UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA MECÁNICA E INDUSTRIAL

IMPLEMENTACIÓN DE UNA RED NEURONAL ARTIFICIAL EN UN FPGA PARA EL CONTROL DE UN MOTOR DE CD

TESIS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO MECATRÓNICO

PRESENTA:

OSVALDO ROMERO JAIMES

DIRECTOR DE TESIS:

M.I. SERAFÍN CASTAÑEDA CEDEÑO

MÉXICO, D. F. JUNIO, 2014

A mis padres, por todo su apoyo y cariño.

AGRADECIMIENTOS

Mamá, gracias por mostrarme que en la vida se pueden alcanzar las metas propuestas. Debes saber que te admiro mucho por todo lo que has logrado y también porque siempre vas por más. Soy afortunado de tener una madre como tú, eres única.

Papá, gracias por todo el apoyo, por estar siempre al pendiente de tus hijos y por ser otro ejemplo de vida para mí, me inyectaste desde pequeño las ganas de hacer múltiples actividades en la vida.

Nelly, te agradezco por ser más que una hermana, una gran amiga y por tus "¡apúrale!". Al fin la terminé.

Gracias a Doña Chepis, que ha sido la mujer más maravillosa que he conocido y que espero algún día ser tan grande como ella. A las tías Jaimes: Mela, Gude, Rebe, Reyna, Lali, Edith y Fina; maravillosas mujeres educadas por la mejor madre que se pueda tener en la vida. Primos, les agradezco su apoyo y por ponerme el ejemplo.

A la familia Romero, a quienes también dedico este trabajo. Tíos Samuel, Miriam, Juan y María e hijos. De mis primos puedo decir que los admiro mucho por todas las ganas que le echan y por ponerles un buen ejemplo a sus hijos. Dedico de manera especial este trabajo a Humberto, a quien desde niño admiré y del que jamás olvidaré sus "zapes". Me siento orgulloso de haber podido estudiar en la misma facultad y contento porque gracias a la UNAM lo pude visitar de nuevo.

Toño y José, más que vecinos, amigos de la familia. Les dedico este trabajo también a ustedes, porque fuera de casa he encontrado en ustedes la inspiración para desarrollarme profesionalmente.

Linda, ¡mi bB! Contigo retomé este trabajo. Gracias por todo tu apoyo y observaciones, me tardé un poco en seguir tu consejo, pero al final lo hice, ya está listo este trabajo, valió la pena la espera.

Agradezco también a todos mis amigos que me apoyaron con su conocimiento, sugerencias, al igual a los que mostraron interés en el proyecto: Juan Carlos, Juan Alejandro, Emilio, etc. También a la *mexikanische FAUmilie¸* gracias Maytte, Jorge, César y Eru, por su apoyo y por sus palabras. Danke euch Sere und Özden, weil ihr immer Interesse an meinem Studium hattet.

A mis amigos de antaño Jerry, José Manuel, Miguel, Donato y Javier tuve el placer de conocerlos desde la preparatoria y la fortuna de conservar su amistad a pesar de las diferentes actividades que realizamos.

A la señora Cabrera, alguna vez me dijo que entrando a la universidad la íbamos a olvidar, pero no, siempre tengo presente todo lo que hizo por nosotros,

chamacos de preparatoria, de los que se encargó como si fueran sus hijos. Gracias por todo ello y más.

Agradezco al Mtro. Serafín Castañeda por la oportunidad que me dio para realizar el servicio social y tesis con él.

Extiendo mi agradecimiento a los profesores que dedicaron su tiempo para leer este trabajo y que ayudaron a que este trabajo fuera lo mejor posible: Mtro. Rigel Gámez, Ing. Juan Carlos Ramírez, Ing. Ana Marissa Juárez e Ing. Gloria Mata.

Por último, agradezco a la Universidad Nacional Autónoma de México por la gran educación que me dio y por la oportunidad de desarrollarme dentro y fuera de ella.

1 ÍNDICE

ABREVIATURAS

Tabla 0-1 Abreviaturas del trabajo

SNC	Sistema nervioso central
RN	Red Neuronal
RNA	Red Neuronal Artificial
MLP	Perceptrón Multicapa
LMS	Last Mean Square
BP	Backpropagation
LMBP	Levenberg Marquardt Backpropagation
P	Proporcional
PD	Proporcional Derivativo

NOMENCLATURA

p: *Letras simples son escalares*
$\boldsymbol{W}$: *Letras en* $\boldsymbol{negritas}$ *representan matrices o vectores*

RESUMEN

En este trabajo se presenta una de las aplicaciones de las redes neuronales artificiales (RNA) en el área del control de sistemas dinámicos, en este caso específico para controlar la posición del rotor de un motor de corriente directa. El objetivo principal es demostrar que una RNA puede imitar la respuesta de un controlador clásico (P y PD).

La implementación física de la RNA es llevada a cabo utilizando un módulo de tiempo real de la compañía *National Instruments*® (cRIO) y los resultados se muestran por medio de una interfaz gráfica desarrollada en LabVIEW®.

Por último, son presentados los resultados de este trabajo así como las mejoras que se pueden realizar en trabajos futuros o en otras áreas, como en el caso de la instrumentación, donde se han usado RNA en el acondicionamiento de señales.

1. INTRODUCCIÓN

1.1 HIPÓTESIS

El motor de corriente directa utilizado en este trabajo será controlado por medio de un sistema basado en FPGA con una RNA. La variable controlada será la posición, mientras que la variable manipulada será el voltaje en las terminales del motor.

Una RNA es capaz de sustituir/ reemplazar un compensador tradicional para controlar la posición del rotor de un motor de corriente directa. Dicha RNA puede ser implementada en un FPGA.

En cuanto al aprendizaje de la RNA, su estructura deberá ser suficiente para memorizar los patrones de entrada y a su vez aproximarse a la respuesta del controlador.

1.2 OBJETIVOS GENERALES

El objetivo principal de esta tesis es diseñar un controlador basado en redes neuronales artificiales que permita controlar la posición del rotor de un motor de corriente directa e implementarlo en un dispositivo que permita ejecutar los procesos en tiempo real y/o en paralelo. Para ello se definen los siguientes objetivos particulares:

- Diseñar un controlador basado en redes neuronales artificiales.
- Implementar una red neuronal artificial que imite la respuesta de un controlador del tipo P y uno del tipo PD para un motor de corriente directa.
- El algoritmo de entrenamiento de la red deberá converger rápidamente.
- Implementar la red neuronal en un dispositivo de tiempo real.
- Verificar que una red neuronal se puede utilizar como una alternativa a las técnicas de control tradicionales.

1.3 JUSTIFICACIÓN

Las denominadas Redes Neuronales Artificiales tienen muchas aplicaciones en diversas áreas, por ejemplo: el sector social y económico, campos de estudio como la probabilidad, estadística, entre otras áreas de la ingeniería. Siendo específicos, una aplicación en el sector de ingeniería se presenta en los sistemas de control. Las RNA, con sus capacidades de aprendizaje, representan una alternativa para encontrar mejores soluciones a los problemas de control donde el modelo matemático es muy complejo o multi-variable, incluso cuando es difícil determinar la dinámica del sistema [1]. Dichas capacidades aunadas a la capacidad de llevar a cabo procesos en paralelo de manera masiva hacen de ellas una alternativa atractiva para ejecutar tareas complejas donde las computadoras concebidas por Babbage y Von Neumann no son muy útiles [2].

