final (de la k-ésima capa) por g. Cada una de la capas tiene un vector de pesos (conectándolo con sus entradas) y un umbral. La i-ésima TLU de la j-ésima capa tiene un vector de pesos denotado por $W_i^{(j)}$. Entonces, se denota la sumatoria de pesos de la i-ésima TLU de la j-ésima capa por $s_i^{(j)}$ (esto es $s_i^{(j)} = X^{(j-1)} \cdot W_i^{(j)}$). El número de TLUs en la j-ésima capa está dado por m_j. El vector $W_i^{(j)}$ tiene componentes $w_{l,i}^{(j)}$ para $l = 1, ..., m_{j-1} + 1$.

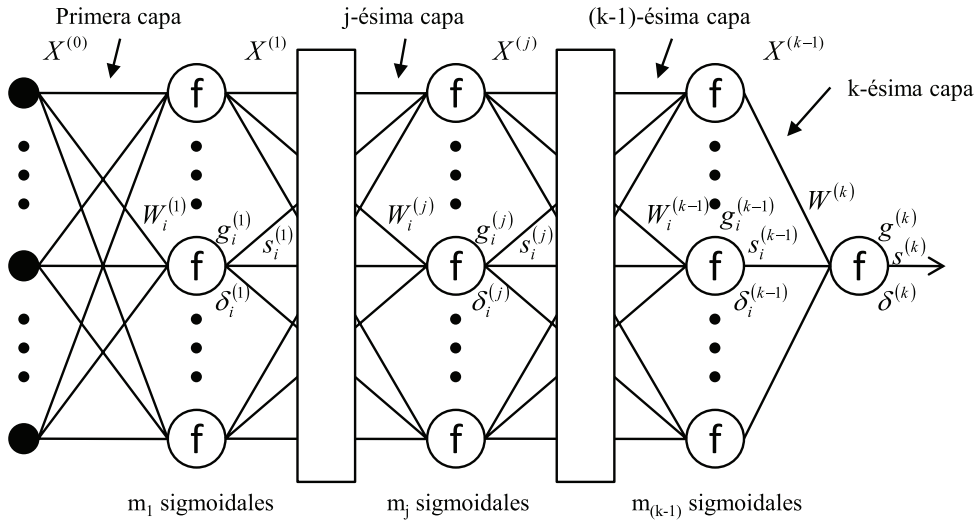


Figura 2.3: Red feedforward multicapa [21].

Además, con la respuesta deseada, d_i, para el i-ésimo vector de entrada, X_i, en el set de entrenamiento, Ξ, se calcula el error cuadrático del set de entrenamiento con:

$$\varepsilon = \sum_{X_i \in \Xi} (d_i - g_i)^2 \tag{2.2}$$

donde g_i es la respuesta actual de la red para la entrada X_i. Para hacer la gradiente descendiente en este error cuadrático, se ajusta cada peso en

la red por una cantidad proporcional al negativo de la derivada parcial de ε respecto a los pesos. Ahora, para hallar un procedimiento de ajuste de pesos, se emplea el error para un vector de entrada, X, cuya salida es g cuando la deseada es d:

$$\varepsilon = (d - g)^2 \tag{2.3}$$

Ya que la dependencia de ε con $W_i^{(j)}$ es mediante $s_i^{(j)}$, se puede usar la regla de la cadena:

$$\frac{\partial \varepsilon}{\partial W_i^{(j)}} = \frac{\partial \varepsilon}{\partial s_i^{(j)}} \cdot \frac{\partial s_i^{(j)}}{\partial W_i^{(j)}} \tag{2.4}$$

con $s_i^{(j)} = X^{(j-1)} \cdot W_i^{(j)}$, $\dfrac{\partial s_i^{(j)}}{\partial W_i^{(j)}} = X^{(j-1)}$.

Nótese que $\dfrac{\partial \varepsilon}{\partial s_i^{(j)}} = -2(d - g)\dfrac{\partial g}{\partial s_i^{(j)}}$. Por lo que,

$$\frac{\partial \varepsilon}{\partial W_i^{(j)}} = -2(d - g)\frac{\partial g}{\partial s_i^{(j)}} X^{(j-1)} \tag{2.5}$$

La cantidad $(d - g)\dfrac{\partial g}{\partial s_i^{(j)}}$, denotada por $\delta_i^{(j)}$, juega un rol importante en los cálculos ya que nos indica qué tan sensible es el error cuadrático de la salida de la red ante cambios en la entrada para cada función umbral.

Ya que estará cambiando el vector de pesos en direcciones alrededor de su gradiente negativo, la regla fundamental para los cambios de los pesos a través de la red será:

$$W_i^{(j)} \Leftarrow W_i^{(j)} + c_i^{(j)} \delta_i^{(j)} X^{(j-1)} \tag{2.6}$$

El siguiente paso es calcular $\delta_i^{(j)}$, usando la siguiente definición:

$$\delta_i^{(j)} = (d - g)\frac{\partial g}{\partial s_i^{(j)}} \tag{2.7}$$

Sin embargo, la salida de la red, g, no es diferenciable respecto a s en todo su dominio porque representa una función umbral. El artificio es reemplazar todas las funciones umbrales por funciones diferenciables como las sigmoidales, "logsig" (ecuación 2.8) o "tansig" (ecuación 2.9). Las salidas de las funciones sigmoidales, superpuesta a una función umbral, se muestra en las figuras 2.4 y 2.5.

$$logsig(s) = \frac{1}{1 + e^{-s}} \tag{2.8}$$

$$tansig(s) = \frac{2}{1 + e^{-2s}} - 1 \tag{2.9}$$

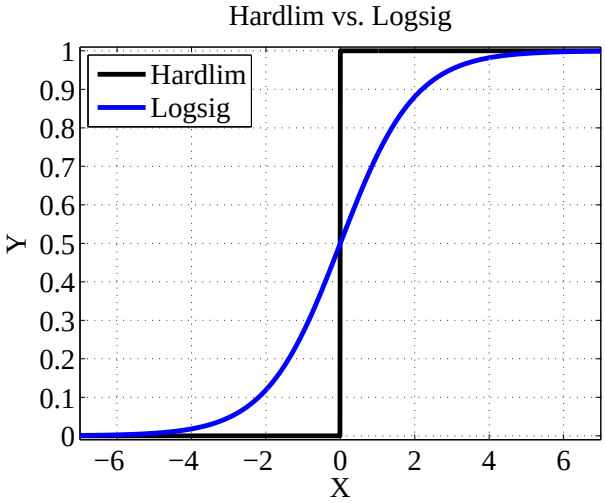


Figura 2.4: Comparación entre función "hardlim" y "logsig".

La salida de la i-ésima unidad sigmoidal en la j-ésima capa está denotada por $g_i^{(j)}$.

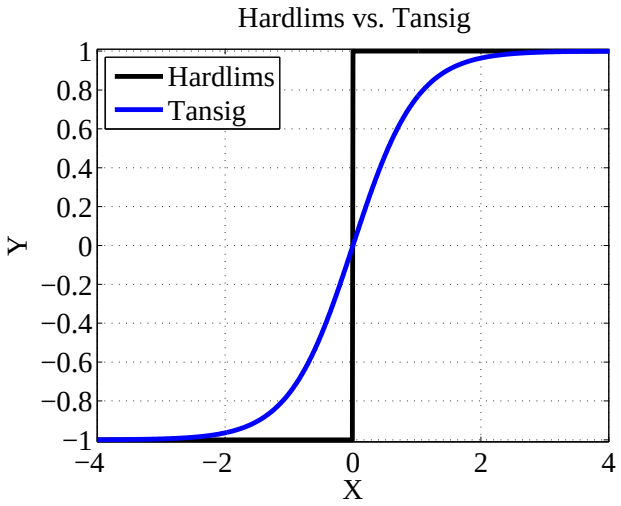


Figura 2.5: Comparación entre función "hardlims" y "tansig".

Primero hay que calcular los pesos para la capa de salida; para esto hallamos $\delta^{(k)}$:

$$\delta^{(k)} = (d - g^{(k)})\frac{\partial g^{(k)}}{\partial s^{(k)}} \tag{2.10}$$

Dadas las funciones sigmoidales, tenemos que $\frac{\partial g}{\partial s} = g(1-g)$ para la función "logsig" y $\frac{\partial g}{\partial s} = (1-g^2)$ para la función "tansig". Sustituyendo nos da:

$$\begin{aligned} \delta^{(k)} &= (d - g^{(k)})g^{(k)}(1 - g^{(k)}) \quad \text{, para "logsig"} \\ \delta^{(k)} &= (d - g^{(k)})(1 - g^{(k)2}) \qquad \text{, para "tansig"} \end{aligned} \tag{2.11}$$

Reescribiendo la ecuación 2.6, el vector de pesos en la capa final es cambiado de acuerdo a:

$$W^{(k)} \Leftarrow W^{(k)} + c^{(k)}\delta^{(k)}X^{(k-1)} \tag{2.12}$$

Puede considerarse que la función sigmoidal implementa un plano "difuso". Para un patrón más alejado del hiperplano difuso, $g(1-g)$ o $(1-g^2)$ tienen valores cercanos a 0, y la regla backpropagation hace poco o nada cambiando los valores de los pesos al orientarlos a la salida deseada.

Cambios en los pesos sólo son hechos dentro de la región difusa cercana al hiperplano, y estos cambios son en la dirección de corrección del error.

Ahora, se sigue con el cálculo de los pesos para las capas ocultas, para lo cual se emplea la ecuación 2.7 de forma semejante a los pesos de la capa de salida.

De nuevo, se usa la regla de la cadena. La salida final, g, depende de $s_i^{(j)}$ a través de cada una de las entradas sumadas a la simoidal en la capa $(j+1)$-ésima. Entonces,

$$\delta_i^{(j)} = (d-g)\left[\frac{\partial g}{\partial s_1^{(j+1)}}\frac{\partial s_1^{(j+1)}}{\partial s_i^{(j)}} + ... + \frac{\partial g}{\partial s_l^{(j+1)}}\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} + ... + \frac{\partial g}{\partial s_{m_{j+1}}^{(j+1)}}\frac{\partial s_{m_{j+1}}^{(j+1)}}{\partial s_i^{(j)}}\right]$$

$$\delta_i^{(j)} = \sum_{l=1}^{m_{j+1}}(d-g)\frac{\partial g}{\partial s_l^{(j+1)}}\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} = \sum_{l=1}^{m_{j+1}}\delta_l^{(j+1)}\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} \tag{2.13}$$

Sólo falta hallar el término $\dfrac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}}$, para lo cual nos basamos en su definición.

$$s_l^{(j+1)} = X^{(j)} \cdot W_l^{(j+1)} = \sum_{v=1}^{m_j+1} g_v^{(j)} w_{vl}^{(j+1)} \tag{2.14}$$

Como los pesos no dependen de s, entonces:

$$\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} = \frac{\partial\left[\sum_{v=1}^{m_j+1} g_v^{(j)} w_{vl}^{(j+1)}\right]}{\partial s_i^{(j)}} = \sum_{v=1}^{m_j+1} w_{vl}^{(j+1)}\frac{\partial g_v^{(j)}}{\partial s_i^{(j)}} \tag{2.15}$$

donde, $\dfrac{\partial g_v^{(j)}}{\partial s_i^{(j)}} = 0$ a menos que $v = i$, en tal caso su valor es semejante a la ecuación 2.11. Por lo que:

$$\begin{aligned}\frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} &= w_{il}^{(j+1)} g_i^{(j)}\left(1 - g_i^{(j)}\right) \quad \text{, para "logsig"} \\ \frac{\partial s_l^{(j+1)}}{\partial s_i^{(j)}} &= w_{il}^{(j+1)}\left(1 - g_i^{(j)2}\right) \quad \text{, para "tansig"}\end{aligned} \tag{2.16}$$

Usando este resultado para $\delta_i^{(j)}$, da:

$$\begin{aligned} \delta_i^{(j)} &= g_i^{(j)} \left(1 - g_i^{(j)}\right) \sum_{l=1}^{m_{j+1}} \delta_l^{(j+1)} w_{il}^{(j+1)} \quad \text{, para “logsig”} \\ \delta_i^{(j)} &= \left(1 - g_i^{(j)2}\right) \sum_{l=1}^{m_{j+1}} \delta_l^{(j+1)} w_{il}^{(j+1)} \quad \text{, para “tansig”} \end{aligned} \tag{2.17}$$

La ecuación anterior es recursiva en el término δ. Es interesante notar que esta expresión es independiente de la función error, que afecta explícitamente sólo a $\delta^{(k)}$. Habiendo calculado el término $\delta_i^{(j+1)}$ para la capa $j+1$, se sigue con el término $\delta_i^{(j)}$.

Reescribiendo la ecuación 2.6, el vector de pesos en las capas ocultas es cambiado de acuerdo a:

$$W_i^{(j)} \Leftarrow W_i^{(j)} + c_i^{(j)} \delta_i^{(j)} X^{(j-1)} \tag{2.18}$$

2.1.2. Algoritmos Genéticos

“La computación Evolutiva es un enfoque alternativo para abordar problemas complejos de búsqueda y aprendizaje a través de modelos computacionales de procesos evolutivos”. Las aplicaciones específicas de estos modelos son los algoritmos evolutivos, cuyo fin es guiar una búsqueda estocástica al evolucionar a un conjunto de estructuras y al seleccionar las más apropiadas. En consecuencia, el propósito final de los algoritmos evolutivos es la “supervivencia del más apto” y es alcanzado mediante la “adaptación al entorno” [24].

A continuación, se detallará los principios de los algoritmos genéticos, los cual fueron tomados de “Una Introducción a la Computación Evolutiva” [24].

Definición

"Los Algoritmos Genéticos son métodos estocásticos de búsqueda ciega de soluciones cuasi-óptimas. En ellos se mantiene una población que representa a un conjunto de posibles soluciones la cual es sometida a ciertas transformaciones con las que se trata de obtener nuevos candidatos y a un proceso de selección sesgado en favor de los mejores candidatos" [24].

Hay que señalar que los algoritmos genéticos añaden un mecanismo de selección de soluciones. Este mecanismo tiene dos vertientes: a corto plazo, los mejores tienen más posibilidades de sobrevivir (relacionado al operador de reemplazo), y a largo plazo, los mejores tienen más posibilidades de tener descendencia (vinculado al operador de selección).

Además, los algoritmos genéticos son métodos de búsqueda (1) *ciega* ya que no utilizan información específica del problema, (2) *codificada* debido a que trabajan sobre las representaciones de los elementos del dominio del problema, (3) *múltiple* porque buscan en forma simultánea entre los candidatos, y (4) *estocástica* ya que pueden controlar la penetración de la búsqueda. Todas estas características otorgan al final mayor robustez a la búsqueda.

Por otra parte, los objetos que evolucionan en un algoritmo genético son cadenas binarias "v" que codifican a los elementos "x" del espacio de búsqueda. Asimismo, la búsqueda es guiada por la función de aptitud $u(x)$, que se obtiene a partir de la función de evaluación $f(x)$ del correspondiente problema. Se debe resaltar que se denomina evaluaciones o aptitudes brutas a los valores de la función de evaluación, y aptitudes a secas o aptitudes netas a los valores de la función de aptitud.

Asimismo, los algoritmos genéticos son un ejemplo fundamental de los algoritmos evolutivos, por lo que siguen el esquema que se muestra en la

figura 2.6. En este esquema, una población de "n" miembros es ingresada al proceso de *selección* para obtener una población de "n" criadores, de la cual se escoge un grupo de "w" progenitores que son los que se van a reproducir. Luego, los progenitores generan "s" nuevos individuos que constituyen su descendencia al ser modificados por los operadores genéticos en la fase de *reproducción*. Finalmente, para formar la nueva población$(t + 1)$, se seleccionan "n" supervivientes de entre los criadores y la descendencia en la fase de *reemplazo*.

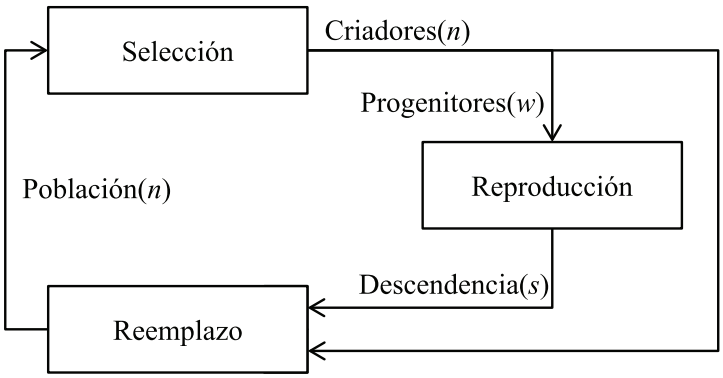


Figura 2.6: Bucle básico del algoritmo genético [24].

Criterios Necesarios para la Implementación

1. **Criterio de codificación**: Se refiere al procedimiento (codificación) de hacer corresponder cada punto del dominio del problema con una cadena binaria (representación).
2. **Criterio de tratamiento de los individuos no factibles**: Las cadenas que codifican elementos no válidos del espacio de búsqueda deben ser reconocidas.
3. **Criterio de inicialización**: Se debe definir la población inicial con la que empieza el algoritmo genético.
4. **Criterio de parada**: Se refiere a las condiciones que indican si el

algoritmo genético ha encontrado una solución aceptable o si ha fracasado.