En sistemas de control convencionales, sólo se pueden comparar valores y tomar decisiones determinadas con base en el modelo matemático propuesto del sistema dinámico. Las RNA, en cambio, son flexibles, pueden aprender del sistema y del entorno sin necesidad de modelar matemáticamente el sistema analizado, similar a técnicas de control más complejas, como controladores adaptables. Por otra parte, se tiene la ventaja de poder trabajar con sistemas no lineales.

En resumen, en el área de control de sistemas dinámicos es frecuente encontrarse con la necesidad de diseñar estrategias de control para procesos no lineales y multi-variables. Las redes neuronales pueden usarse para resolver problemas de control no lineales [3].

1.4 INTRODUCCIÓN A LAS REDES NEURONALES ARTIFICIALES

1.4.1 REDES NEURONALES

En biología, una red neuronal es un conjunto de conexiones sinápticas ordenadas que dan como resultado la unión de neuronas con otras en sus regiones correspondientes.

El cerebro contiene múltiples redes y forma parte del sistema nervioso central (SNC). Las características del SNC son:

- Inclinación a adquirir conocimiento de la experiencia.
- Conocimiento almacenado en conexiones sinápticas.
- Gran plasticidad neuronal.
- Comportamiento no lineal.
- Alta tolerancia a fallos (muerte de neuronas).
- Apto para reconocimiento, percepción y control.

Neurona: Tipo de célula del sistema nervioso cuya característica es la excitabilidad eléctrica de su membrana plasmática.

Se estima que en cada milímetro del cerebro hay cerca de 50,000 neuronas y en total se encuentran más de cien mil millones de neuronas en el cerebro [4].

Las partes de la neurona biológica se muestran en la Fig. 1-1, las cuales se describen a continuación:

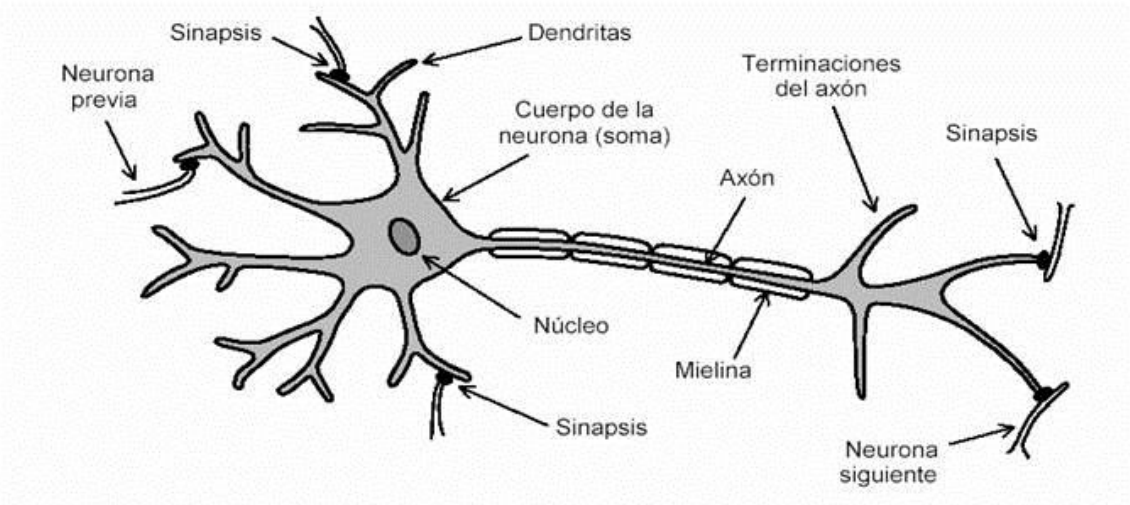


Fig. 1-1 Partes de una Red neuronal biológica [5].

- **Soma**: Es el *cuerpo* de la neurona, contiene al núcleo. Se encarga de las actividades metabólicas de la neurona y recibe información de las demás neuronas (vecinas) a través de las conexiones sinápticas.

- **Dendritas**: Son aquellas que parten del soma y tienen ramificaciones. Se encargan de la recepción de las señales de las otras células a través de conexiones llamadas sinápticas. Si se piensa en términos de electrónica, podemos afirmar que las dendritas son las entradas del sistema.

- **Axón**: Es la salida de la neurona, se utiliza para mandar impulsos o señales a otras células nerviosas. Cuando el axón está cerca de sus células de destino se ramifica y forma sinapsis con el soma, cuyas uniones pueden ser "inhibidora" o "excitadora" dependiendo del transmisor que las libere.

Cada neurona recibe de 10,000 a 100,000 sinapsis y el axón realiza una cantidad similar de conexiones [4].

1.4.2 REDES NEURONALES ARTIFICIALES

Una red neuronal artificial es básicamente un modelo basado en las redes neuronales cerebrales.

En analogía con una red neuronal biológica se busca hacer una red de neuronas que se intercomuniquen y cuya información permita entrenar a la red. Las partes de la RNA son similares a una RN biológica. Constan de un número grande de procesadores simples ligados por conexiones con pesos. Las unidades de procesamiento se denominan neuronas. Dichas unidades reciben entradas de otros nodos y generan una salida simple escalar dependiente de la información local disponible (guardada o proveniente de las conexiones con pesos) [6].

La siguiente tabla muestra las ventajas de las RNA.

Tabla 1-1 Ventajas de las redes neuronales artificiales [7].

No linealidad	El procesador neuronal [8] es básicamente no lineal y, por consecuencia, la red neuronal también.
Transformación entrada - salida	El proceso de aprendizaje consiste básicamente en presentar a la red un ejemplo y modificar sus pesos sinápticos de acuerdo con su respuesta. Aprende, por lo tanto, una transformación entrada/salida.
Adaptabilidad	La red tiene la capacidad de adaptar sus parámetros, aún en tiempo real.
Tolerancia a fallas	Debido a la interconexión masiva, la falla de una neurona no altera seriamente la operación.
Analogía con las redes biológicas	Esto permite la utilización mutua del conocimiento de las dos áreas.

A continuación se muestran algunos tipos de modelos matemáticos de redes neuronales, más adelante se detallarán los modelos seleccionados.

1.4.2.1 MODELO DE UNA NEURONA ARTIFICIAL

La neurona es la unidad de proceso de información fundamental en una red neuronal [9]. Una neurona simple se compone de una entrada, un peso y un sesgo, los cuales son transformados por una función de activación que da como resultado la salida de la red. La siguiente figura muestra el modelo de una neurona artificial de múltiples entradas. Posteriormente se describen sus elementos [7]. La nomenclatura empleada se puede encontrar en el libro *Neural Network Design* [10].