5. **Funciones de evaluación y aptitud**: Se debe determinar la función de evaluación y aptitud más apropiada para el problema.
6. **Operadores genéticos**: Se tiene que precisar los operadores para el proceso de reproducción, tales como el cruce y la mutación, etc. y sus variaciones.
7. **Criterios de selección**: Se debe establecer el proceso de búsqueda en favor de los miembros más aptos.
8. **Criterios de reemplazo**: Se refiere a los criterios con que se seleccionan a los criadores y a los supervivientes.
9. **Parámetros de funcionamiento**: Estos son el tamaño de la población, las probabilidades de aplicación de los operadores genéticos, la precisión de la codificación, las tolerancias de la convergencia, etc.

Mecanismos de Muestreo de Poblaciones

El muestreo de poblaciones es la selección de un subconjunto de "k" individuos de una población. Existen tres grupos en relación al grado de azar en el proceso:

1. **Muestreo directo**: Se sigue un criterio fijo, como "los k mejores", "los k peores", etc. para escoger un subconjunto de individuos de la población.
2. **Muestreo aleatorio simple o equiprobable**: Todos los individuos de la población tienen las mismas probabilidades de ser escogidos.
3. **Muestreos estocásticos**: Las probabilidades de selección de los individuos de la población están en función de su aptitud. Si u_i es la

aptitud del individuo x_i, la puntuación o aptitud relativa p_i se halla según la ecuación 2.19.

$$p_i = \frac{u_i}{u_1 + ... + u_n} \qquad (\forall i = 1, ..., n) \tag{2.19}$$

Se observa de la expresión que se trata de una distribución de probabilidades ya que $p_1 + ... + p_n = 1$.

A continuación, se explican algunos tipos de muestreo estocástico:

- **Por sorteo**: Se constituye la muestra realizando "k" ensayos de una variable aleatoria. Los pasos a seguir son los siguientes:

  1. Se calculan las puntuaciones acumuladas:

$$\begin{aligned} q_0 &= 0 \\ q_i &= p_1 + ... + p_i \, (\forall i = 1, ..., n) \end{aligned} \tag{2.20}$$

  2. Se generan "k" números aleatorios uniformes $r_j \leftarrow [0, 1]$ $(\forall j = 1, ..., k)$.
  3. Se escoge el individuo x_i para cada $j = 1, ..., k$ que cumpla con

$$q_{i-1} < r_j < q_i$$

- **Por restos**: Cada individuo x_i tiene $[p_i.k]$ puestos en la muestra y se reparten, usualmente por sorteo, los puestos vacantes en función de sus puntuaciones.
- **Universal o por ruleta**: A cada elemento de la población se le asigna una porción de una ruleta, la cual es proporcional a su aptitud relativa. Luego, la ruleta se gira una sola vez y, mediante "k" puestos equidistantes, se seleccionan "k" individuos [25].

Para simular un giro de la ruleta, los siguientes números son definidos a partir del número aleatorio simple $r \leftarrow [0, 1]$:

$$r_j = \frac{r + j - 1}{k} \qquad (\forall j = 1, ..., k) \tag{2.21}$$

Con esto, los individuos son escogidos al comparar los r_j al igual a como se hace con el muestreo por sorteo.

Cabe resaltar que en la bibliografía antigua se suele llamar muestreo por ruleta (roulette wheel sampling) al muestreo por sorteo (lottery sampling) y viceversa.

- **Por torneos**: Se muestrean "k" individuos al repetir "k" veces el proceso de elegir al mejor individuo de un subconjunto de z elementos que han sido tomados aleatoriamente de la población base.

  Otra opción es elegir dos individuos de la población al azar. Luego, se genera un número aleatorio $r \leftarrow [0, 1]$. Si $r < p$ (donde p es un parámetro), se selecciona el individuo con mejor aptitud; de lo contrario, se selecciona el individuo de menor aptitud. Los dos individuos son devueltos a la población original y pueden ser seleccionados de nuevo [25].

- **Elitismo**: Es un complemento a los métodos de selección. Fuerza al algoritmo genético a conservar un cierto número de los mejores individuos en cada generación ya que estos se pueden perder si no son seleccionados o si son destruidos por un operador genético [25].

Además, los mecanismos de muestreo aceptan variaciones:

1. **Muestreo diferenciado:** Los elementos de la población base pueden ser escogidos para formar la muestra sólo una vez.

2. **Muestreo conservador o duro:** Todos los elementos de la población base pueden ser escogidos.
3. **Muestreo excluyente extintivo:** Algunos individuos son descartados del proceso de muestreo a priori.

Procesos del Algoritmo Genético

- **Selección**: Involucra el muestreo de los "n" elementos de la población de criadores a partir de la población inicial.

  El propósito de la selección es hacer hincapié en los individuos de la población con mejor aptitud con la esperanza de que su descendencia a su vez tenga aptitudes elevadas [25]. Sin embargo, tomar simplemente a los mejores individuos e ignorar a los otros daría lugar a una pérdida rápida de la diversidad. Los individuos que no tienen tan buena aptitud pueden tener propiedades que demuestren superioridad en el largo plazo [22]. No obstante, una selección basada en individuos no tan aptos resulta también en una evolución lenta [25].

- **Reproducción**: Se basa en conseguir una descendencia de "s" miembros a partir de la aplicación de los operadores de transformación como el cruce, la mutación, la inversión, etc. sobre ciertos miembros de la población de criadores. Hay que señalar que la población variará más de una generación a otra si el valor de "s" es más grande.

  Existen dos grupos de operadores básicos: el cruce y la mutación. El cruce monopunto forma dos descendientes a partir del intercambio de segmentos de dos progenitores que han sido divididos en una posición elegida al azar (véase figura 2.7). Para esto, se genera un número aleatorio $r \leftarrow [1, l-1]$ que indica la posición en donde se corta a ambos progenitores. Para determinar quiénes se van a cruzar, se genera

un número aleatorio $r_i \leftarrow [0, 1]$ para cada individuo y son seleccionados aquellos con $r_i < p_{cruce}$ (probabilidad de cruce). En caso que un individuo quedara desemparejado, se lo elimina o se lo empareja con otro elegido aleatoriamente.

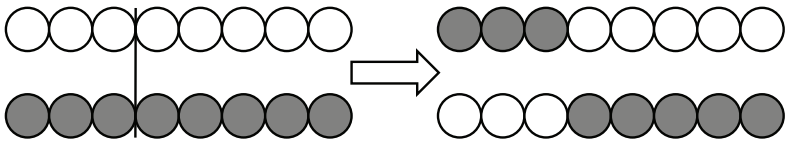


Figura 2.7: Operador genético: Cruce.

La mutación bit a bit conmuta un bit de entre todos los bits de la población (véase figura 2.8). Para determinar qué bits van a mutar, se genera un número aleatorio $r_i \leftarrow [0, 1]$ para cada uno de los $n \times l$ bits de la población y son elegidos aquellos con $r_i < p_{mutación}$ (probabilidad de mutación).

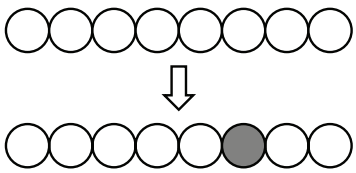


Figura 2.8: Operador genético: Mutación.

De los operadores se observa que el proceso de reproducción efectúa la búsqueda propiamente dicha ya que el cruce se encarga de explotar las mejores características de la población (búsqueda en profundidad); y la mutación, de explorar nuevos dominios del espacio de búsqueda (búsqueda en anchura).

- **Reemplazo**: Consiste en obtener una nueva población de "n" miembros a partir de los "n" miembros de la población de criadores y de los "s" miembros de la población de descendientes. Se distinguen varios tipos:

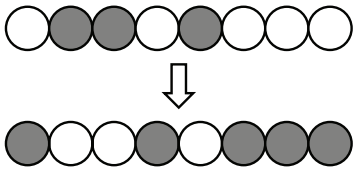


Figura 2.9: Operador genético: Inversión.

1. **Reemplazo inmediato (o al vuelo)**: Los "s" descendientes suplantan a sus progenitores.
2. **Reemplazo con factor de llenado**: Los "s" descendientes ocupan el lugar de los miembros de la población de criadores que más se les asemejen.
3. **Reemplazo por inserción (o de tipo coma)**: Existen dos casos según el tamaño de la descendencia y de la población:
   - $s \leq n$. Se muestrean "s" miembros de la población de criadores, los cuales serán reemplazados por los descendientes.
   - $s > n$. Se constituye la nueva población muestreando "n" miembros de la población de descendientes (selección generacional [22, 25]).
4. **Reemplazo por inclusión (o de tipo más)**: Se muestrean "n" miembros de una población que agrupa a los "s" descendientes con los "n" progenitores.

Esquemas

Un esquema es un patrón de similitud entre un grupo de individuos. Por ejemplo, el esquema (*1*10011) incorpora a los individuos 01010011, 01110011, 11010011 y 11110011.

Sus propiedades son las siguientes:

1. Un esquema con "k" símbolos de indiferencia incorpora a 2^k cadenas

binarias.

2. Una cadena binaria de longitud "len" coincide con en 2^{len} esquemas.
3. Todas las cadenas binarias de longitud "len" forman 3^{len} esquemas.
4. Una población compuesta por "n" cadenas binarias de longitud "len" abarca entre 2^{len} y $n \cdot 2^{len}$ esquemas.

Además, se definen para una población de cadenas binarias $P[t]$ y un esquema S:

- **Orden de un esquema, o(S)**: Es el número de posiciones fijas del esquema S.
- **Longitud característica de un esquema,** $\delta(\mathbf{S})$: Es la distancia entre las posiciones fijas extremas del esquema S.
- **Presencia de S en P[t],** $\xi(\mathbf{S}, \mathbf{P[t]})$: Es el número de cadenas de la población $P[t]$ que coinciden con el esquema S.
- **Aptitud del esquema S en P[t], Fitness(S,P[t])**: Es el promedio de las aptitudes de las cadenas de la población $P[t]$ que coinciden con el esquema S en el instante t.
- **Aptitud media de la población en el instante t, AveFitn(P[t])**: Es la aptitud del esquema $(*...*)$ en $P[t]$ en el instante t.

$$AveFitn(P[t]) = Fitness((*...*), P[t]) \tag{2.22}$$

- **Aptitud relativa de S en P[t], AveFitn(S,P[t])**: Es la aptitud del esquema S en la población $P[t]$ respecto a la aptitud media de la población $P[t]$ en el instante t.

$$AveFitn(S, P[t]) = \frac{Fitness(S, P[t])}{AveFitn(P[t])} \tag{2.23}$$

Ecuación de Crecimiento de Esquemas

En un algoritmo genético, la presencia de un esquema S en la población $P[t]$, compuesta por "n" elementos de tamaño "len", evoluciona en promedio según la fórmula 2.24.

$$\overline{\xi}\left(S, P[t+1]\right) = \overline{\xi}\left(S, P[t]\right) \cdot k_g \cdot k_s \tag{2.24}$$

donde k_g es el factor de crecimiento, que mide la tendencia del esquema S a aumentar su presencia en la población, y k_s es el factor de supervivencia, que mide la probabilidad de que el esquema S continue en la siguiente generación.

Ahora bien, la selección permite aumentar la presencia de los esquemas en la población intermedia. En el caso de una selección por sorteo, el factor de crecimiento k_g será,

$$k_g = AveFitn\left(S, P[t]\right) \tag{2.25}$$

Por otro lado, los operadores genéticos pueden destruir a los esquemas antes de que pasen a la siguiente generación, lo cual está cuantificado por el factor de supervivencia k_s:

$$k_s \geq \left(1 - \frac{p_{cruce} \cdot \delta(S)}{len - 1}\right)\left(1 - p_{mutación}\right)^{o(S)} \tag{2.26}$$

Teorema Fundamental de los Algoritmos Genéticos

La presencia de un esquema S evoluciona estadísticamente en progresión geométrica en la población $P[t]$, como se determina en la ecuación 2.24. Entonces,

- La presencia de los esquemas se incrementa exponencialmente cuando su aptitud está por encima de la media (aventajados) porque $k_g > 1$.
- La presencia de los esquemas aventajados se incrementa cuando estos son cortos y de bajo orden ya que $k_s \simeq 1$.

Con esto, se puede entender de otra forma a los operadores genéticos:

- **Selección**: La selección busca incrementar la presencia de los esquemas aventajados. Sin embargo, no introduce nuevos esquemas.
- **Cruce**: El cruce intercambia información útil como esquemas cortos, de bajo orden y aventajados entre individuos.
- **Mutación**: La mutación proporciona variedad en los esquemas, con lo que evita perder información.

Hipótesis de los Bloques Constructivos

La hipótesis de los bloques constructivos sostiene que el algoritmo genético funcionará eficazmente si hay mayor intervención de esquemas aventajados, cortos y de bajo orden llamados bloques constructivos.

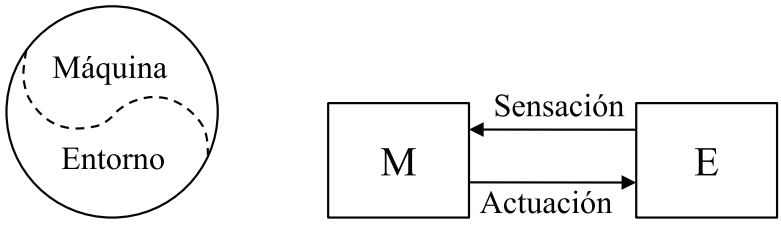
Propiedad del Paralelismo Implícito

Los algoritmos genéticos procesan en cada generación "n" estructuras junto con al menos n^3 esquemas, realizando así implícitamente una búsqueda en paralelo. Entonces, los algoritmos genéticos ejecutan externamente cadenas de códigos e internamente patrones de similitud entre ellas.

2.1.3. Planeamiento de Movimientos

El planeamiento de movimientos se refiere a la construcción de entradas de un sistema dinámico no lineal (máquina), las cuales lo conducen desde un estado inicial a un estado objetivo específico [26].

Hay que notar que el límite entre la máquina y su entorno es una línea arbitraria que varía de problema en problema (figura 2.10(a)). Una vez trazado, los sensores proveen de información sobre el entorno, lo que provee de entradas a la máquina durante su ejecución. Además, la máquina ejecuta acciones, que proveen de actuación sobre el entorno. La actuación puede alterar el entorno en alguna forma que es más tarde medido por los sensores (figura 2.10(b)). De esta forma, la máquina y su entorno están muy relacionados durante su ejecución [26].



(a) (b)

Figura 2.10: Relación entre una máquina y su entorno [26].

Planificador

El planificador, ya sea una máquina o una persona, es quién construye un plan. En el caso que el planificador sea una máquina, se considerará a este como un algoritmo de planeamiento. Por otro lado, en algunos casos, las personas pueden ser planificadoras al desarrollar un plan que funcione en todas las situaciones. El modelo del planeamiento es dado como entrada a la persona y esta "computa" un plan. Además, en este caso, no hay entradas adicionales porque la persona cumple el rol del algoritmo [26].

Plan

Un plan impone una estrategia o comportamiento específico. Un plan puede simplemente especificar una secuencia de acciones a ser tomadas. Sin embargo, puede ser más complicado. Si fuera posible predecir estados futuros, entonces el plan podría especificar acciones como una función de estado. En este caso, sin importar los estados futuros, se determina la acción adecuada. Por otro lado, puede ser que el estado no pueda ser medido. En este caso, la acción adecuada puede ser determinada desde cualquier información que se tenga. Esto generalmente será referido como una información de estado, en el cual las acciones de un plan son condicionadas [26].

Una vez que el plan ha sido determinado, existen tres formas de emplearlo [26]:

1. **Ejecución**: La ejecución puede ser realizada ya sea en simulación o en un dispositivo mecánico (robot) conectado al mundo físico.

   Un plan es ejecutado mayormente por una máquina, siendo la ejecución de dos tipos. En la primera, el planificador produce un plan, que está codificado de alguna manera y es dado como entrada a la máquina (figura 2.11(a)). En este caso, la máquina es considerada programable y puede aceptar planes posibles antes de la ejecución. Además, se asume que la primera vez que el plan es dado, la máquina se vuelve autónoma y puede dejar de interactuar con el planificador.

   En el segundo caso, el plan producido por el planificador codifica una máquina entera (figura 2.11(b)). El plan es una máquina de propósito especial que ha sido diseñada para solucionar tareas específicas dadas originalmente por el planificador. Bajo esta interpretación, se puede ser minimalista y diseñar una máquina lo más pequeña posi-

ble que suficientemente resuelva las tareas deseadas.

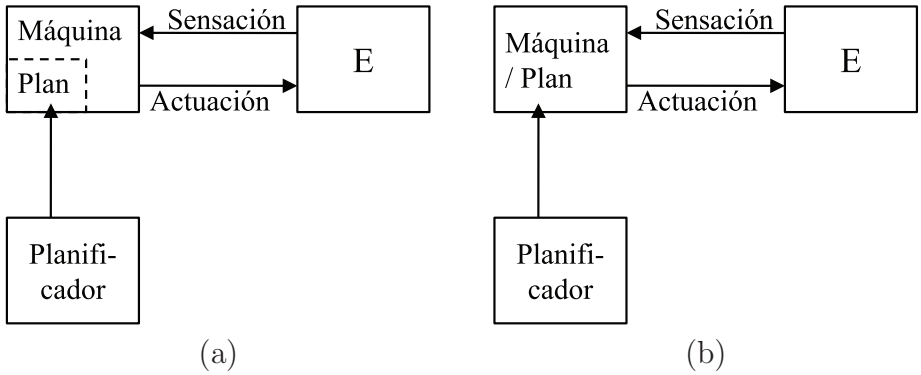


Figura 2.11: Tipos de ejecución de una máquina [26].

2. **Refinamiento**: Si un plan es refinado, entonces el planificador lo acepta como entrada y determina un nuevo plan que es una mejora en el mejor de los casos. Asimismo, el nuevo plan puede tomar en cuenta más aspectos del problema o puede simplemente ser más eficiente. Varios refinamientos pueden ser repetidamente aplicados para producir una secuencia de planes mejorados hasta que el final es ejecutado.

3. **Inclusión jerárquica**: Bajo inclusión jerárquica, un plan es incorporado como una acción en un plan más grande. El plan puede ser considerado como una subrutina en un plan mayor. Para que esto funcione, es importante que el plan garantice terminación con el propósito de que el plan mayor pueda ejecutar más acciones cuando lo requiera. Por otro lado, la inclusión jerárquica puede ser ejecutada varias veces, con lo que se forma un árbol de planes. En este árbol, cada vértice es un plan y el vértice principal representa el plan maestro.

2.2. ALGORITMOS DE PLANEAMIENTO DE MOVIMIENTOS

2.2.1. Algoritmo para Generar Posiciones

Este algoritmo busca generar en cada intervalo de tiempo la posición siguiente que debe tener cada articulación para que el robot pueda desplazarse. Para esto, se entrena una red neuronal, la cual tiene por entrada el ángulo actual de cada servomotor y por salida el ángulo siguiente. En la figura 2.12 está el esquema del algoritmo completo.

El primer bloque es el "Generador de Posiciones" que es donde está implementada la red neuronal de tipo "perceptrón" (modelo del apartado 2.1.1), cuya arquitectura consta de 8 entradas (una por cada articulación), una capa oculta con 8 neuronas con función de transferencia "purelin" (ecuación 2.27) cada una y la capa de salida con 8 neuronas (de igual manera, una por cada articulación). Para determinar la arquitectura de la red, se probó con diferentes redes feedforward de diferentes arquitecturas de modo que aquella que aprendía satisfactoriamente una secuencia básica de movimiento (la secuencia era conocida a priori) empleando menos recursos era seleccionada. La arquitectura de la red se muestra en la figura 2.13 y el cálculo en el apéndice B.

$$purelin(n) = n \tag{2.27}$$

El segundo bloque es el "Optimizador del Generador de Posiciones" que es donde está el algoritmo de entrenamiento de la red. El funcionamiento del algoritmo de entrenamiento se basa en ensayos de prueba y error. Primero, el algoritmo genera aleatoriamente las variaciones de los pesos de la red, las cuales son sumadas a los pesos actuales. Después, se prueban estos nuevos pesos durante cuatro ciclos completos de trabajo del robot

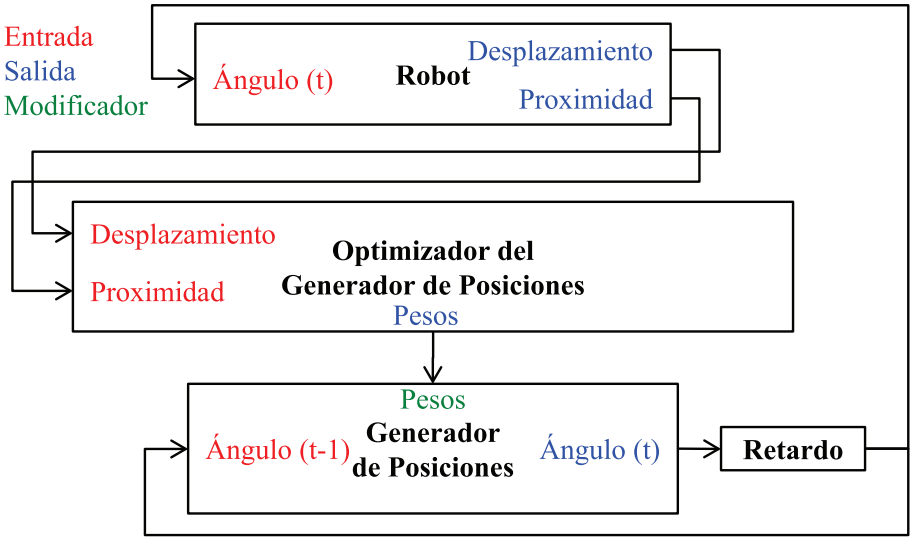


Figura 2.12: Esquema del "algoritmo para generar posiciones".

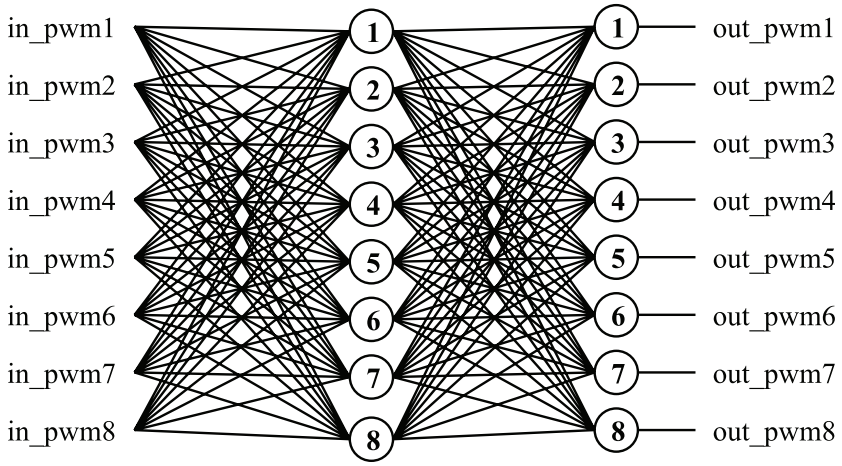


Figura 2.13: Red neuronal del "algoritmo para generar posiciones".

para determinar al final si los pesos han permitido al robot obtener un desplazamiento positivo y si la base del robot no ha tocado el suelo durante el funcionamiento. Si estas dos condiciones no se cumplen, los pesos son restituidos; en caso contrario, se los mantiene como los nuevos pesos de la red. Una vez que se realiza la prueba, se sigue con una nueva generación de pesos. El ciclo de funcionamiento del algoritmo de entrenamiento se muestra en la figura 2.14.

Además, el entrenamiento se ha divido en dos partes: un entrenamien-

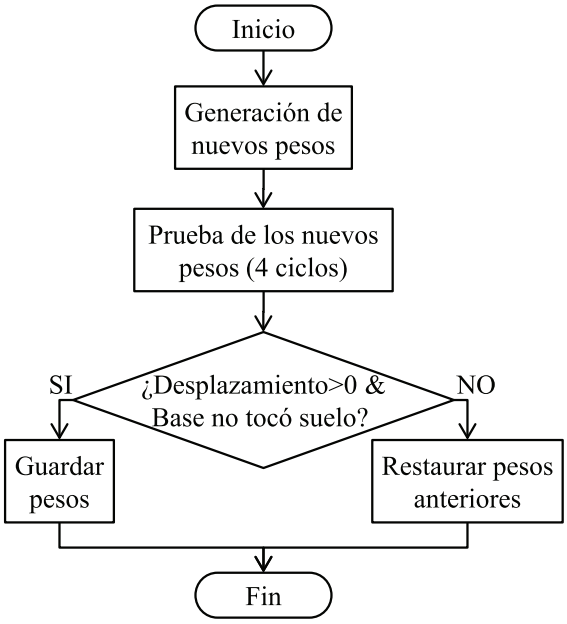


Figura 2.14: Entrenamiento del "algoritmo para generar posiciones".

to out-line y otro in-line. Primero, se hace un pre entrenamiento out-line en MATLAB® con una secuencia básica de desplazamiento conocida a priori mediante el algoritmo de entrenamiento backpropagation del apartado 2.1.1. El código se encuentra en el apéndice B. Y, luego, se completa el entrenamiento en el robot en cada ciclo de operación.

Hay que tener presente que la red neuronal representa una gran carga para el FPGA debido a que se requiere de bastantes operaciones de suma y multiplicación y de transferencia de registros. Este hecho se agrava si se piensa implementar el algoritmo de entrenamiento de la red. Por consiguiente, la red neuronal se hace con máquinas de estado para agrupar sus operaciones y reducir el costo del FPGA. La forma de agrupar la red neuronal y el algoritmo de entrenamiento es mediante el uso de variables auxiliares que almacenan los datos y luego los cargan en sus variables correspondientes según el estado en que se encuentra la máquina de estados. Con esto, en la red sólo se calculan 8 multiplicaciones en la capa oculta en vez de las 8×8. Se puede comparar ambos métodos en las figuras

2.15 y 2.16. Hay que resaltar que el cálculo es casi inmediato sin el uso de máquinas de estado; mientras que al usarlas el tiempo total de cálculo no debe sobrepasar el tiempo disponible para su ejecución.

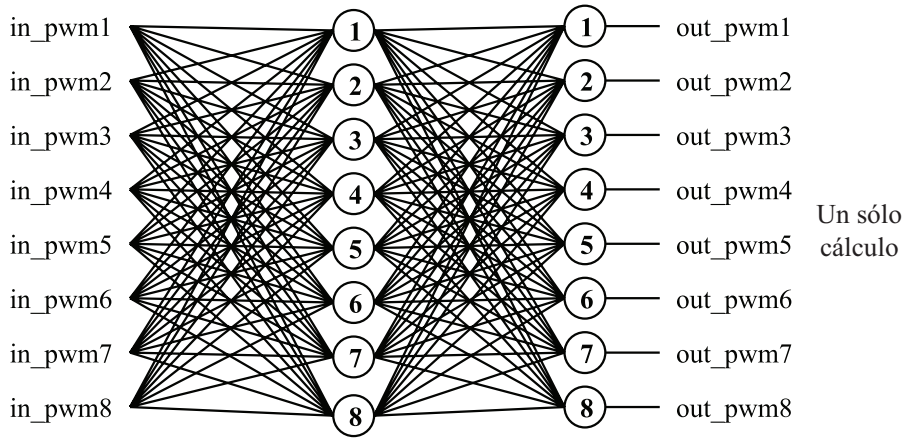


Figura 2.15: Cálculo de la red neuronal con programación combinacional.

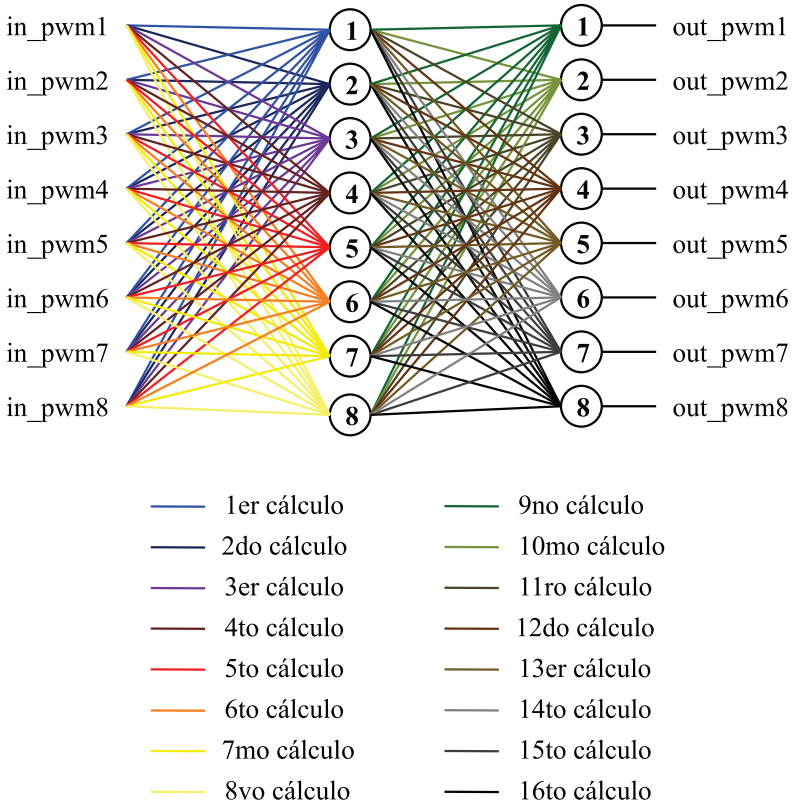


Figura 2.16: Cálculo de la red neuronal con máquinas de estado.

Además, en cada ciclo de trabajo, son ejecutados los bloques “Genera-

dor de Posiciones" y "Optimizador del Generador de Posiciones". Gracias al empleo de máquinas de estado y a la ejecución en paralelo del FPGA, esto no representa un problema con la disponibilidad de tiempos.

Otro punto importante es la representación de la parte decimal de los números en la red neuronal. Para hallarla, se ha escrito simplificadamente la ecuación de la red que relaciona la entrada con la salida, la cual se muestra en la ecuación 2.28, y se ha determinado que los pesos de la capa oculta tengan una precisión de 0.015625 (6 bits).

$$\begin{array}{cccccccccccc} & (& \text{in\_pwm} & \times & \text{Wo} & + & \text{bo} &) & \times & \text{Ws} & + & \text{bs} & = & \text{out\_pwm} \\ \textit{Bits} & & 7 & & 9 & & 16 & & & 16 & & 32 & & 32 \\ \frac{\textit{Entero}}{\textit{Decimal}} & & \frac{1}{5} & & \frac{2}{6} & & \frac{4}{11} & & & \frac{6}{9} & & \frac{11}{20} & & \frac{11}{20} \end{array} \tag{2.28}$$

La variable *in_pwm* es realmente un número entero de 6 bits pero se le ha agregado un bit para el signo positivo y se le ha divido entre 32 para normalizarla. De igual forma, la variable *out_pwm* es realmente un número entero de 6 bits pero como resultado de la red neuronal, sus partes entera y decimal se han incrementado. Además, se tiene que restaurar multiplicando por 32 para obtener su verdadero valor.

Adicionalmente, el desplazamiento del robot ha sido dividido en dos secuencias para poder combinar dos formas de movimiento; por lo que la red presenta dos juegos de pesos.

El diagrama de flujo completo del "algoritmo para generar posiciones" se encuentra en el apéndice C.

2.2.2. Algoritmo para Generar Desplazamientos

Este algoritmo de planeamiento de movimientos permite al robot RE1 tener una idea de cómo una variación en su posición genera una variación en su desplazamiento. Esta relación está representada por la ecuación 2.29.

$$\Delta\text{desplazamiento} = f\left(P, \Delta P\right) = g\left(P_t, P_{t-1}\right) \tag{2.29}$$

En consecuencia, debe existir una función que relacione estas magnitudes y un algoritmo que permita al robot RE1 decidir sobre cuál es la mejor variación en su posición, de forma tal que genere el mayor desplazamiento. Por lo que el problema se separa en dos partes. La primera consiste en hallar la función de la ecuación 2.29. Para lo cual, esta función debe ser de fácil implementación en el FPGA, ser lo más precisa y que se modifique para adaptarse a la estructura del robot. Un modelo mediante relaciones matemáticas requeriría de un cálculo de senos y cosenos y de tener que considerar las tolerancias del diseño mecánico [27]. Para evitar esto, se optó por emplear una red neuronal con su algoritmo de entrenamiento incluido en el robot ya que esta puede modificarse y adaptarse a las tolerancias y a las variaciones en la estructura del robot y, principalmente, puede hallar la ecuación 2.29. Además, una red neuronal es más sencilla de implementar en el FPGA que varias funciones trigonométricas requeridas por el modelo.

La segunda parte consiste de un algoritmo que maximice el desplazamiento a través de la red neuronal, contemplando a su vez ciertas restricciones como la variación máxima del ángulo de giro (variaciones grandes hacen que el robot RE1 cambie de una posición a otra muy rápido causando daños en él). Por estas razones, se ha empleado el algoritmo genético

utilizando por función de evaluación a la red neuronal. Además, los algoritmos genéticos son de fácil implementación en el FPGA ya que el FPGA es un entorno netamente digital (representación en bits, apartado 2.1.2) y porque el algoritmo genético no requiere hallar derivadas como en otros métodos (por ejemplo, el método de Newton [28]).

Habiendo definido cada parte de este algoritmo, se presenta el esquema en la figura 2.17.

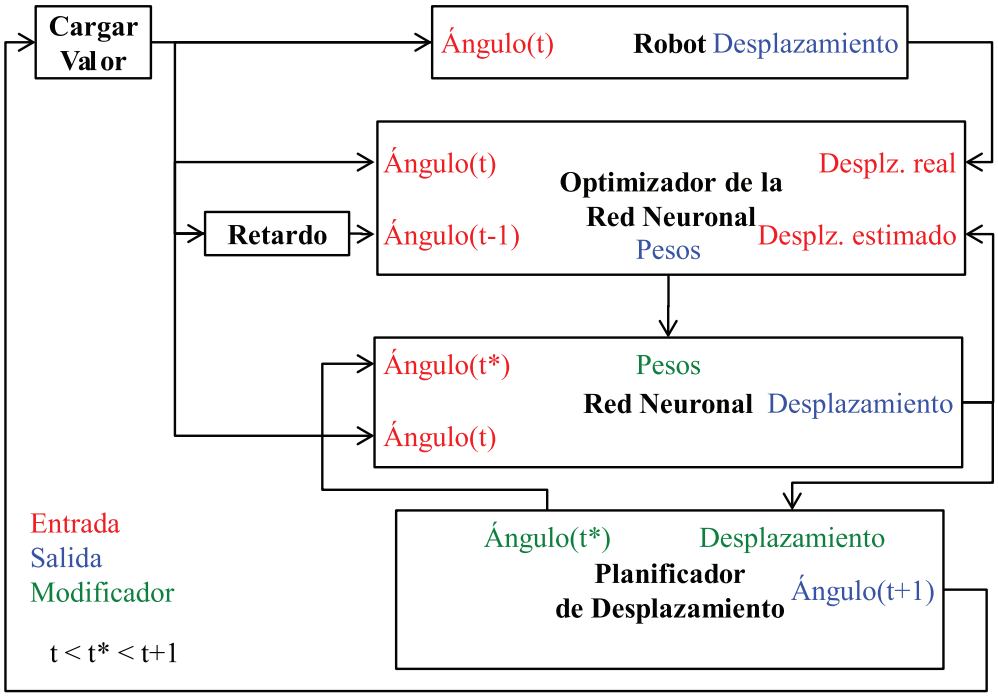


Figura 2.17: Esquema del "algoritmo para generar desplazamientos".