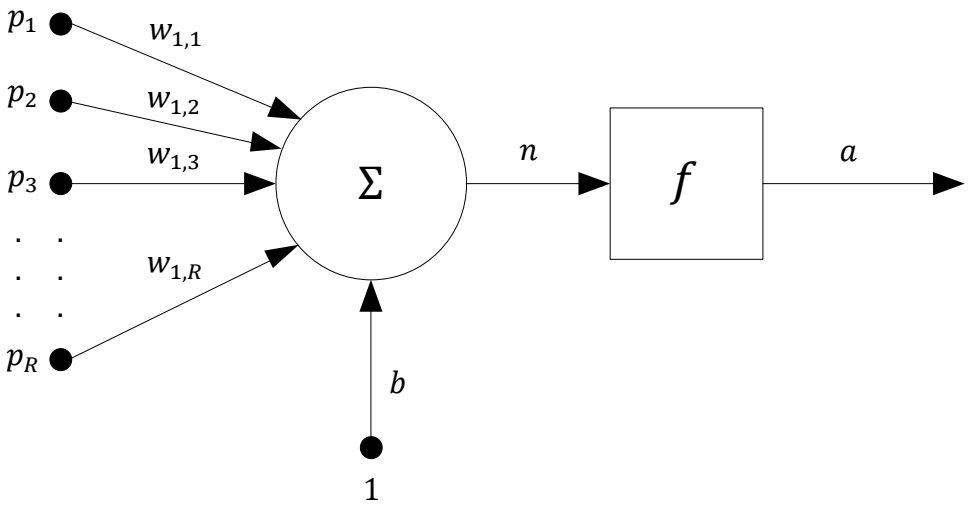


Fig. 1-2 Modelo de una neurona artificial.

Los elementos identificados de la figura anterior son:

- **Valores de entrada**: Estos corresponden a las entradas individuales p_1, p_2, p_3 y p_R, las cuales forman el vector de entrada $\boldsymbol{p}$, de dimensión Rx1, es decir:

$$\boldsymbol{p} = [p_1 \ p_2 \ p_3 \dots p_R \ ]^T$$

- **Enlaces de conexión**: Parametrizados por los pesos sinápticos $w_{1,R}$. El primer subíndice corresponde a la neurona receptora, mientras que el segundo subíndice corresponde al patrón de entrada. El conjunto de pesos sinápticos forman el vector de pesos $\boldsymbol{W}$ de dimensión 1xR, para este caso.

$$\boldsymbol{W} = \begin{bmatrix} w_{1,1} \ w_{1,2} \ w_{1,3} \dots w_{1,R} \end{bmatrix}$$

- **Umbral o sesgo**: Este elemento desplaza la entrada, su nomenclatura es la letra b. Para este modelo es una cantidad escalar.

- **Sumador**: Suma los componentes de las señales de entrada multiplicadas por los elementos del vector $\boldsymbol{W}$ más el valor del sesgo. Esta suma es una cantidad escalar.

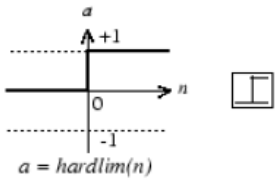
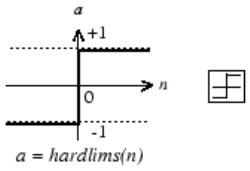
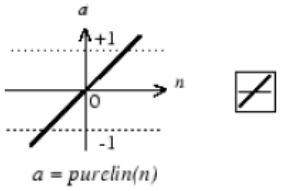
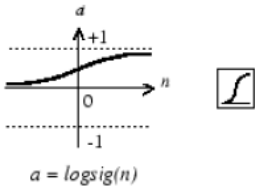
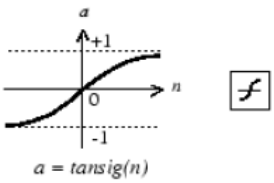
$$n = w_{1,1}p_1 + w_{1,2}\,p_2 + w_{1,3}p_3 + \cdots + w_{1,R}p_R + b$$

  Esta expresión puede ser escrita en la siguiente forma matricial:

$$n = \boldsymbol{W}\boldsymbol{p} + b$$

- **Función de activación** $f(n)$: Puede ser una función lineal o no lineal de n. Se escoge una función de activación para satisfacer alguna especificación del problema que se quiere resolver. En la siguiente tabla se ejemplifican algunas funciones, se incluye comandos de MATLAB®.

Tabla 1-2 Funciones de Activación.

Función	Relación Entrada/Salida	Gráfica	Función en MATLAB
Límite alto	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		*hardlim*
Límite simétrico alto	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		*hardlims*
Lineal	$a = n$		*purelin*
Sigmoidea - logarítmica	$a = \dfrac{1}{1 + e^{-n}}$		*logsig*
Sigmoidea tangente hiperbólica	$a = \dfrac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$		*tansig*

Otra manera de expresar el modelo de la neurona artificial es la siguiente:

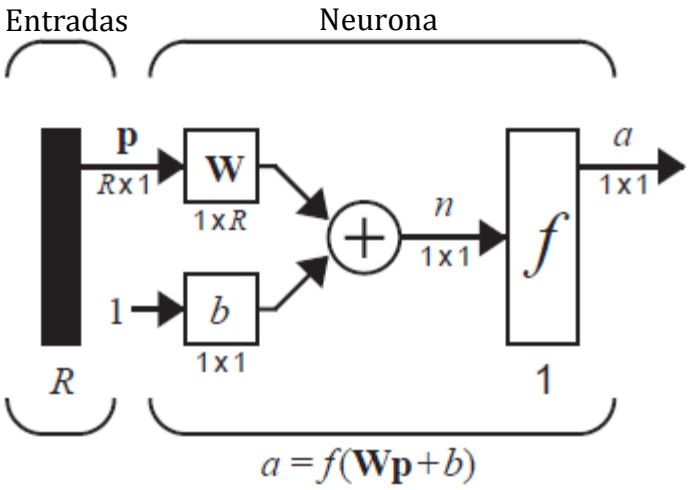


Fig. 1-3 Notación abreviada de una neurona con R entradas [10].

1.4.2.2 CAPA DE NEURONAS

Una capa es un conjunto de redes neuronales asociadas en paralelo con respecto a sus entradas [10]. Se puede ver como una red de una capa de S neuronas. La siguiente figura muestra la representación de una capa.

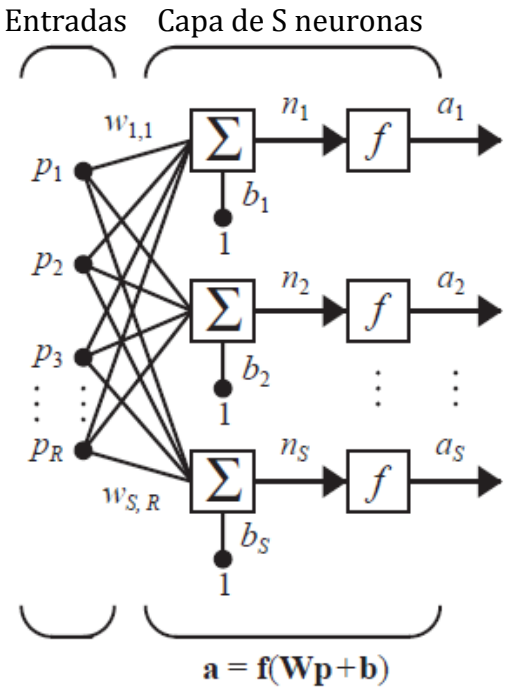


Fig. 1-4 Capa de S neuronas [10].

Cada una de las R entradas está conectada a cada neurona y los pesos ahora tienen S renglones, formando una matriz de SxR. De esta manera, el vector de pesos es el siguiente:

$$\boldsymbol{W} = \begin{bmatrix} w_{1,1} & w_{1,1} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

Observando la Fig. 1-4, encontramos que la salida de la neurona $\boldsymbol{a}$ puede ser representada por un vector de dimensión Sx1, al igual que $\boldsymbol{n}$. Otra manera de representar se muestra en la Fig. 1-5:

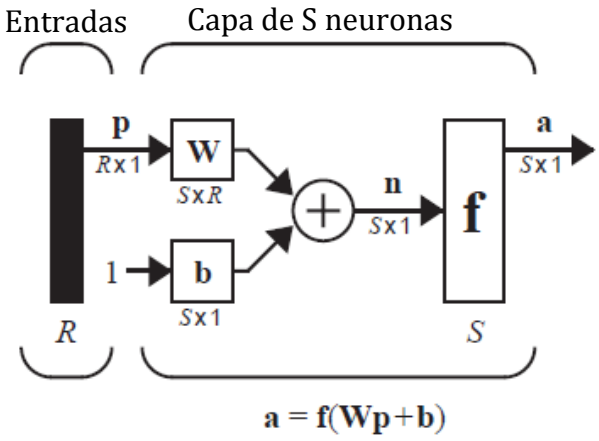


Fig. 1-5 Notación abreviada de una capa de S neuronas [10].

1.4.2.3 RED NEURONAL DE CAPAS MÚLTIPLES

Las redes neuronales de capas múltiples o multicapa se distinguen por tener una o más capas de neuronas ocultas, cuyos nodos computacionales se denominan neuronas ocultas o unidades ocultas [10].

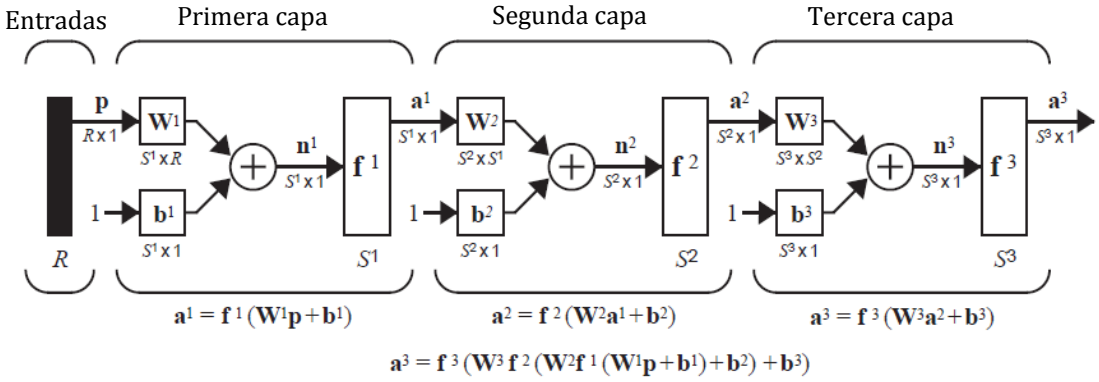


Fig. 1-6 Notación abreviada una red multicapa [10].

La notación es la misma que para las capas simples, el único cambio es el uso de superíndices de $\boldsymbol{S}$, los cuales señalan la capa en la que se está trabajando. La salida final se representa con la letra $\boldsymbol{a}$ con el superíndice más grande, que debe ser igual a la cantidad de capas de la red neuronal.

1.4.2.4 EJEMPLOS DE RNA

La tabla siguiente muestra algunos antecedentes de las RNA:

Tabla 1-3 Ejemplos de las redes neuronales artificiales en la historia [11].

NOMBRE	DISEÑADOR	CARACTERÍSTICAS	AÑO	TIPO
Adaline y Madaline	Bernard Widrow	Técnicas de adaptación para el reconocimiento de patrones.	1960	Predicción
Adaptive Resonance Theory Networks (ART)	Carpenter, Grossberg	Reconocimiento de patrones y modelo del sistema neuronal. Concepto de resonancia adaptiva.	1960-86	Conceptualización
Back-Propagation	Rumelhart y Parker	Solución a las limitaciones de su red predecesora, el Perceptrón.	1985	Clasificación
Bi-Directional Associative Memory (BAM) Networks	Bart Kosko	Inspirada en la red ART.	1987	Asociación
The Boltzmann Machine	Aackley, Hinton y Sejnowski	Similar a la red Hopfield.	1985	Asociación
Brain-State-in a Box	James Anderson	Red Asociativa Lineal.	1970-86	Asociación
Cascade-Correlation-Networks	Fahhman y Lebiere	Adición de nuevas capas ocultas en cascada.	1990	Asociación
Counter-Propagation	Hecht-Nielsen	Clasificación adaptiva de patrones.	1987	Clasificación
Delta-Bar-Delta (DBD) Networks	Jacobb	Métodos heurísticos para acelerar la convergencia.	1988	Clasificación
Digital Neural Network Architecture (DNNA) Networks	Neural Semiconductor Inc.	Implementación Hardware de la función sigmoidea.	1990	Predicción
Directed Random Search (DRS) Networks	Maytas y Solis	Técnica de valores Random en el mecanismo de ajuste de pesos.	1965-81	Clasificación
Functional-link-Networks (FLN)	Pao	Versión mejorada de la red *Backpropagation*.	1989	Clasificación
Hamming Networks	Lippman	Clasificador de vectores binarios utilizando la Distancia Hamming.	1987	Asociación
Hopfield Networks	Hopfield	Concepto de la red en términos de energía.	1982	Optimización
Learning Vector Quantization (LVQ) Networks	Kohonen	Red Clasificadora.	1988	Clasificación
Perceptron Networks	Rosenblatt	Primer modelo de sistema Neuronal Artificial.	1950	Predicción
Probabilistic Neural Network (PNN)	Spetcht	Clasificación de patrones utilizando métodos estadísticos.	1988	Asociación
Recirculation Networks	Hinton y McClelland	Alternativa a la red *Backpropagation*.	1988	Filtrado
Self-Organizing Maps (SOM)	Kohonen	Aprendizaje sin supervisión.	1979-82	Conceptualización
Spatio-Temporal-Pattern Recognition (SPR)	Grossberg	Red Clasificadora Invariante en el espacio y tiempo.	1960-70	Asociación

Como se puede observar en la tabla anterior, el desarrollo de la RNA es relativamente reciente. El estudio de estos sistemas ha permitido encontrar distintas aplicaciones, las cuales tienen que ver con el tipo del sistema neuronal.

De manera general, las aplicaciones notables de las RNA tienen que ver con la predicción de eventos, clasificación e identificación de patrones, asociación y filtrado de datos. En la siguiente sección se mencionan algunas aplicaciones específicas.