El bloque "Red Neuronal" contiene a la red que aprende la relación de la ecuación 2.29. Esta red neuronal tiene ocho neuronas en la capa oculta con función de transferencia "tansig" y una neurona en la capa de salida con función "purelin", como se muestra en la figura 2.18. El cálculo de cómo se ha determinado la estructura de la red neuronal se encuentra en el apéndice B. Además, sus pesos iniciales son aleatorios y mediante el bloque "Optimizador de la Red Neuronal", que contiene al algoritmo de entrenamiento backpropagation, son actualizados.

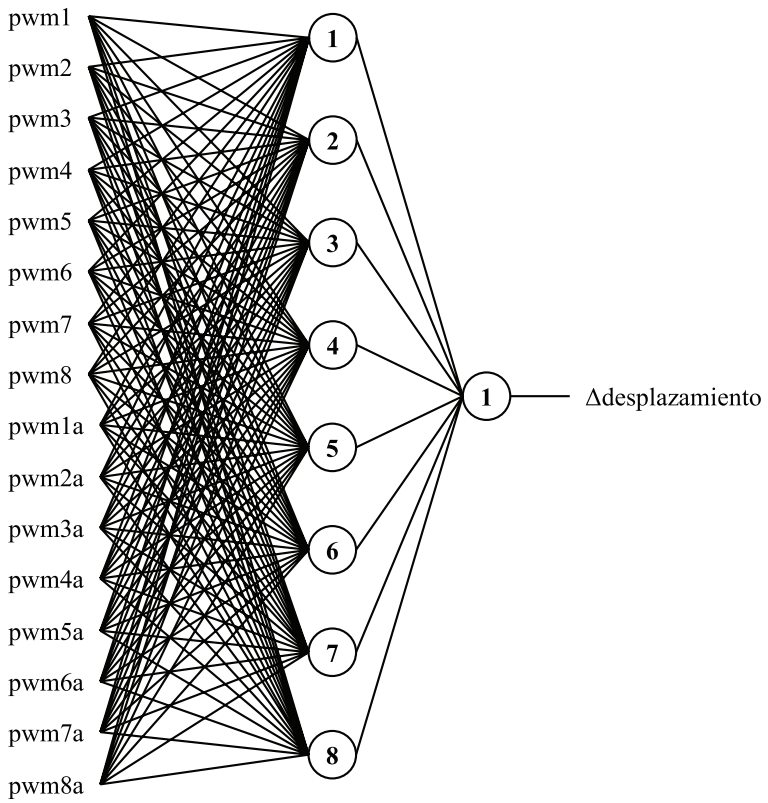


Figura 2.18: Red neuronal del "algoritmo para generar desplazamientos".

En el bloque "Planificador de Desplazamiento", se ha desarrollado el algoritmo genético. Los criterios para implementarlo, de acuerdo con el apartado 2.1.2, se exponen a continuación.

1. **Criterio de codificación**: No se requiere codificar la posición de los servomotores porque estas ya están en el sistema binario.
2. **Criterio de tratamiento de los individuos no factibles**: Son tomados como elementos no factibles las cadenas que generan variaciones en una articulación mayor a 6°. Estas cadenas son eliminadas en el proceso de reemplazo.
3. **Criterio de inicialización**: El algoritmo genético empieza con una población aleatoria y luego es usada la población final del proceso anterior.

4. **Criterio de parada**: El criterio que se emplea para terminar el algoritmo genético es la realización de dieciséis generaciones. El porqué del número de generaciones se halla en el apéndice B.
5. **Funciones de evaluación y aptitud**: La función de evaluación es la misma que la de aptitud, las cuales son la red neuronal.
6. **Operadores genéticos**: Los operadores genéticos empleados son el cruce y la mutación. Respecto al cruce, sólo dos individuos por generación pueden cruzarse; y a la mutación, sólo dos bits puede ser mutados en cada generación (véase apéndice B).
7. **Criterios de selección**: Se ha utilizado un método de selección por sorteo ya que requiere de pocas operaciones en el FPGA.
8. **Criterios de reemplazo**: El método de reemplazo empleado es el inmediato, es decir, los descendientes sustituyen a sus progenitores, salvo que sea un individuo no factible, en cuyo caso el progenitor sobrevive.
9. **Parámetros de funcionamiento**: El tamaño de la población es de cuatro cadenas. Esto es debido al costo en el FPGA ya que cada cadena tiene 48 bits para representar las ocho posiciones de las articulaciones del robot RE1.

Finalmente, el bloque "Retardo" guarda el valor anterior de la posición del robot, es decir, los ángulos de cada articulación; y el bloque "Cargar Valor" ordena al robot a tomar la posición hallada.

Por la mismas razones presentadas en el "algoritmo para generar posiciones", se ha implementado la red neuronal mediante máquinas de estado para reducir el costo del FPGA, aunque las operaciones se pueden realizar en un solo ciclo. Sólo hay que considerar que el tiempo total en calcular

no sobrepase el tiempo disponible para su ejecución.

Además, se han simplificado las ecuaciones para el entrenamiento con el algoritmo backpropagation ya que existen multiplicaciones por 1 que simplemente utilizan más recursos del FPGA. En el cuadro 2.1 está un paralelo que muestra la transformación de las ecuaciones. Nótese que el vector $W^{(2)} = [W_s\ b_s]^T$ y que $W_i^{(1)} = [W_o\ b_o]^T$, donde b_s y b_o son los valores umbrales negativos.

Original	**Reducida**
Capa de salida	
$W^{(2)} \Leftarrow W^{(2)} + c^{(2)}\delta^{(2)}X^{(1)}$	$W_s \Leftarrow W_s + c\delta_s X_s$ $b_s \Leftarrow b_s + c\delta_s$
Capa oculta	
$W_i^{(1)} \Leftarrow W_i^{(1)} + c_i^{(1)}\delta_i^{(1)}X^{(0)}$	$W_o \Leftarrow W_o + c\delta_o X_o$ $b_o \Leftarrow b_o + c\delta_o$

Cuadro 2.1: Ecuaciones del algoritmo backpropagation.

Otro aspecto importante es la lógica para los números con decimales; la cual se muestra en la ecuación 2.30 para la red neuronal y en la ecuación 2.31 para el algoritmo backpropagation. Obsérvese que en algunos casos la relación "Entero/Decimal" está afectada por una suma o resta, que representa los bits que se van a añadir (+) o remover (-) en la representación del número para equilibrar la expresión matemática.

$$\begin{array}{lcccccc} & \text{pwm} & \times & \text{W}_\text{O} & + & \text{b}_\text{O} & = & s \\ Bits & 7 & & 9 & & 16 & & 16 \\ \frac{Entero}{Decimal} & \frac{1}{5} & & \frac{3}{5} & & \frac{5}{10} & & \frac{5-1}{10-5} \end{array} \tag{2.30}$$

$$\begin{array}{lcccccc} & f_{tansig}(s) & \times & \text{W}_\text{S} & + & \text{b}_\text{S} & = & \Delta\text{desplazamiento} \\ Bits & 7 & & 14 & & 20 & & 20 \\ \frac{Entero}{Decimal} & \frac{0}{5} & & \frac{7}{6} & & \frac{8}{11} & & \frac{8}{11} \end{array}$$

Hay que señalar que la función “tansig” empleada recibe un número de 10 bits con 5 bits decimales y devuelve uno de 6 bits con 5 bits decimales (véase apartado 4.3).

	(	Δdist	−	Δdiste	=	δ_S	)	×	W_S	=	δ_O
Bits		8		8		8			14		22
$\frac{Entero}{Decimal}$		$\frac{7}{0}$		$\frac{7}{0}$		$\frac{7}{0}$			$\frac{7}{6}$		$\frac{15}{6}$

	b_S	⇐	b_S	+	c	δ_S
Bits	20		20		−	8
$\frac{Entero}{Decimal}$	$\frac{8}{11}$		$\frac{8}{11}$		$\frac{0}{3}$	$\frac{7}{0+8}$

	W_S	⇐	W_S	+	c	δ_S	X_S
Bits	14		14		−	8	6
$\frac{Entero}{Decimal}$	$\frac{7}{6}$		$\frac{7}{6}$		$\frac{0}{1}$	$\frac{7}{0}$	$\frac{0}{5}$

(2.31)

	b_O	⇐	b_O	+	c	δ_O
Bits	16		16		−	22
$\frac{Entero}{Decimal}$	$\frac{5}{10}$		$\frac{5}{10}$		$\frac{0}{3}$	$\frac{15}{6+1}$

	W_O	⇐	W_O	+	c	δ_O	X_O
Bits	9		9		−	22	7
$\frac{Entero}{Decimal}$	$\frac{3}{5}$		$\frac{3}{5}$		$\frac{0}{1}$	$\frac{15}{6-4}$	$\frac{1}{5-3}$

Finalmente, el diagrama de flujo completo del “algoritmo para generar desplazamientos” se encuentra en el apéndice C.

CAPÍTULO 3
ALGORITMOS DE SENSORES Y ACTUADORES

3.1. SENSORES

Los sensores son vitales para la elaboración de los algoritmos de planeamiento de movimientos debido a que mantienen informado al robot RE1 sobre su posición, ya sea externa o interna. Con referencia a esto, los sensores se clasifican en dos grupos. El primer grupo describe a los sensores que le informan sobre su posición respecto a un sistema de referencia; dentro de ellos se incluyen los sensores de distancia, inclinación, orientación y proximidad. El segundo grupo de sensores le indican la posición de una parte de él respecto a otra; perteneciendo a este sólo el sensor de ángulo.

Se ha enfatizado en esta diferencia porque son los sensores del primer grupo los que funcionan como parámetros de performance que regulan el proceso de optimización del comportamiento del robot RE1; y los del segundo grupo son los que se busca controlar.

3.1.1. Sensor de Ángulo

El sensor de ángulo permite tener una realimentación de la posición (ángulo) de cada articulación del robot en cada ciclo de trabajo.

El sensor empleado es el potenciómetro implementado en el servomotor MG995 (véase figura 3.1), del cual se ha tomado la lectura del voltaje, que es proporcional a la posición y se ha digitalizado mediante un ADC.

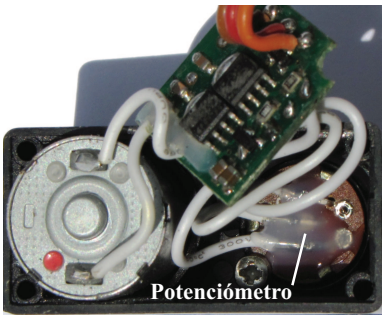


Figura 3.1: Foto: Sensor de ángulo.

Entonces, para evitar un mal funcionamiento del servomotor debido a una alteración de la señal en el potenciómetro causado por introducción de ruido o por una lectura inadecuada, se ha realizado el siguiente tratamiento: Primero, se ha aislado el circuito del servomotor del filtro y del ADC mediante un seguidor emisivo (configurado con un OPAMP). Además, se ha conectado un condensador al ingreso del seguidor para reducir el ruido. Segundo, debido a que la señal es de baja frecuencia, se ha filtrado a través de un filtro "pasa bajo" de frecuencia de corte de 10Hz. Se demuestra que la señal es de baja frecuencia debido a que esta varía en la misma proporción que gira la articulación (velocidad máxima 60°/0.2s, véase anexo E), la cual gira a una velocidad angular máxima de 5.24rad/s o una frecuencia máxima de 0.83Hz. Y tercero, se ha digitalizado mediante un ADC implementado en el microcontrolador PIC16F877A (abreviado a PIC) .

Por otro lado, la programación del sensor consiste en digitalizar la señal de cada potenciómetro mediante el PIC, que convierte ocho señales analógicas secuencialmente, y enviársela al FPGA. Para evitar utilizar una gran cantidad de pines tanto en el PIC como en el FPGA, la comunicación se ha dividido en dos buses: uno de datos de 8 bits, por el cual se envía el valor de la conversión, y otro de dirección de 3 bits en código gray, por el cual se indica a quién pertenece dicha conversión. Para esto, el PIC establece primero el bus de datos de manera tal que el FPGA lo lea cada

vez que haya un cambio en el bus de dirección. Además, con el código gray, sólo un bit cambia lo que evita la aparición de estados indeterminados. En consecuencia, con las dos consideraciones, se asegura la adquisición correcta del valor de cada posición.

El diagrama de flujo simplificado del algoritmo implementado en el FPGA se muestra en la figura 3.2. Por otro lado, el diagrama de flujo completo y el código del FPGA se encuentran en el apéndice D y del PIC, en el apéndice E.

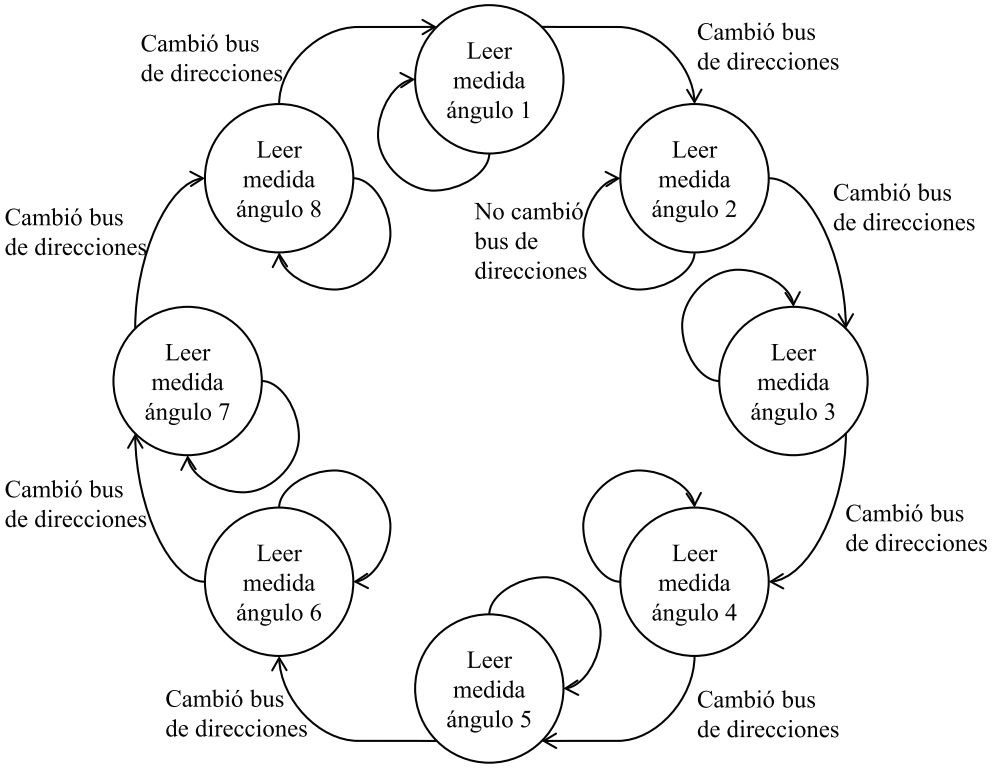


Figura 3.2: Diagrama de flujo simplificado del sensor de ángulo.

3.1.2. Sensor de Distancia

El sensor de distancia implementado en el robot RE1 es el sensor ultrasónico PING))), mostrado en la figura 3.3, que mide la distancia que avanza el robot.

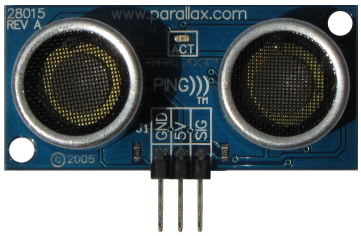


Figura 3.3: Foto: Sensor de distancia.

El funcionamiento del sensor consiste en generar un pulso de salida que tenga un ancho de igual tiempo que lo que una ráfaga de sonido de 40KHz emitida por él mismo se demora en viajar de ida y regreso desde él hacia un obstáculo [29]. Para que ocurra esto, se debe entregar primero un pulso de $5\mu s$ y esperar como mínimo $200\mu s$, tal como está representado en el diagrama de tiempos de la figura 3.4 (véase anexo A).

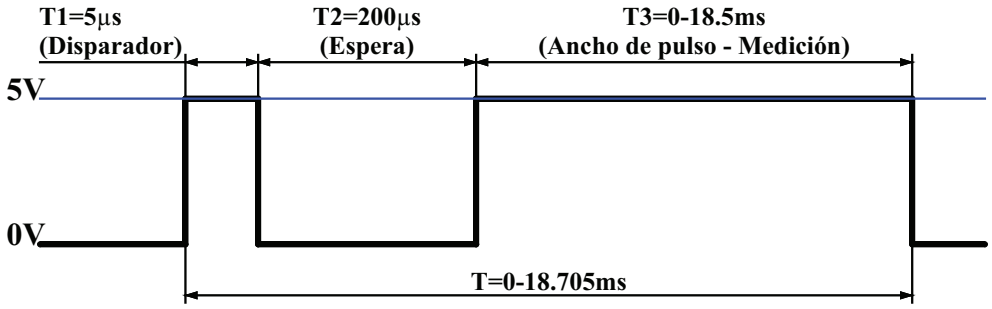


Figura 3.4: Diagrama de tiempos del sensor de distancia.

Hay que considerar para el desarrollo del algoritmo de operación que por seguridad el pulso para iniciar la medición y el pulso con la lectura, que están asociados al mismo pin, han sido separados correspondiéndoles un pin a cada uno (pin Dout y Din respectivamente) mediante el microcontrolador PIC16F84A. Los diagramas de flujo y el código para el PIC16F84A que independiza ambas señales se encuentran en el apéndice E.

En la figura 3.5, se muestra el algoritmo programado en el FPGA que permite operar el sensor. El algoritmo empieza enviando un flanco por el

pin Dout del FPGA al PIC16F84A que al recibirlo genera el pulso de $5\mu s$ al sensor. Luego de un rato, el sensor activa su salida con lo que empieza la medición y el PIC16F84A activa el pin Din del FPGA. Finalmente, cuando se desactiva la salida del sensor (y, por ende, el pin Din del FPGA), termina la medición y se determina la distancia que es almacenada en una variable de 8 bits. El diagrama de flujo completo y el código del FPGA se encuentran en el apéndice D.

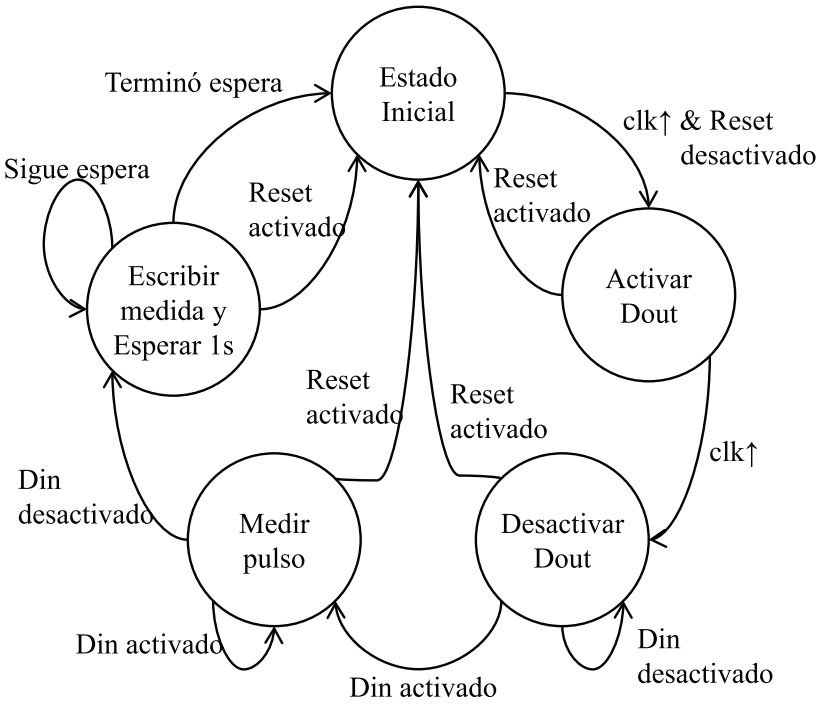


Figura 3.5: Diagrama de flujo simplificado del sensor de distancia.

Respecto a la información técnica del sensor de distancia, esta se localiza en el anexo A.

3.1.3. Sensor de Inclinación

La inclinación del robot es una variable que permite compensar la medición de la distancia ante ráfagas de sonido no normales (perpendiculares) a la pared de referencia.

El sensor implementado en el robot RE1 es el acelerómetro MEMSIC MXD2125 de la figura 3.6, el cual mide la inclinación a través de la gravedad [29].

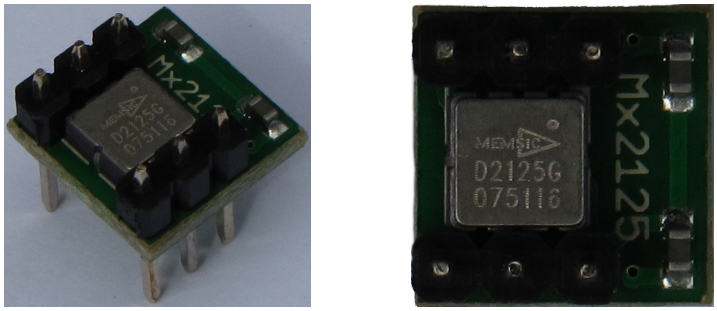


Figura 3.6: Foto: Sensor de inclinación.

Este sensor tiene una cámara de gas con un calentador en el centro y cuatro sensores de temperatura alrededor de los cuatro extremos de un plano cartesiano. Si el sensor se mantiene estable, lo único que mide es la gravedad. Cuando uno mantiene el nivel del acelerómetro, el gas caliente se eleva en el centro superior de la cámara y todos los sensores miden la misma temperatura. Dependiendo de como se incline el acelerómetro, el aire caliente se almacenará cerca a uno o dos sensores de temperatura. Por comparación de las temperaturas, la aceleración estática y la dinámica son medidas y convertidas a señales de PWM [29]. Hay que tener presente que el sensor no da como respuesta el ángulo de inclinación sino la proyección de la gravedad sobre su superficie.

Cabe resaltar que en el funcionamiento del robot, la medición de la inclinación no se verá afectada por la aceleración ya que esta es baja.

Respecto a la medición, este sensor genera automáticamente dos señales de PWM con el valor de la inclinación en “X” y en “Y” (véase figura 3.7) [29]. Estas dos señales de PWM tienen un período de 10 ms (véase anexo B). Ambas señales son leídas simultáneamente por el FPGA

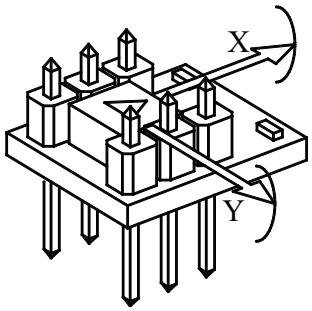
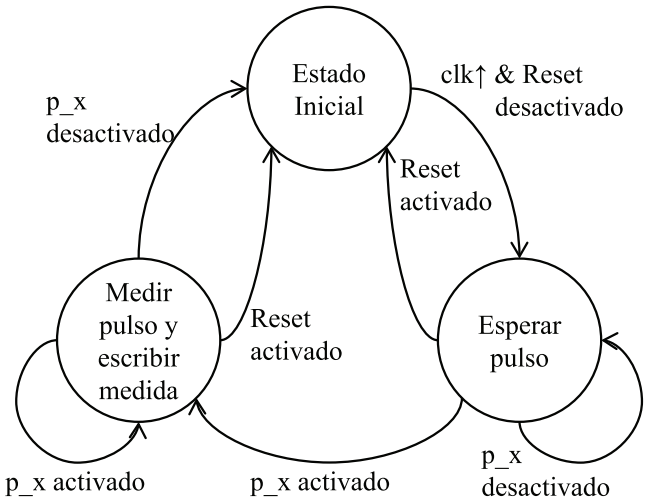
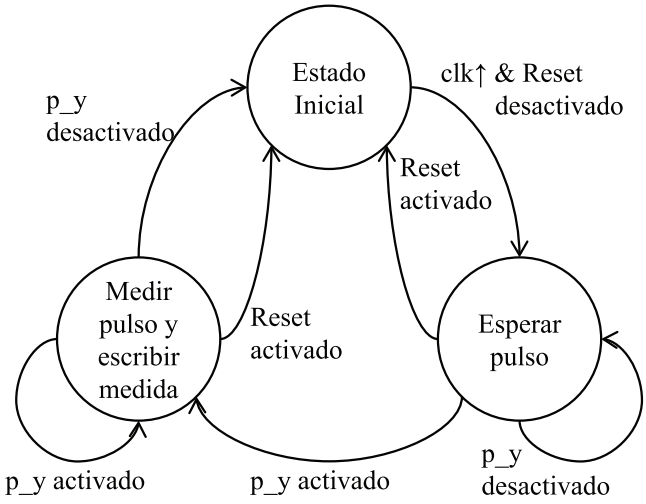


Figura 3.7: Ejes del sensor de inclinación.



a) Medición en X



b) Medición en Y

Figura 3.8: Diagrama de flujo simplificado del sensor de inclinación.

y almacenadas en dos variables de 8 bits al determinarse sus tiempos de activación. El diagrama de flujo simplificado se muestra en la figura 3.8. El diagrama de flujo completo y el código se encuentran en el apéndice D y las especificaciones técnicas en el anexo B.

3.1.4. Sensor de Orientación

Como la distancia es medida entre el sensor de distancia y una pared en la dirección del sensor, la variación de esta no siempre corresponde a un desplazamiento del robot ya que esta también puede aumentar sólo con rotar este. Entonces, gracias a la orientación, se puede compensar esta medición.

El sensor de orientación instalado en el robot RE1 es el Hitachi HM55B de la figura 3.9. Su principio de funcionamiento es semejante al de una brújula: calcula la orientación a través del campo magnético de la tierra. Bajo un sistema cartesiano interno paralelo a la superficie de este como se observa en la figura 3.10, el sensor determina la componente positiva en "X" y la componente negativa en "Y" del campo, con lo que se forma un vector cuya dirección representa la orientación [29].

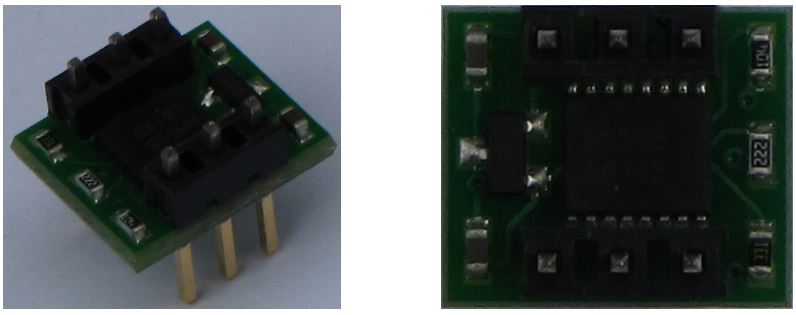


Figura 3.9: Foto: Sensor de orientación.

Para operar este sensor, hay que enviar una serie de comandos que están mostrados en la figura 3.11 (véase anexo C). Primero, se debe resetear

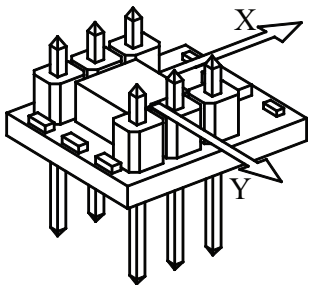
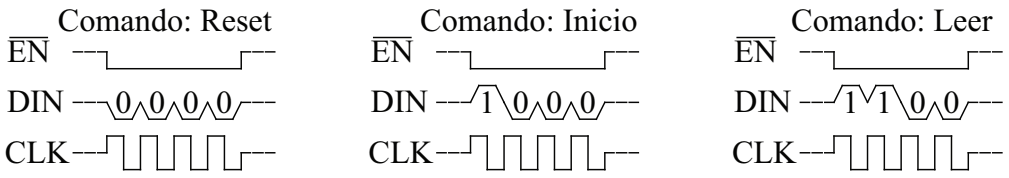
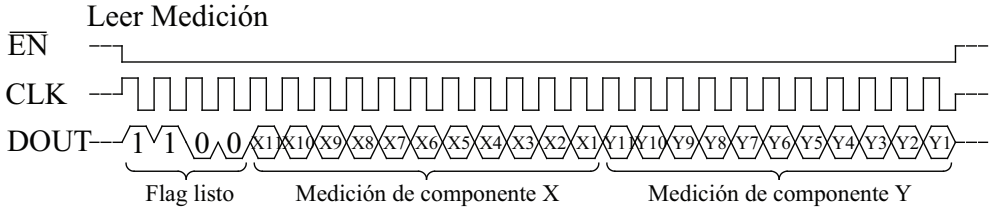


Figura 3.10: Ejes del sensor de orientación.

al sensor y luego empezar la medición. Después, para obtener la medición existen dos formas: una es preguntar consecutivamente si la medición ya terminó y la otra es esperar el tiempo que se demora en medir. En la primera, se envía el comando leer y se recibe el flag hasta que este indique que la medición está lista. En la segunda, que fue la empleada por su facilidad, se espera entre 30 y 40 ms (se esperó 640 ms debido a que no se requiere tanta velocidad), se envía el comando leer y se obtiene el flag para luego recibir la medición en "X" y en "Y". El diagrama de flujo simplificado se muestra en la figura 3.12, el completo con el código se encuentra en el apéndice D y las especificaciones técnicas en el anexo C.



a) Envío de comandos.



b) Recepción de datos.

Figura 3.11: Diagrama de tiempos del sensor de orientación.

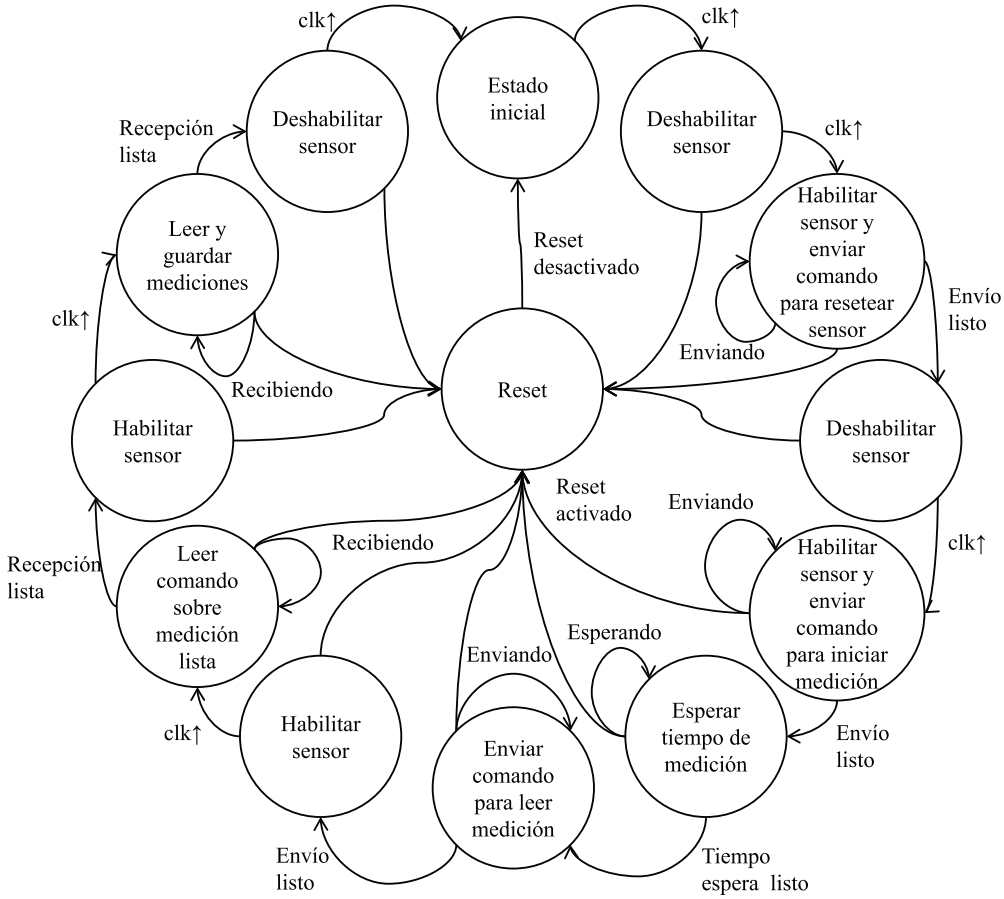


Figura 3.12: Diagrama de flujo simplificado del sensor de orientación.

3.1.5. Sensor de Proximidad

El sensor de proximidad se emplea para evitar que durante el entrenamiento la base del robot roce el suelo. Para este fin, está implementado un interruptor con el CNY70 (véase anexo D), que está conformado por un LED emisor y uno receptor de luz infrarroja (véase figura 3.13). Este funciona haciendo reflejar un haz de luz infrarrojo sobre una superficie a menos de 15 mm. Si la superficie es clara, se reflejará mayor cantidad del haz activando la base del transistor receptor; en cambio, si la superficie es oscura o muy distante, no habrá activación.

Para el funcionamiento del robot RE1, la señal de salida de cada interruptor es procesada directamente por el FPGA tomando su activación

y desactivación con un “1” y “0” digital respectivamente.

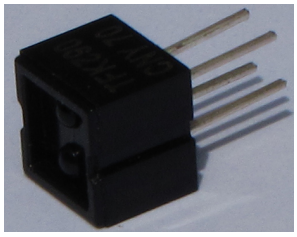


Figura 3.13: Foto: Sensor de proximidad.

3.2. ACTUADORES

Los únicos actuadores implementados en el robot RE1 son los servomotores TowerPro MG995 (véase figura 3.14) que están instalados en cada articulación.

Figura 3.14: Foto: Servomotor.

Estos servomotores tienen un sistema de control interno cuya función es controlar el ángulo de giro. La referencia o posición deseada es entregada y codificada en el tiempo de activación de una señal PWM, que tiene un periodo de 20 ms y en la cual un pulso de 0.7 ms de activación equivale a la posición de 0° del servomotor y uno de 2.7 ms equivale a la de 180°. Las posiciones intermedias tienen un tiempo de activación proporcional a este rango.