1.4.2.5 APLICACIONES

Las aplicaciones de las RNA son variadas, se pueden encontrar en diversas áreas [6]. Algunos ejemplos de éstas son:

- Automóviles:
  - Sistemas de piloto automático.
  - Detección de fallas por reconocimiento externo de vibraciones.
- Bancos:
  - Lectura de cheques y otros documentos.
  - Evaluación de aplicaciones de créditos.
- Electrónica:
  - Predicción de secuencia de códigos.
  - Distribución de elementos en circuitos integrados.
  - Control de procesos.
  - Análisis de fallas.
  - Visión artificial.
  - Reconocimiento de voz.
  - Filtros.
- Finanzas:
  - Asesoría de préstamos.
  - Previsión en la evolución de precios.
  - Seguimiento de hipotecas.
  - Análisis de uso de línea de crédito.
  - Identificación de falsificaciones.
  - Interpretación y reconocimiento de firmas.
- Manufactura:
  - Control de la producción y del proceso.
  - Análisis y diseño de productos.
  - Diagnóstico de fallas en el proceso y maquinarias.
  - Identificación de partículas en tiempo real.
- Medicina:
  - Análisis de células portadoras de cáncer mamario.
  - Análisis de electroencefalograma y de electrocardiograma.
  - Diseño de prótesis.
- Robótica:
  - Control dinámico de trayectoria.
  - Robots elevadores.
  - Controladores.
  - Sistemas ópticos.
  - Modelado de sistemas.
- Seguridad:

    - Códigos de seguridad adaptativos.
    - Criptografía.
    - Reconocimiento de huellas digitales.
- Telecomunicaciones:
    - Compresión de datos e imágenes.
    - Automatización de servicios de información.
    - Traslación en tiempo real del lenguaje hablado.
- Transporte:
    - Diagnóstico de frenos en camiones.
    - Sistemas de ruteo y seguimiento de flotas.
- Voz:
    - Reconocimiento de voz.
    - Compresión de voz.
    - Clasificación de vocales.
    - Transformación de texto escrito a voz.

1.4.2.6 DISPOSITIVOS DE IMPLEMENTACIÓN

Una RNA demanda grandes recursos computacionales debido a la cantidad de operaciones que se tienen que llevar a cabo y de manera paralela.

Algunos de los dispositivos más utilizados para implementar una RNA son los siguientes:

- VLSI Analógico
- Opto-electrónico
- **FPGA**
- Neuro-Chips (VLSI Digital)
- Neuro-Tarjetas
- Máquinas paralelas de propósito general
- Biochips

En este trabajo se implementará la red en un FPGA, precisamente por la característica de procesamiento en paralelo, como se explica en la siguiente sección.

1.4.2.7 FPGA

Un FPGA (*Field Programmable Gate Array* o campo de arreglo de compuertas programable) es un dispositivo compuesto de cientos o inclusive miles de bloques lógicos comunicados por conexiones programables.

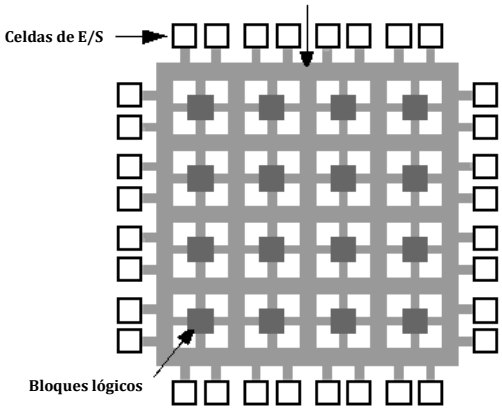


Fig. 1-7 Esquema básico de un FPGA (Fuente: Wikimedia Commons).

Este dispositivo es capaz de realizar tareas generalizadas o específicas. Su mayor ventaja es que puede realizar dichas tareas en paralelo. También tiene la capacidad de simular un circuito de aplicación específica [12].

Las aplicaciones donde más comúnmente se utilizan los FPGA incluyen a los sistemas aeroespaciales y de defensa, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora, procesamiento digital de señales, etc. Un FPGA puede remplazar a un micro-procesador en algunas aplicaciones, como en el caso de los satélites.

La estructura original de un FPGA se ve limitada al arreglo de compuertas básicas AND y OR. Es por ello que existen tarjetas de desarrollo basadas en FPGA que, por medio de periféricos y elementos adicionales (memorias RAM, tarjetas de audio y video, relojes externos, ente otros), complementan su estructura básica para realizar tareas más complejas como transmisión de datos, procesamiento de video, interfaces gráficas, o videojuegos. La siguiente figura muestra una tarjeta de este tipo.

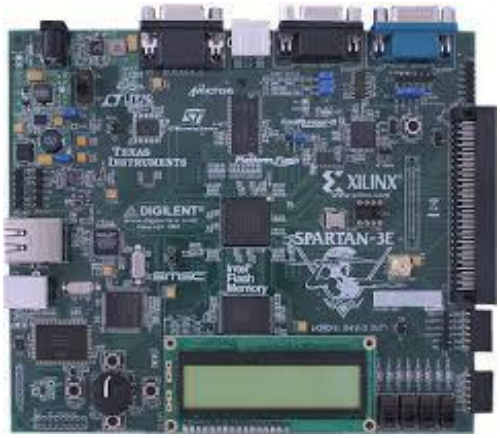


Fig. 1-8 Tarjeta de desarrollo Spartan 3E de Xilinx (Fuente: Digilent Inc.).

1.5 REDES NEURONALES APLICADAS AL ÁREA DE CONTROL

Las RNA aplicadas a control se pueden ver como un desarrollo alterno innovador en la metodología de control tradicional ante nuevos retos. Si observamos el creciente desarrollo en el área de control, es frecuente la necesidad de tratar con sistemas cada vez más complejos, de cumplir con los requerimientos de diseño demandantes y de realizar estos requerimientos con un conocimiento menos preciso de la planta y su entorno. Es decir, la necesidad de controlar bajo una creciente incertidumbre de la dinámica del sistema.

1.5.1 ANTECEDENTES

Existen diversos trabajos relacionados con las redes neuronales aplicadas en el área de control de sistemas. Parte de la motivación para usar las RNA es por la capacidad que tienen para implementar trayectorias no lineales, por su estructura paralela masiva y por su capacidad de adaptarse [13]. Usualmente se utilizan para aprender del sistema y poder predecir comportamientos futuros en la planta. Por otra parte algunos trabajos recientes, entre los años 2007 y 2012, muestran que un controlador convencional puede acompañarse de un compensador basado en redes neuronales, para mejorar el desempeño del controlador diseñado ante los cambios dinámicos y perturbaciones externas (Fig. 1-9 y Fig. 1-10). La unión de ambos métodos de control se realiza sumando las señales de control generadas por ambos algoritmos de control [14] [15].