Para su funcionamiento, se implementó en el FPGA un generador de señales PWM que tiene por entrada a una variable llamada v_pwm de 6 bits que regula el ancho de la señal y a una señal de reloj (clock) de 10 μs como base de tiempo. Además, existe un contador de 11 bits cuyo valor máximo es el período y que

aumenta en uno cada vez que recibe un flanco positivo del reloj y regresa a 0 al desbordarse.

De lo descrito, se demuestra que el periodo de la señal PWM es de 20.48 ms, la precisión es de 10 μs y los 6 bits generan una variación de 55°.

$$2^{11} \times 10\mu\text{s} = 20.48\text{ms}$$

Por otro lado, la variable *v_pwm* es sumada por una constante que determina la posición mínima de la articulación y cuyo valor depende de cada servomotor. Esta constante fue hallada experimentalmente bajo la premisa de que el mismo valor de *v_pwm* debe generar la misma posición en las articulaciones, teniendo una posición mínima las articulaciones superiores distinta de las inferiores. Además, los valores de *v_pwm* de cuatro articulaciones han sido negados (operación lógica not) porque estos fueron colocados al revés para lograr la simetría en el montaje, y por lo tanto, un aumento en el *v_pwm*, no genera un aumento en la misma dirección de la posición de la articulación; en cambio, al negarlo si lo hace. Los valores de las constantes y la negación para cada servomotor se muestran en el cuadro 3.1 y el resumen del cálculo realizado en la ecuación 3.1.

$$\text{tiempo activo (bits)} = \pm v\_pwm + \text{constante} \tag{3.1}$$

Servomotor	**Negado**	**Constante** (binario)
1	No	00010010001
2	Si	00011000011
3	Si	00010010001
4	No	00001011010
5	No	00010010001
6	Si	00011000011
7	Si	00010010001
8	No	00001011010

Cuadro 3.1: Valores para generar la señal PWM para cada servomotor.

Finalmente, esta señal corregida es constantemente comparada con el valor del contador. Cuando el contador es menor, la salida estará activa (generación del pulso); en cambio, cuando sea mayor, se desactivará. De esta comparación, se visualiza que la modificación de la señal *v_pwm* es análoga al tiempo de activación o ancho del pulso. El diagrama de flujo simplificado se esquematiza en la figura 3.15 y el diagrama de flujo completo con el código en VHDL implementado en el FPGA se encuentra en el apéndice F.

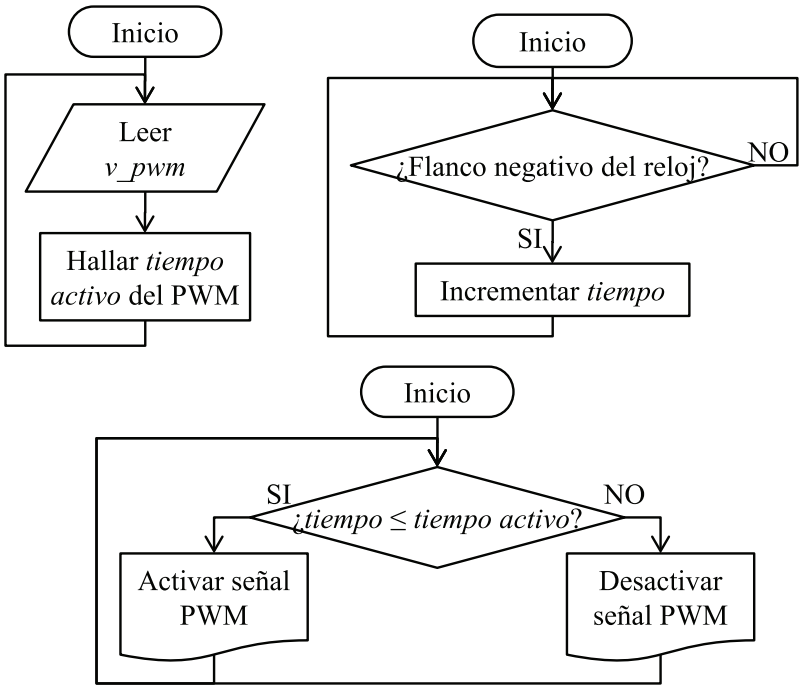


Figura 3.15: Diagrama de flujo del funcionamiento del servomotor.

Cabe resaltar que la variable *tiempo* es de 11 bits por lo que en un overflow se reinicia a "000 0000 0000".

Respecto a la información técnica del servomotor MG995, esta se localiza en el anexo E.

CAPÍTULO 4

OPERACIONES COMPLEMENTARIAS

4.1. OPERACIONES ARITMÉTICAS

Para el desarrollo completo y adecuado de los algoritmos del robot RE1 se debe considerar las operaciones matemáticas disponibles. Por ejemplo, para poder implementar las redes neuronales, es necesario realizar sumas y multiplicaciones (ecuaciones 2.12 y 2.18), y también, dependiendo de la función de transferencia, divisiones y potencias (ecuaciones 2.8 y 2.9). Una opción es emplear la aritmética del punto flotante, pero esta requiere de la implementación específica de cada operación y de un alto costo en área del FPGA. A cada variable le correspondería un número igual de bits (según la norma IEEE-754, un formato de simple precisión emplea 32 bits y uno de doble precisión 64 bits [30]), con lo que se desperdiciaría recursos del FPGA en variables de baja precisión. Otra alternativa es utilizar la aritmética de los números enteros con las operaciones de la suma (incluyendo la resta) y la multiplicación que ya están implementadas en el Xilinx ISE y que son menos pesadas. Sin mebargo, esta opción no ofrece una mayor precisión.

Comparando ambas opciones se ha escogido la segunda porque, para la investigación, hace un uso más eficiente de los recursos del FPGA (poco costo en área y menor número de bits desperdiciados) y porque no se requiere de tanta precisión ya que las variables de los sensores y actuadores son enteras.

Hay que resaltar que se ha ampliado esta alternativa para obtener las demás

operaciones que son necesarias en los algoritmos y que se estará trabajando en un sistema de base 2 debido a que el FPGA es un sistema netamente digital.

En consecuencia, el primer arreglo que se ha hecho es considerar a cada número como una fracción. Luego, se ha establecido que el denominador de cada número sea un número entero potencia de 2, debido a que estamos trabajando en un sistema binario y a que así es posible separar los bits que representan la parte entera de la decimal. El valor del denominador, o su equivalente en número de bits de la parte decimal, dependerá de la precisión que se busque y se puede calcular según la ecuación 4.1.

$$\text{precisión} > \frac{1}{\text{denominador}} \text{ o precisión} > \frac{1}{2^{\text{bits parte decimal}}} \tag{4.1}$$

Como ejemplo, si se quisiera representar el número 5.2 con una precisión de 0.1 necesitaría 4 bits en la parte decimal porque este debe ser mayor a $-log_2(0.1) = 3.3$. Entonces, el número equivalente más cercano es 5.1875 ya que

$$\frac{83}{16} = 5.1875 < 5.2 < \frac{84}{16} = 5.25$$

En sí, esta idea del punto es sólo imaginaria ya que dentro del grupo de bits no hay información que diga que es entero o decimal. Del ejemplo anterior, el número 5.2 fue representado por 83 = 01010011 pero no se indica que los 4 bits menos significativos son la parte decimal. Más aún, esta representación también puede contener a los siguientes números: 83, 41.5, 20.75, 10.375, **5.1875**, etc. A pesar de esto, esta representación simplifica los cálculos y se tiene que tener siempre presente lo que representa cada bit.

Con respecto a las operaciones, se ha hecho lo siguiente:

4.1.1. Suma

Para poder sumar números con decimales hay que tener en cuenta el mismo principio que sumar fracciones, es decir, sólo se pueden sumar directamente números con la misma cantidad de bits en la parte decimal. En consecuencia, el resultado también tiene la misma representación.

Cabe resaltar que en el FPGA es factible sumar directamente números con distinta parte decimal porque esta idea es sólo imaginaria. En sí, lo que se está sumando son números enteros. Sin embargo, el resultado no va a tener significado.

4.1.2. Multiplicación

Al igual que la suma, la multiplicación se realiza teniendo en cuenta la multiplicación de fracciones. En este caso, es posible multiplicar dos números con diferente cantidad de bits en la parte decimal; siendo el número de bits del resultado igual a la suma de bits de los factores, tanto en tamaño total como en tamaño de la parte decimal.

4.1.3. División

Esta operación está implementada en el Xilinx ISE para números enteros con el dividendo constante y requiere un costo significativo en el FPGA. Para simplificar su uso, se ha preferido hacer divisiones en potencias de 2 ya que en el sistema binario esta operación es simplemente remover los bits menos significativos.

$$10/2 = 5$$

$$(1010)_2/2 = (101)_2$$

4.1.4. Otras operaciones

Para realizar otras operaciones se ha optado por dos métodos, según la facilidad de implementación. El primer método es aproximar la operación por medio de arreglos de sumas y multiplicaciones; y el segundo método es hacer uso de tablas de conversión.

4.2. CÁLCULO DEL ÁNGULO DE ORIENTACIÓN

La orientación del robot es hallada con el sensor HM55B, que mide las componentes del vector de campo magnético [29], (llamadas v_x y v_y) en donde está el robot. Sin embargo, la información que se necesita es la dirección del vector que se traduce en su ángulo, el cual es el ángulo de orientación.

La expresión matemática que permite hallar el ángulo de orientación es la ecuación 4.2. Siendo la componente en "y" el cateto opuesto y en "x" el adyacente de un triángulo rectángulo, el ángulo formado se calcula mediante la función de arco tangente. Además, se ha escalado el resultado ya que la función arco tangente varía entre -π y π y uno en la programación necesita un valor entre 0 y 255 (valor de 8 bits).

$$\text{orientación} = \arctan\left(\frac{v\_y}{v\_x}\right)\frac{255}{2\pi} + \frac{255}{2} \tag{4.2}$$

Debido a que las únicas operaciones disponibles son la suma o la resta y la multiplicación, se halló una aproximación a esta función para así lograr su implementación.

La ecuación aproximada se halló a partir de la relación trigonométrica que relaciona en un vector cada componente suya con su magnitud y dirección. Siendo el campo magnético terrestre un vector de magnitud B, las componentes serían iguales a la ecuación 4.3. Estas dos componentes son funciones del ángulo de orientación y mediante ellas se debe hallar este mismo ángulo; entonces, el objetivo es relacionarlas de manera tal que se consiga una función identidad. Si

analizamos y dividimos a cada 90° a las funciones seno y coseno, se observa que cada una se puede arreglar de manera tal que se aproxime a una recta. Asimismo, se puede hallar una mejor recta si promediamos ambos resultados. En el FPGA, este valor va a estar representado por un número de 8bits (0-255) por lo que una vez obtenida la aproximación, tiene que multiplicarse por 255/360 para su representación.

$$v\_x = B.\cos(\theta) \text{ y } v\_y = B.\text{sen}(\theta) \qquad \text{con } \theta : \text{orientación} \tag{4.3}$$

La función aproximada obtenida está expresada en la ecuación 4.4 y se muestra en la figura 4.1. El error de la aproximación fue del 1.84 % demostrándose poca variación con la función lineal ideal.

$$\text{orientación} = \frac{255}{360}\begin{cases} 45(-v\_x + v\_y)/B + 45 & ,si \text{ v\_x} \geq 0 \wedge \text{v\_y} \geq 0 \\ 45(-v\_x - v\_y)/B + 135 & ,si \text{ v\_x} < 0 \wedge \text{v\_y} \geq 0 \\ 45(v\_x - v\_y)/B + 225 & ,si \text{ v\_x} < 0 \wedge \text{v\_y} < 0 \\ 45(v\_x + v\_y)/B + 315 & ,si \text{ v\_x} \geq 0 \wedge \text{v\_y} < 0 \end{cases} \tag{4.4}$$

Para determinar si la aproximación era adecuada, se implementó y probó en el robot; para lo cual, se determinó experimentalmente que la magnitud del campo magnético B en la ecuación 4.4 es 30. El resultado es la ecuación 4.5 y su evaluación se muestra en la figura 4.2 donde el error fue de 2.56 % para la función exacta y 4.15 % para la aproximación.

$$\text{orientación} = \begin{cases} (-v\_x + v\_y) + 32 & ,si \text{ v\_x} \geq 0 \wedge \text{v\_y} \geq 0 \\ (-v\_x - v\_y) + 96 & ,si \text{ v\_x} < 0 \wedge \text{v\_y} \geq 0 \\ (v\_x - v\_y) + 159 & ,si \text{ v\_x} < 0 \wedge \text{v\_y} < 0 \\ (v\_x + v\_y) + 223 & ,si \text{ v\_x} \geq 0 \wedge \text{v\_y} < 0 \end{cases} \tag{4.5}$$

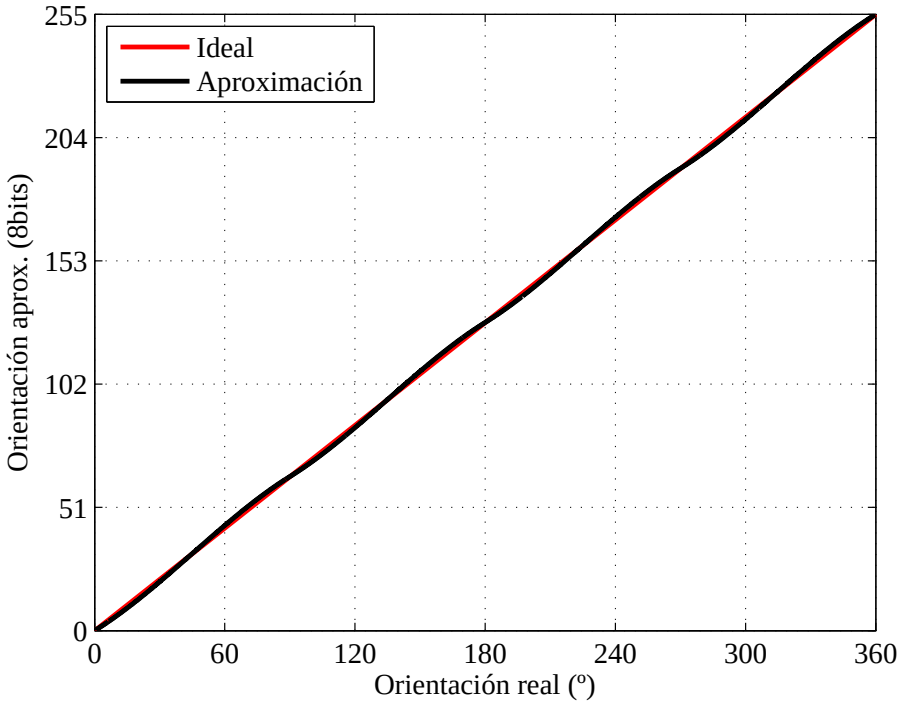


Figura 4.1: Función aproximada para hallar la orientación.

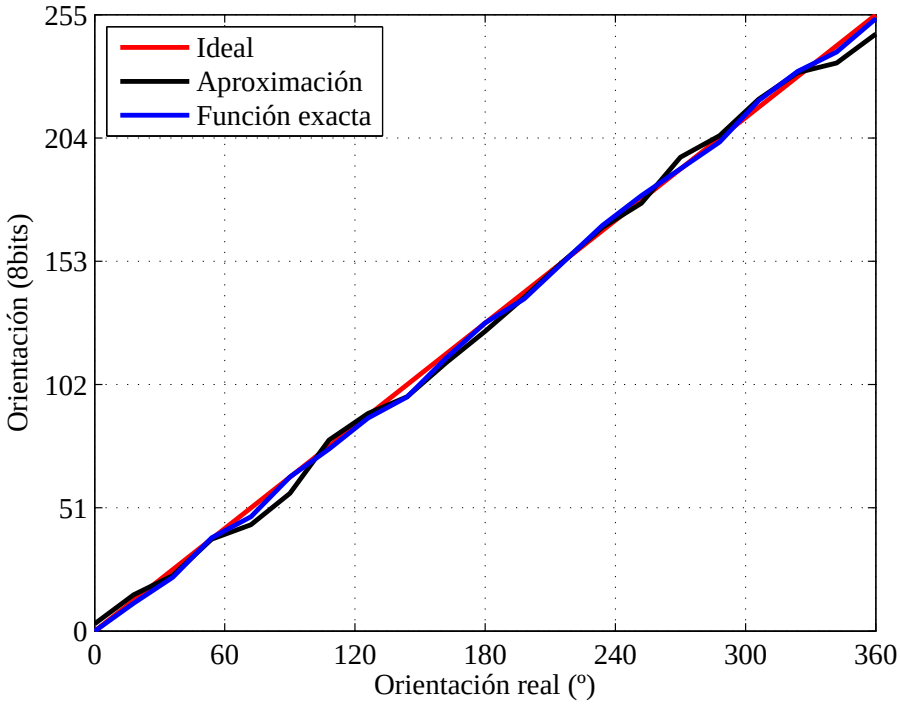


Figura 4.2: Resultado experimental al hallar el ángulo de orientación.

4.3. APROXIMACIÓN DE LA FUNCIÓN TANSIG

La red neuronal del "algoritmo para generar desplazamientos" (apartado 2.2.2) implementada en el robot utiliza la función de transferencia "tansig", que está definida por la ecuación 2.9.

Para calcularla, se puede emplear la aritmética de los números reales o las tablas de conversión; prefiriéndose emplear la segunda opción por su facilidad de implementación y por el reducido costo en área del FPGA.

Para su construcción, se requieren tres datos. El primero es el número de bits de la parte decimal de la salida de la tabla. Nótese que la función "tansig" varía entre -1 y 1, por lo que sólo se necesita un bit adicional para el signo. Y los otros dos datos son el tamaño total y el de la parte decimal de la entrada en bits.

Finalmente, la tabla implementada tiene 10 bits totales y 5 bits decimales en la entrada y 6 bits en la salida. El resultado se muestra en la figura 4.3 y el error promedio obtenido fue de 1.53 %.

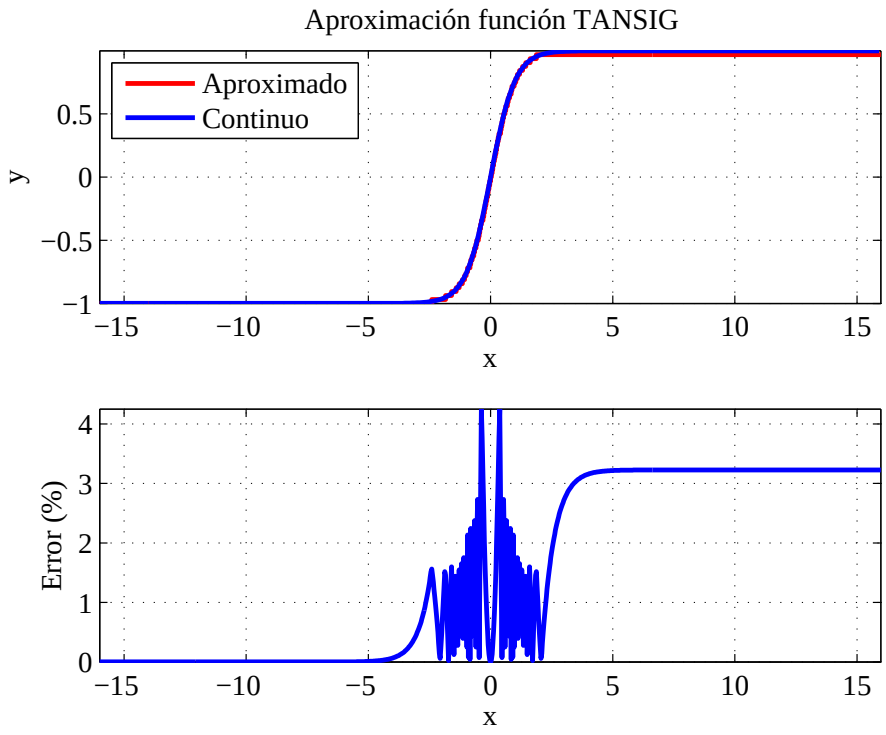


Figura 4.3: Tabla de conversión de la función "tansig" implementada.

4.4. COMPENSACIÓN DE LA MEDICIÓN DE LA DISTANCIA

La medición de la distancia es tomada en la dirección del sensor de distancia y a partir de este hasta una pared que se encuentra delante del robot RE1, la cual es el sistema de referencia. En consecuencia, la medición dependerá de la dirección del sensor y de la separación entre el robot y la pared. Por lo tanto, esta medición no siempre indica un desplazamiento ya que si el robot girara o se inclinara sin desplazarse, el sensor también girará con lo cual cambiará su dirección y, en consecuencia, la medición aumentará.

Durante su funcionamiento, el robot puede girar libremente respecto a los tres ejes de un sistema cartesiano. Si tomamos la dirección del desplazamiento como el eje "X", la normal al suelo como el eje "Z" y la perpendicular común como el eje "Y"; entonces, la componente en "Y" del sensor de inclinación mide el giro en "X", la componente en "X", el giro en "Y" y el ángulo de orientación representa el giro en "Z". En consecuencia, la combinación de estos tres ángulos representa el ángulo de la dirección del sensor en un plano paralelo a la pared. De esto, se puede resumir que la distancia resulta según la ecuación 4.6 siendo el ángulo de la dirección igual a 4.7.

$$\text{distancia} = \text{medición} \times \cos(\theta_{dirección}) \tag{4.6}$$

$$\begin{aligned} \cos(\theta_{dirección}) &= \cos(\theta_X) \times \cos(\theta_Y) \times \cos(\theta_Z) \\ \cos(\theta_X) &= \cos\left(\text{arcsen}\left(\text{inclinación\_compY}\right)\right) \\ \cos(\theta_Y) &= \cos\left(\text{arcsen}\left(\text{inclinación\_compX}\right)\right) \\ \cos(\theta_Z) &= \cos\left(\text{orientación}\right) \end{aligned} \tag{4.7}$$

Se observa que el factor de corrección es el coseno del ángulo de la dirección del sensor. Sin embargo, en el FPGA no se puede calcular la función trigonométrica coseno; por lo cual, se ha aproximado mediante las ecuaciones 4.8 y 4.9.

$$\text{sen}\,(\phi) \simeq \ \phi, \ \text{para}\,|\phi| \leq 0{,}1rad \tag{4.8}$$

$$\cos|\phi - \phi_0| \simeq \ 1 - |\phi - \phi_0|\,, \ \text{para}\,|\phi - \phi_0| \leq 0{,}1rad \tag{4.9}$$

Entonces, la ecuación final resultante es:

$$\begin{aligned} \cos(\theta_{dirección}) &= \cos(\theta_X) \times \cos(\theta_Y) \times \cos(\theta_Z) \\ \cos(\theta_X) &= 1 - |\text{inclinaciónY} - \text{inclinaciónY}_0| \\ \cos(\theta_Y) &= 1 - |\text{inclinaciónX} - \text{inclinaciónX}_0| \\ \cos(\theta_Z) &= 1 - |\text{orientación} - \text{orientación}_0| \end{aligned} \tag{4.10}$$

Luego de implementar la ecuación 4.10 y registrar las medidas (en base hexadecimal) al girar el robot RE1 sin desplazarlo, se obtuvo los resultados presentados en el cuadro 4.1, en donde la fila "P_0" es la condición inicial o referencia para el cálculo. Se observa que en el punto de referencia la medición y la distancia (compensada) son iguales y que la distancia no cambia en forma significativa a medida que el robot gira ligeramente.

	Medición	Inclinación "X"	Inclinación "Y"	Orientación	Distancia compensada
Po	39	42	41	83	39
	3A	42	41	93	39
	3C	42	41	A7	3A
	3A	42	41	5D	39
	3F	42	41	43	3B
	3E	3F	41	83	3B
	3A	40	41	87	39
	3C	45	41	8D	39
	40	48	41	89	3B
	3C	42	3F	83	3A
	3A	42	40	87	39
	3A	42	42	84	39
	3D	42	43	82	3B
	40	44	42	B9	3C

Cuadro 4.1: Resultado de la compensación de la distancia.

CAPÍTULO 5
SIMULACIONES Y RESULTADOS EXPERIMENTALES

5.1. ALGORITMO PARA GENERAR POSICIONES

5.1.1. Simulación

Como el objetivo del algoritmo es planear el movimiento del robot RE1 para que pueda desplazarse, se decidió tomar por factor de evaluación a la característica distancia versus tiempo. La gráfica obtenida se muestra en la figura 5.1. Asimismo, se observa en la figura 5.2 la trayectoria seguida por el robot.

Además, para poder observar el resultado de la implementación, se ha presentado una secuencia de fotos en la figura 5.3, las cuales han sido tomadas con un periodo de 30 segundos.

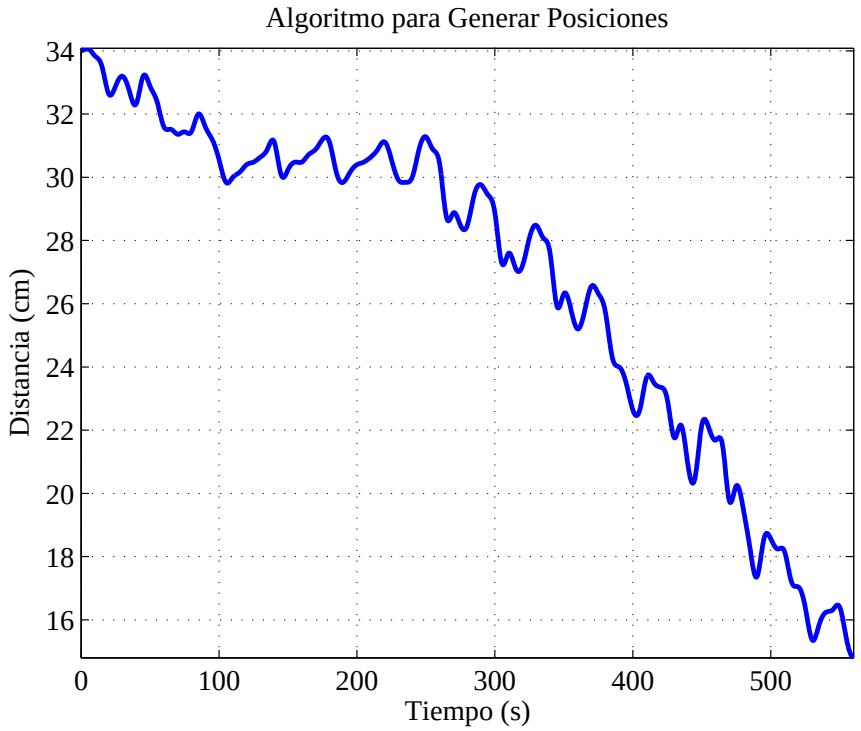


Figura 5.1: "Algoritmo para generar posiciones": Distancia vs. tiempo.

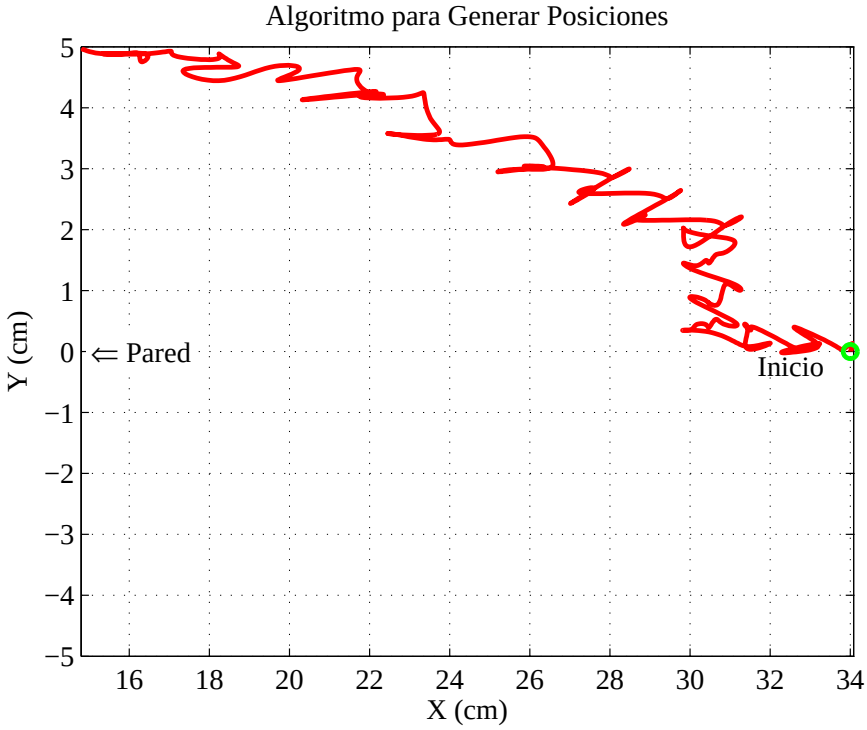
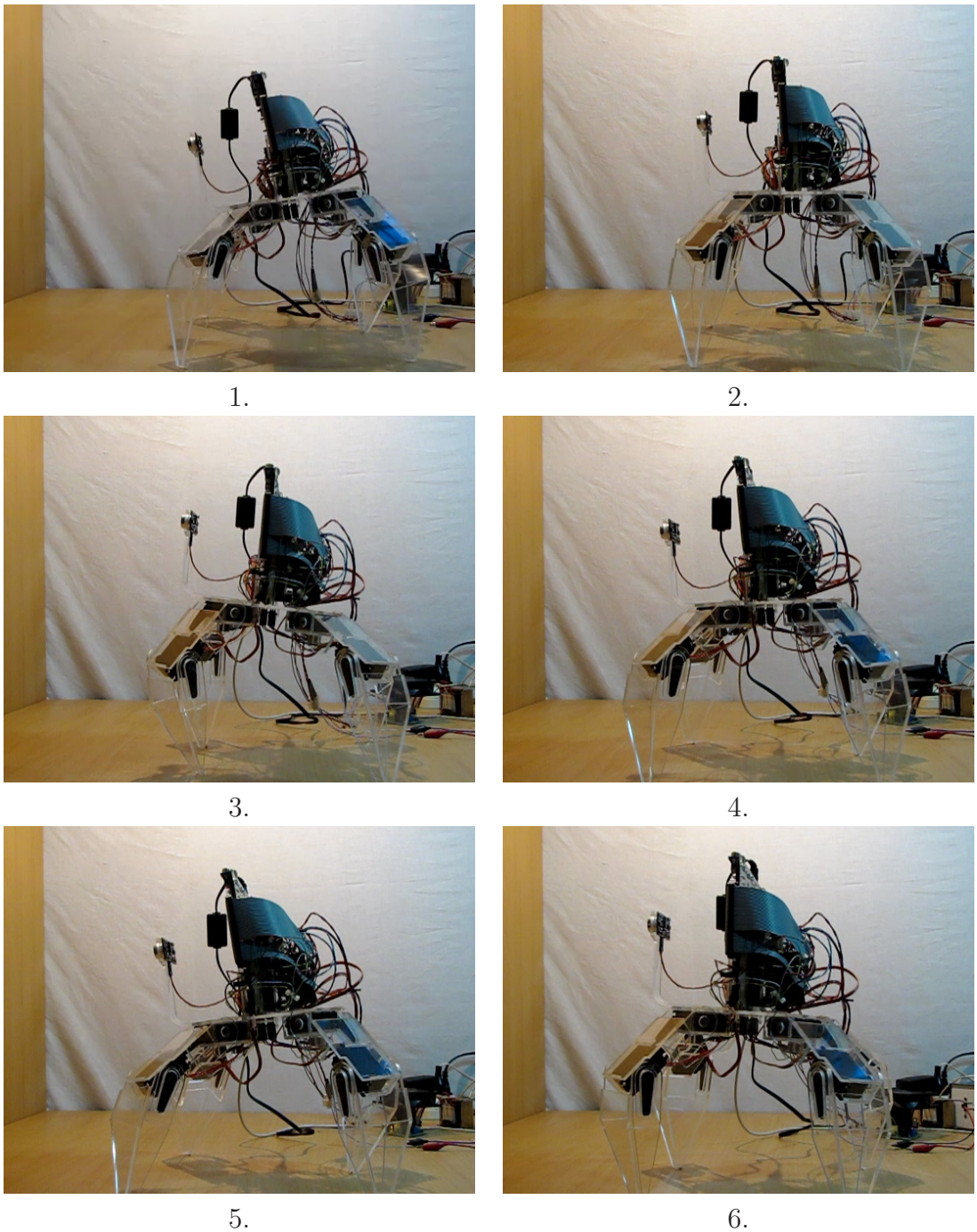


Figura 5.2: "Algoritmo para generar posiciones": Trayectoria.



1. 2. 3. 4. 5. 6.

Figura 5.3: Simulación del “algoritmo para generar posiciones”.

5.1.2. Resultados

Se han determinado dos factores que evalúen la eficiencia del algoritmo. El primero es el promedio de la derivada, llamado PD (ecuación 5.1), que mide la velocidad promedio del desplazamiento del robot. Y el segundo factor es el valor máximo de la derivada, llamado VMD (ecuación 5.2), que mide la máxima velocidad desarrollada.

$$PD = promedio\left(-\frac{\mathrm{d}distancia}{\mathrm{d}t}\right) \tag{5.1}$$

$$VMD = máx\left\{-\frac{\mathrm{d}distancia}{\mathrm{d}t}\right\} \tag{5.2}$$

Ambos factores han sido negados para poder calcular la velocidad del robot desde la figura 5.1 que representa la distancia entre este y la pared.

Luego de simular el algoritmo, se obtuvieron los siguientes resultados:

PD	2.057 cm/min
VMD	25.852 cm/min

5.2. ALGORITMO PARA GENERAR DESPLAZAMIENTOS

5.2.1. Simulación

De igual manera que el anterior algoritmo, se ha tomado por factor de evaluación a la característica distancia versus tiempo. El resultado obtenido se muestra en la figura 5.4 y la trayectoria en la figura 5.5.

Así mismo, se ha presentado una secuencia de fotos en la figura 5.6, tomadas con un periodo de 15 segundos, que muestran el resultado de la implementación del algoritmo.