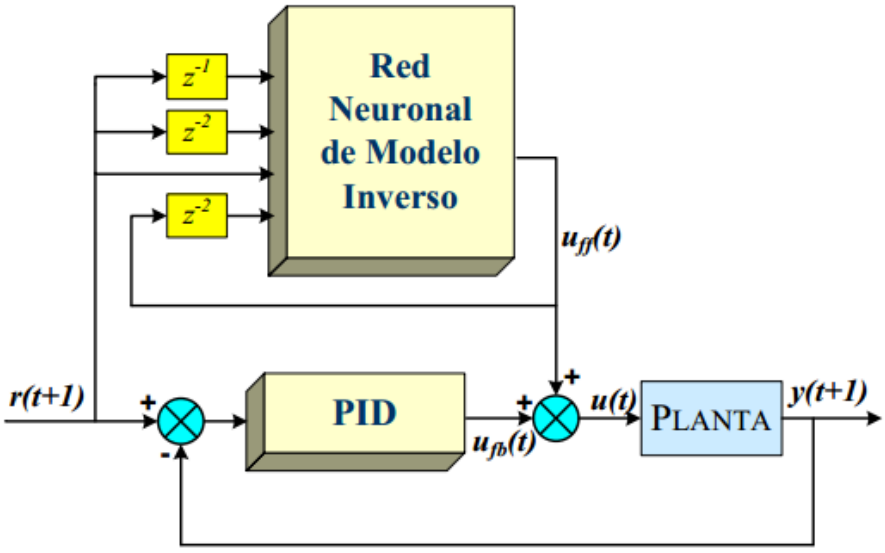


Fig. 1-9 Diagrama de bloques de un compensador neuronal con control PID [1][14].

[1] El símbolo z representa el estado de la lectura. En el caso de z^{-1}, se puede entender que es el valor previo y en z^{-2}, dos valores antes.

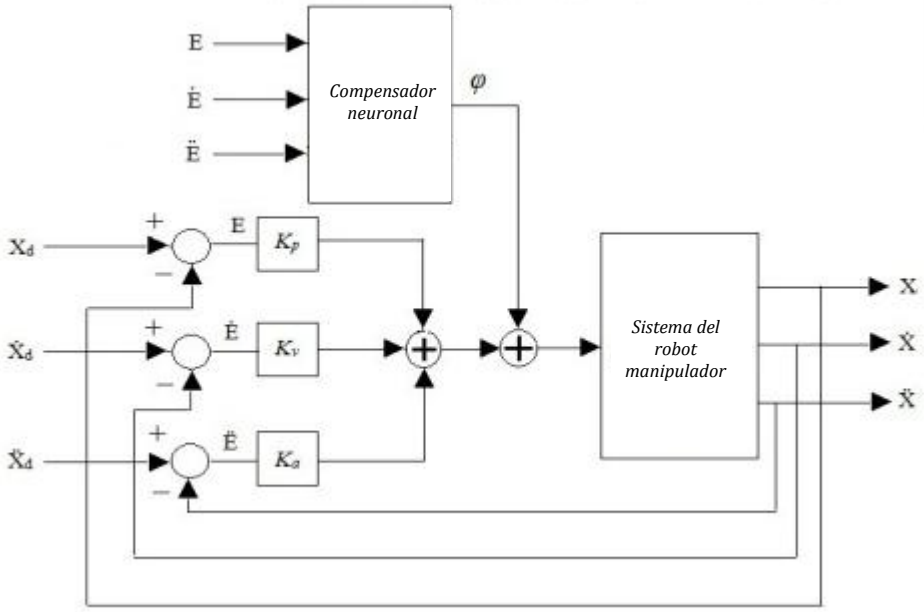


Fig. 1-10 Manipulador robótico antropomórfico con controlador PID y compensador neuronal [15].

En el caso de los motores de corriente directa se pueden encontrar trabajos donde las RNA son usadas para controlar par, posición o velocidad. Un trabajo similar al mostrado anteriormente presentado en el año 2008 por Andrei Cozma y Dan Pitica, llamado *Artificial Neural Network And PID Based Control System For DC Motor Drives* [16], encargado de implementar un sistema de control para motores de imán permanente. Su objetivo es mejorar el desempeño de un controlador de velocidad tipo PID entrenando una RNA con el algoritmo de retro-propagación a partir de valores experimentales.

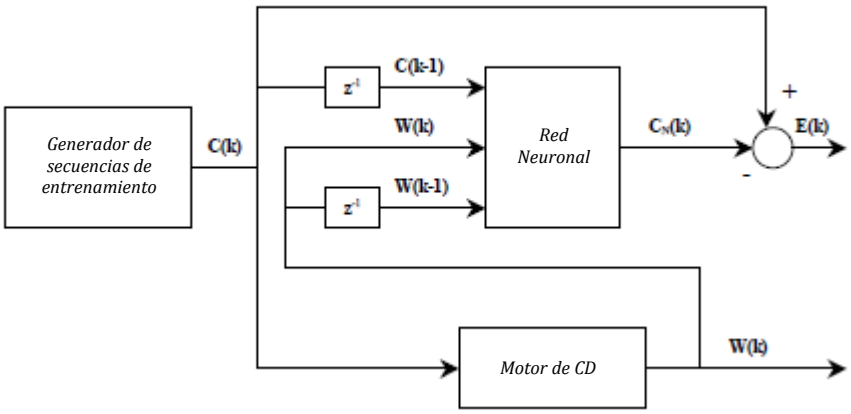


Fig. 1-11 Diagrama de entrenamiento de RNA [16].

La arquitectura de este sistema de control es más compleja, puesto que utilizan una red neuronal, entrenada por medio de un generador de valores de entrenamiento, para sustituir el controlador de velocidad complementado por un supervisor de posición. Los resultados muestran mejor desempeño en seguimiento

de la función de entrada que un PID tradicional. Este desempeño se aprecia en la respuesta ante el perfil de velocidad mostrado en el trabajo, puesto que la respuesta comparada (método de Ziegler-Nichols) presenta curvas en los vértices del perfil trapezoidal, mientras que el controlador neuronal hace que la respuesta adopte la forma trapezoidal [16].

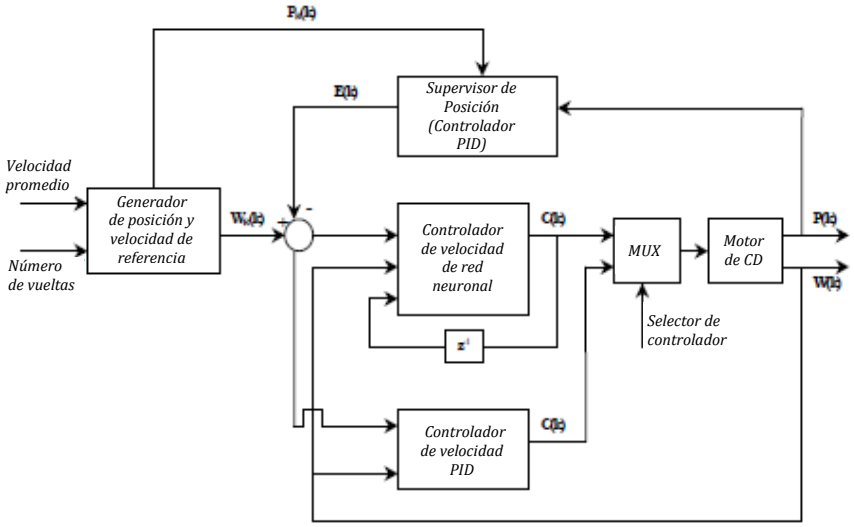


Fig. 1-12 Diagrama de bloques de neuro-controlador de velocidad [16].