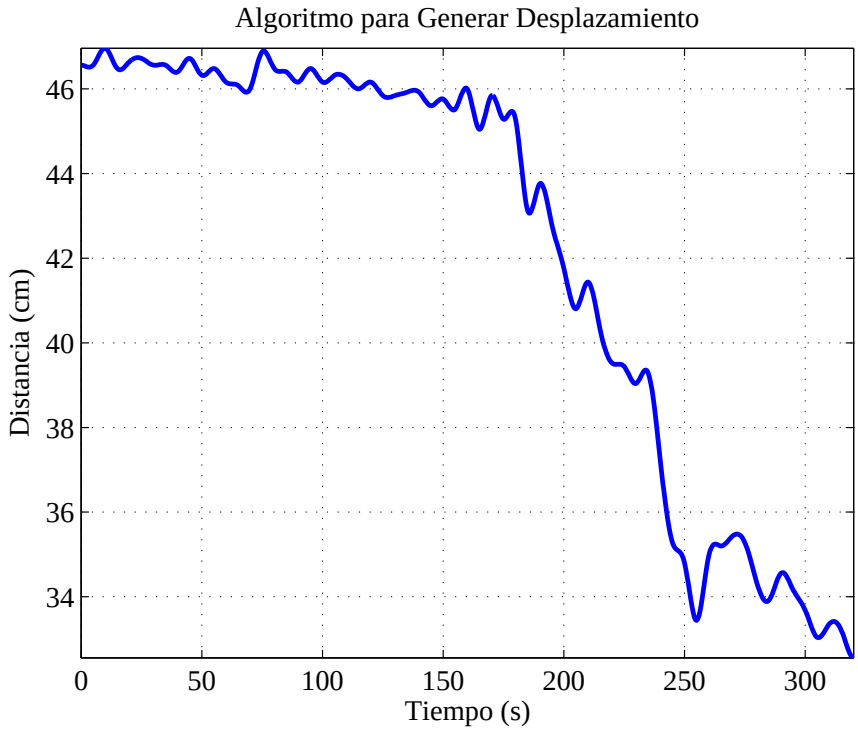


Figura 5.4: "Algoritmo para generar desplazamientos": Distancia vs. tiempo.

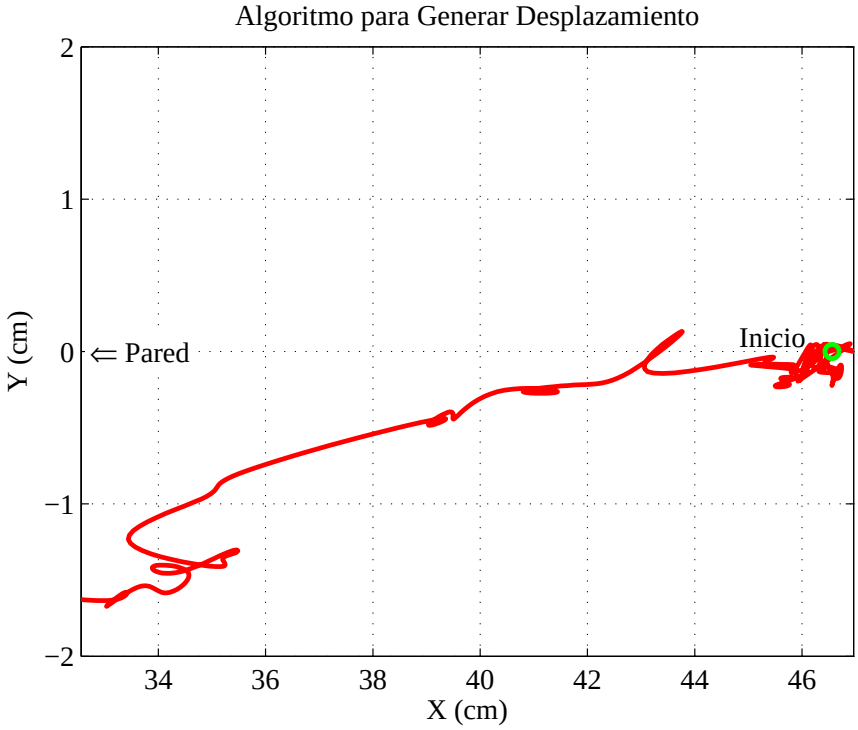
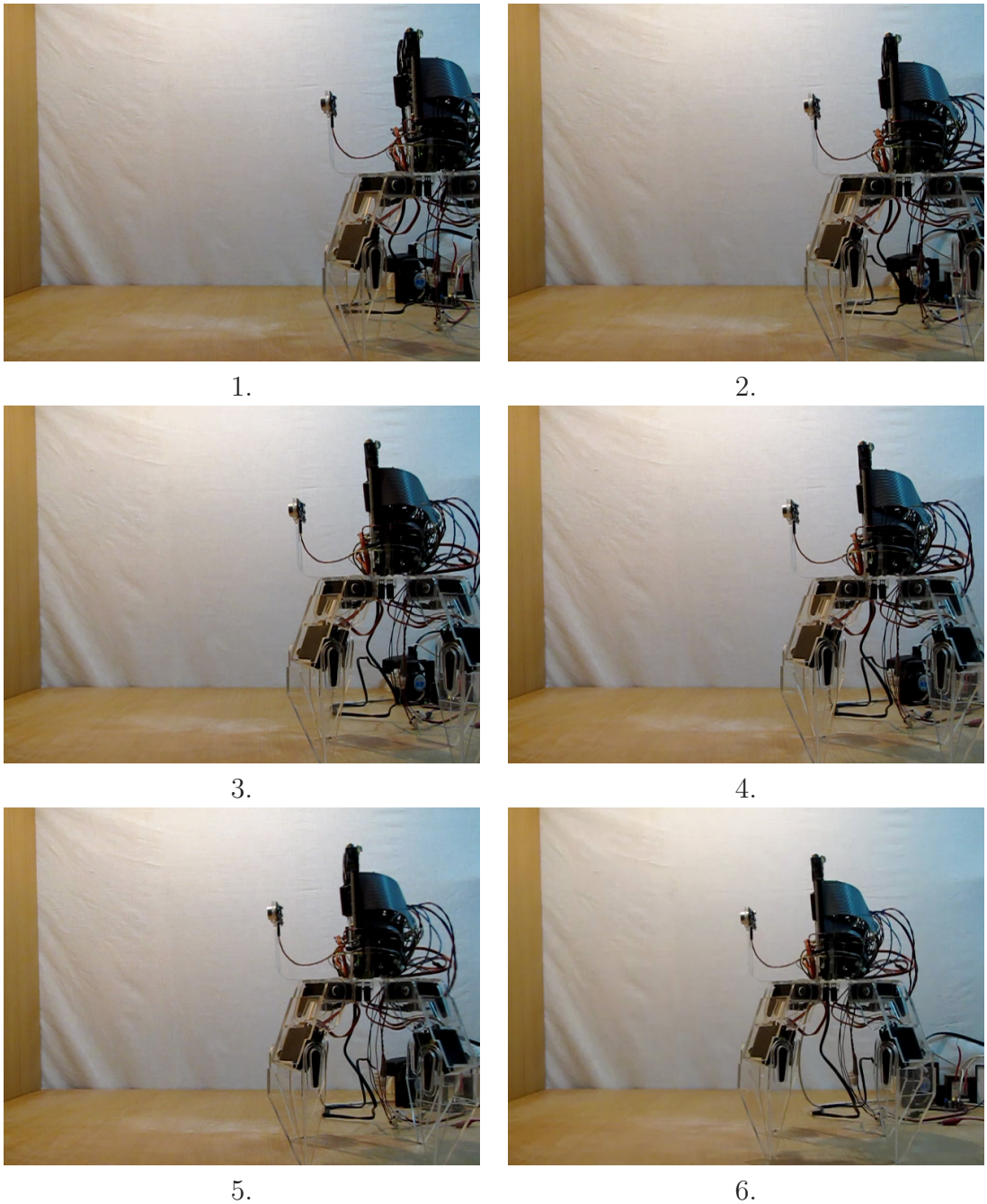


Figura 5.5: "Algoritmo para generar desplazamientos": Trayectoria.



1. 2. 3. 4. 5. 6.

Figura 5.6: Simulación del “algoritmo para generar desplazamientos”.

5.2.2. Resultados

Para poder comparar la eficiencia de ambos algoritmos, se han calculado los mismos factores y se han obtenido los siguientes resultados:

PD	2.625 cm/min
VMD	32.207 cm/min

CONCLUSIONES

- Se concluye que el “algoritmo para generar posiciones” y el “algoritmo para generar desplazamientos” permiten al robot RE1 actuar autónomamente en el planeamiento de su movimiento, de modo que este puede desplazarse. En las figuras 5.1 y 5.4, se aprecia como la velocidad de traslación del robot va aumentando a medida que cada uno de estos dos algoritmos va asimilando más el comportamiento del robot RE1, al generar secuencias de movimiento o al relacionar el desplazamiento con la posición de él.

- El “algoritmo para generar posiciones” tiene por ventaja al entrenamiento out-line que, aunque no incluye eventos dinámicos, permite al robot RE1 seguir una secuencia básica de movimiento. Este hecho se observa en la figura 5.1, en donde el robot se desplaza desde un inicio. Otra ventaja del algoritmo es la manipulación del movimiento inicial del robot RE1 debido a que las secuencias futuras de traslación del robot van a ser variaciones de la inicial que fue dada por el entrenamiento out-line. En la figura 5.1, se ve que el movimiento del robot sigue un patrón que varía en el tiempo. Por otro lado, la desventaja del algoritmo radica en su método de aprendizaje de prueba y error que le lleva a perder tiempo en ensayos fallidos, como se aprecia en el periodo entre 100 y 150 segundos de la figura 5.1.

- El “algoritmo para generar desplazamientos” tiene las ventajas, por un lado, de no guardar secuencias sino relaciones, como se observa en la figura 5.4, en donde el desplazamiento del robot RE1 no sigue ningún patrón; y, por el otro lado, de dar restricciones al movimiento del robot RE1 (en el algoritmo genético) evitando así ciertas conductas como un giro mayor a 6° en una articulación, por ejemplo. En contraposición, este algoritmo tiene por desventaja al tiempo muerto al iniciar la traslación, en el cual se obtiene los primeros pesos válidos de la red neuronal. En la figura 5.4, el robot recién se mueve en forma notable a partir de los 150 segundos.

- Se concluye que el “algoritmo para generar posiciones” y el “algoritmo para generar desplazamientos” tienen un desempeño semejante ya que la velocidad promedio desarrollada por el robot RE1 fue parecida en ambos algoritmos (2.057 cm/min para el “algoritmo para generar posiciones” y 2.625 cm/min para el “algoritmo para generar desplazamientos”).

- La ventaja de las redes neuronales y de los algoritmos genéticos es su adaptabilidad. En las redes neuronales, los algoritmos de entrenamiento permiten enseñar a la red la respuesta o conducta deseada ante una entrada dada. Este hecho se percibe en el entrenamiento out-line del “algoritmo para generar posiciones”. De igual modo, los algoritmos genéticos hayan máximos de funciones abstractas como la red neuronal del “algoritmo para generar desplazamientos”.

- La aleatoriedad es importante para el “algoritmo para generar posiciones” y para el “algoritmo para generar desplazamientos” porque les permite explorar más ampliamente los espacios de búsqueda. En consecuencia, al primer algoritmo le permite no atascarse en máximos locales y al segundo buscar la máxima variación del desplazamiento en la red neuronal.

BIBLIOGRAFÍA

[1] CLUNE, Jeff, BECKMANN, Benjamin, OFRIA, Charles & PENNOCK, Robert. *Evolving Coordinated Quadruped Gaits with the HyperNEAT Generative Encoding.* Proceedings of the 2009 IEEE Congress on Evolutionary Computing Special Section on Evolutionary Robotics, Trondheim, Norway., páginas 2764-2771. 2009.

[2] BONGARD, Josh, ZYKOV, Victor & LIPSON, Hod. *Resilient Machines Through Continuous Self-Modeling.* Science, volumen 314: páginas 1118-1121, Noviembre 2006.

[3] HORNBY, Gregory S., TAKAMURA, Seichi, YAMAMOTO, Takashi & FUJITA, Masahiro. *Autonomous Evolution of Dynamic Gaits with Two Quadruped Robots.* IEEE Transactions on Robotics, volumen 21, 2005.

[4] ZHANG, Jiaqi & CHEN, Qijun. *Learning Based Gaits Evolution for an AIBO Dog.* Proceedings of the 2007 IEEE Congress on Evolutionary Computation, Singapore., páginas 1523-1526. 2007.

[5] GALLAGHER, John C., BEER, Randall D., ESPENSCHIED, Kenneth S. & QUINN, Roger D. *Application of Evolved Locomotion Controllers to a Hexapod Robot.* Robotics and Autonomous Systems, ELSEVIER, volumen 19: páginas 95-103, 1996.

[6] WETTERGREEN, David & THORPE, Chuck. *Gait Generation for Legged Robots.* Proceedings of the IEEE International Conference on Intelligent Robots and Systems, volumen 2: páginas 1413-1420, Julio 1992.

[7] BONGARD, Josh, ZYKOV, Victor & LIPSON, Hod. *Automated Synthesis of Body Schema using Multiple Sensor Modalities.* Computational Synthesis Laboratory, Cornell University. Paper.

[8] BONGARD, Josh & LIPSON, Hod. *Automatic Synthesis of Multiple Internal Models through Active Exploration.* American Association for Artificial Intelligence, 2005.

[9] BONGARD, Josh, & LIPSON, Hod. *Automated Damage Diagnosis and Recovery for Remote Robotics.* Sibley School of Mechanical and Aerospace Engineering, Cornell University. Paper.

[10] HIROSE, Shigeo, OHNO, Hidetaka, MITSUI, Takeo & SUYAMA, Kiichi. *Design and Experiments of In-Pipe Inspection Vehicles for ø25, ø50, ø150 Pipes.* Journal of Robotics and Mechatronics, volumen 12: número 3, páginas 310-317. 2000.

[11] CARBONE, C., MALCHIKOV, A., CECCARELLI, M. & JATSUN, S. *Design and Simulation of Kursk Robot for In-Pipe Inspection.* SYROM 2009: Proceedings of the 10th IFToMM International Symposium on Science of Mechanisms and Machines, held in Brasov, Romania, october 12-15, 2009. Springer, páginas 103-114. 2010. Primera edición.

[12] SERNA, M. A., AVELLO, A., BRIONES, L. & BUSTAMANTE, P. *ROBICEN: A Pneumatic Climbing Robot for Inspection of Pipes and Tanks.* Experimental Robotics V: The Fifth International Symposium Barcelona, Catalonia, June 15 - 18, 1997. Springer Berlin/Heidelberg, volumen 232: páginas 325-334. 1998.

[13] NONAMI, Kenzo & HUANG, Qing-Jiu. *Humanitarian Mine Detecting Six-legged Walking Robot and Hybrid Neuro Walking Control with Position/Force Control.* Mechatronics, volumen 13: páginas 773-790. 2003.

[14] SQUYRES, Steve. *Roving Mars: Spirit, Opportunity and the Exploration of the Red Planet.* Scribe Publications, 2005.

[15] ERICKSON, James K., CALLAS, John L. & HALDEMANN, Albert F. C. *The Mars Exploration Rover Project : 2005 surface operations results.* 56th International Astronautical Congress, Fukuoka, Japan, Octubre 17-21, 2005.

[16] BLITCH, John. *Tactical Mobile Robots for Complex Urban Environments.* Mobile Robots XIV, SPIE, volumen 3838: número 1, páginas 116-118, 1999.

[17] ARKIN, Ronald C., COLLINS, Thomas R. & ENDO, Yoichiro. *Tactical Mobile Robot Mission Specification and Execution.* Mobile Robots XIV, SPIE, volumen 3838: número 1, páginas 150-163, 1999.

[18] COLLINS, Thomas R., ARKIN, Ronald C., CRAMER, Michael J. & ENDO, Yoichiro. *Field Results for Tactical Mobile Robot Missions.* Assoc. for Unmanned Vehicle Systems International, 2000.

[19] GUIJARRO, Bertha, FONTELA, Óscar & SÁNCHEZ, Noelia. *Inteligencia Artificial: Técnicas, métodos y aplicaciones.* Capítulo 15: Redes Neuronales, páginas 649-689. Mc Graw Hill. 2008. Primera edición.

[20] FAUSETT, Laurene, editor. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications.* Prentice-Hall, Inc., páginas 3-5, 43-47. 1994.

[21] NILSSON, Nils J. *Introduction To Machine Learning: An Early Draft of a Proposed Textbook.* Stanford, páginas 39-68. Diciembre 1996. Publicación independiente.

[22] PFEIFER, Rolf & SCHEIER, Christian. *Understanding Intelligence*. MIT Press, páginas 139-152, 167-172, 236-240. 1999.

[23] HASSOUN, Mohamad H. *Fundamentals of Artificial Neural Networks*. MIT Press, páginas 57-142, 197.1995.

[24] PÉREZ, Anselmo. *Una Introducción a la Computación Evolutiva*. páginas 17-45. Marzo 1996. Publicación independiente.

[25] MITCHELL, Melanie. *An Introduction to Genetic Algorithms*. MIT Press, páginas 166-174. 1998.

[26] LA VALLE, Steven M. *Planning Algorithms*. Cambridge University Press, páginas 1-37. 2006.

[27] SIEGWART, Roland & NOURBAKHSH, Illah R. *Introduction to Autonomous Mobile Robots*. MIT Press, páginas 47-88. 2004.

[28] NOCEDAL, Jorge & WRIGHT, Stephen J. *Numerical Optimization*. Springer, páginas 34-63. 1999.

[29] LINDSAY, Andy. *Smart Sensors and Applications*. Parallax Inc, páginas 41-166.2006. Tercera edición.

[30] ANSI/IEEE Std 754-1985. *IEEE Standard for Binary Floating-Point Arithmetic*. The Institute of Electrical and Electronics Engineers, Inc., 1985.

[31] XILINX. *Spartan-3E FPGA Starter Kid Board User Guide*, Junio 2008. Manual.