Una aplicación destacable de las redes es en la sintonización de controladores PID. El controlador PID es muy utilizado en la industria por su versatilidad, sin embargo, uno de los principales inconvenientes es que la sintonía de sus parámetros es dependiente del diseñador y muchas veces resulta ser un método iterativo para encontrar una mejor respuesta del sistema. Un trabajo de este tipo fue presentado en el *6to. Congreso Nacional de Mecatrónica* (Noviembre, 2007), llamado *Control NeuroPID de un motor de CD de 180 V* [17], donde por medio del algoritmo de retro-propagación sintonizan los elementos de un PID de manera empírica (obsérvese la Fig. 1-13). Es de interés señalar que esta sintonización es automática, se lleva a cabo a partir del entrenamiento de las redes neuronales, el entrenamiento es en línea. Sin embargo, al realizar este tipo de sintonización se está sujeto al ajuste por parte de las redes neuronales y el diseño del PID resulta prescindible para controlar la planta. Desde el punto de vista de un diseño clásico de controladores, resulta poco práctico para sistemas críticos, donde sea importante tener definido algún parámetro, por ejemplo, el tiempo de sobrepaso. Una desventaja de este tipo de controlador es que al hacer el entrenamiento en línea se debe verificar que el proceso a realizar no afecte a otros subsistemas, pues en el proceso de entrenamiento se pueden tener salidas no deseadas, respuestas inesperadas, daños en elementos mecánicos, incluso encender el sistema cuando se desea que esté apagado.

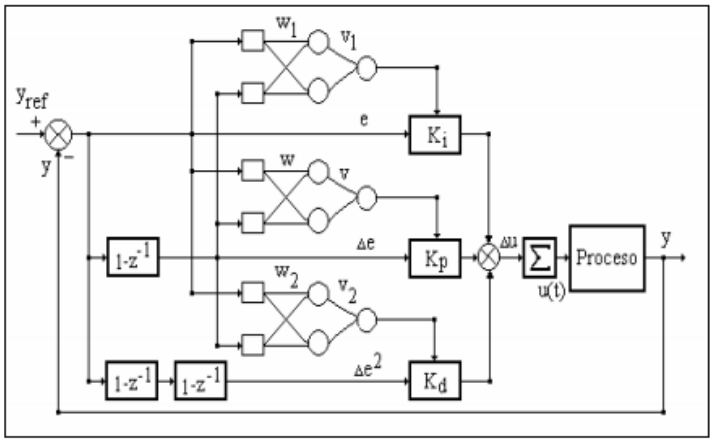


Fig. 1-13 Aplicación de las RNA como sintonizadores de los parámetros de un controlador PID.

La tesis publicada en el año 2009 por el Instituto Tecnológico de León titulada *Control de par de un motor de corriente directa mediante redes neuronales artificiales, a través de una PC* propone la sustitución de un controlador PI por uno basado en redes neuronales artificiales. Los resultados no fueron los esperados por el autor, en algunas pruebas responde la RNA mejor que el controlador PI, como en la prueba escalón, donde tiene una mejor respuesta ante el ruido presente en el motor y logra atenuar ligeramente mejor el pico inicial de corriente, pero en términos generales no es mejor que un PI tradicional, sobre todo al momento de aplicar una carga, la RNA responde pobremente ante ello. El autor justifica sus resultados no tan satisfactorios debido a las limitaciones de la paquetería de software utilizada, a la falta de tiempo para realizar más entrenamientos, a la arquitectura de la RNA, al tipo de prototipo utilizado y a la necesidad de filtrar la señal de entrada [18].

Actualmente hay una tendencia creciente a aplicar sistemas neuronales o neuro-difusos en el área de control y robótica, recurriendo a estos sistemas para optimizar los controladores convencionales y para compensar ciertos inconvenientes de éstos al momento de ser implementados o para hacerlos más robustos. Sin embargo, a pesar de encontrar mejor desempeño en algunos casos, falta mucho por hacer, ya que estos métodos alternativos requieren de un buen diseño de la red y de mejores métodos de entrenamiento [19]. La velocidad de convergencia en estos sistemas se vuelve crítica en el entrenamiento en línea y es por ello que el diseño de la red es importante, lo cual no es una tarea fácil, es un proceso heurístico. Por otra parte, el buen desempeño no se garantiza aunque se tenga un error mínimo en el entrenamiento, pueden encontrarse zonas con errores considerables. Otra tarea complicada es la de mejorar los dispositivos de aplicación, usualmente se utilizan sistemas electrónicos que permitan llevar a cabo procesos en paralelo como los amplificadores operacionales, computadoras, FPGA y los CPLD (Dispositivos Lógicos Programables Complejos), que son dispositivos formados de múltiples bloques lógicos, similares a los PLD (Dispositivos Lógicos Programables), donde cada bloque se comunica entre sí utilizando una matriz programable de interconexiones.

En el caso de grandes diseños, donde las redes neuronales son muy complejas y se requiere de bastante memoria interna se vuelve costosa la implementación. Empero, el desarrollo tecnológico en el área de hardware motiva a implementar las RNA en los dispositivos actuales, ya que se pueden encontrar tarjetas basadas en procesadores de alta velocidad o en FPGA de bajo costo.

1.6 ESQUEMA DE TRABAJO

El propósito de este trabajo es sustituir un controlador convencional por un controlador basado en redes neuronales y, en este caso, controlar la posición del eje del motor.

Para esto, se ha definido un esquema que nos permita cumplir el objetivo principal: comparar los controladores P y PD contra los controladores basados en RNA (llamados también controladores neuronales). A continuación se explica brevemente cada uno de los elementos.

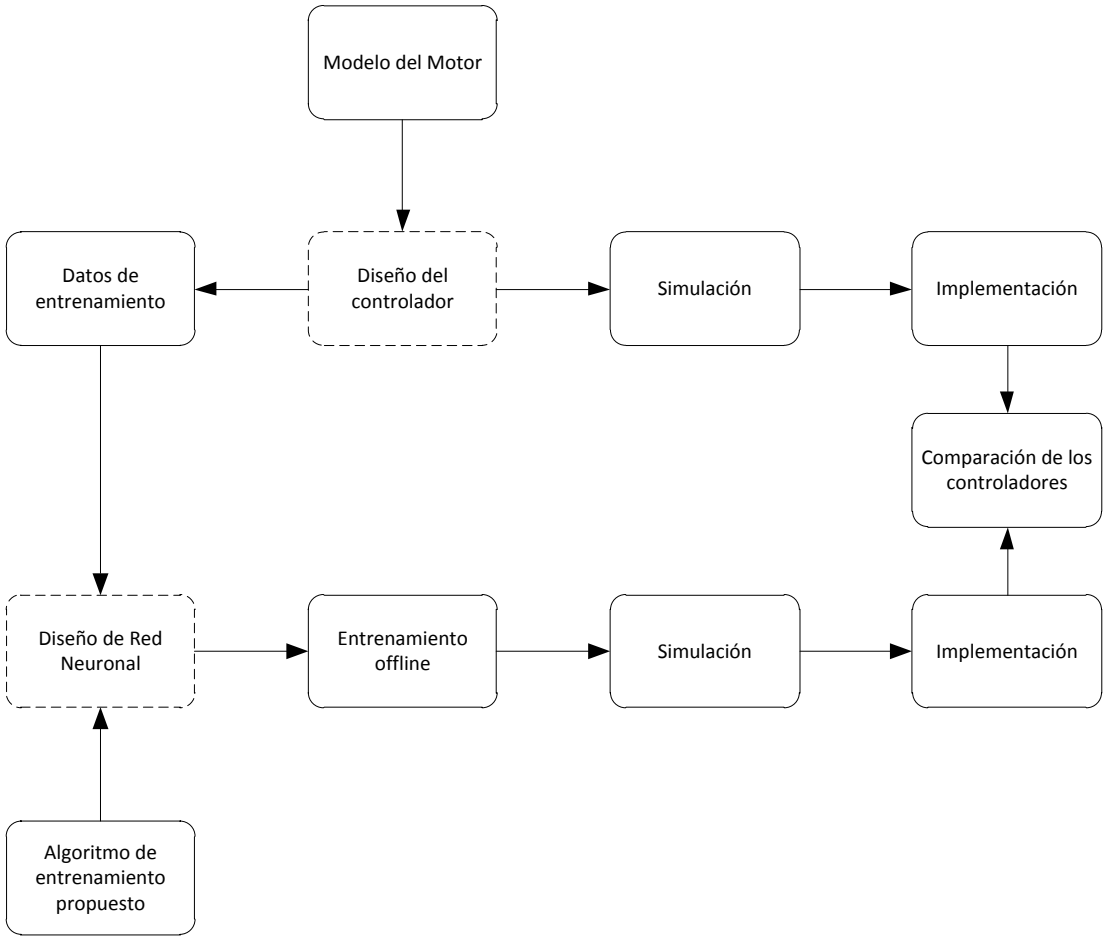


Fig. 1-14 Esquema del trabajo.

1. Diseño del controlador: Primeramente se diseñará un controlador tipo P y PD convencional para el motor de CD. Se simulará y se implementará. Los valores de entrada y de salida del controlador serán almacenados, éstos serán los datos de entrenamiento para la red neuronal artificial.

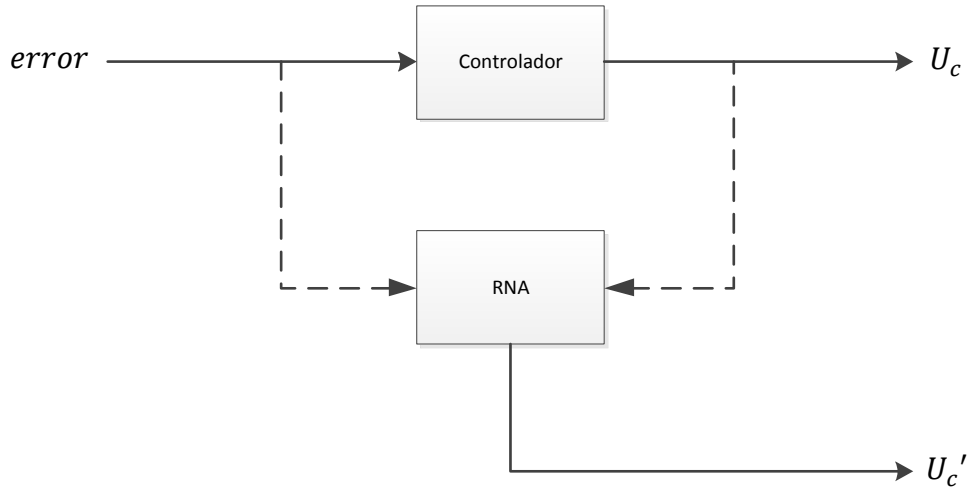


Fig. 1-15 Entrenamiento de la RNA.

2. Diseño de Red Neuronal Artificial: A partir de los datos de entrenamiento se diseña la RNA y se entrena por medio del algoritmo elegido. La nueva salida U_c' será utilizada para controlar, en la etapa final, la posición del rotor del motor de CD. El diagrama de bloques de control de este proceso se muestra en la figura siguiente:

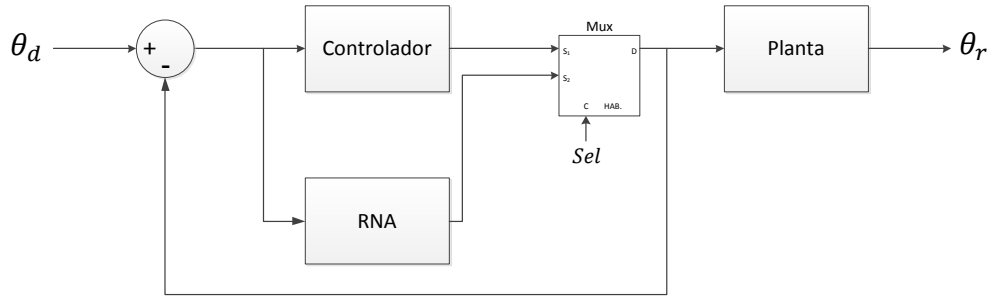


Fig. 1-16 Diagrama de bloques de la implementación de los controladores.

Con esta configuración se podrá alternar de controlador en la implementación. Por último, se almacenan los datos de ambos controladores y se comparan para corroborar la hipótesis propuesta.

2. MOTOR DE CORRIENTE DIRECTA

2.1 INTRODUCCIÓN AL MOTOR DE CD

Los motores de corriente directa o corriente continua, son utilizados en un sinfín de aplicaciones, de uso común y regularmente económicos.

Se puede definir el motor de corriente directa como una máquina que convierte energía eléctrica en mecánica. Además, esta máquina eléctrica tiene dos posibilidades de aplicación, se puede utilizar como motor o como generador. Este último aspecto es importante para el diseño de sistemas electrónicos ya que el motor puede regresar corriente por el hecho de ser generador, por ello es adecuado tener una separación entre los sistemas de control y de potencia. Cabe mencionar que los motores de CD son utilizados con frecuencia en la enseñanza de ingeniería.

El motor de CD se compone principalmente de dos partes:

- **Estator**: Es el soporte mecánico del motor, generalmente de forma cilíndrica y contiene un hueco en el centro. Se encuentran en el estator los polos del motor, los cuales pueden ser de imanes permanentes o devanados con hilo de cobre sobre núcleo de hierro.

- **Rotor**: Es el eje del motor, como su nombre lo dice es el componente que gira, usualmente de forma cilíndrica, también devanado y con núcleo magnético que gira dentro de un campo magnético creado por el estator.

En la vida cotidiana se pueden encontrar diversos sistemas que utilizan motores de corriente directa.

Ejemplos:

- Robots
- Ascensores
- Máquinas herramienta (máquinas CNC)
- Trenes de laminación (rodillos)
- Lectores de discos
- Carros a control remoto
- Autos eléctricos
- Ventiladores de computadora

2.2 MODELADO DEL MOTOR DE CD

En esta sección se presenta el modelado del motor de corriente directa, con el objetivo de obtener la función de transferencia (mostrada a continuación) que relaciona el voltaje de entrada con la posición del motor.

$$G(s) = \frac{\theta_m(s)}{E(s)}$$

Esta función de transferencia es utilizada para la simulación del controlador convencional. En el siguiente capítulo se describe a detalle la etapa de simulación.

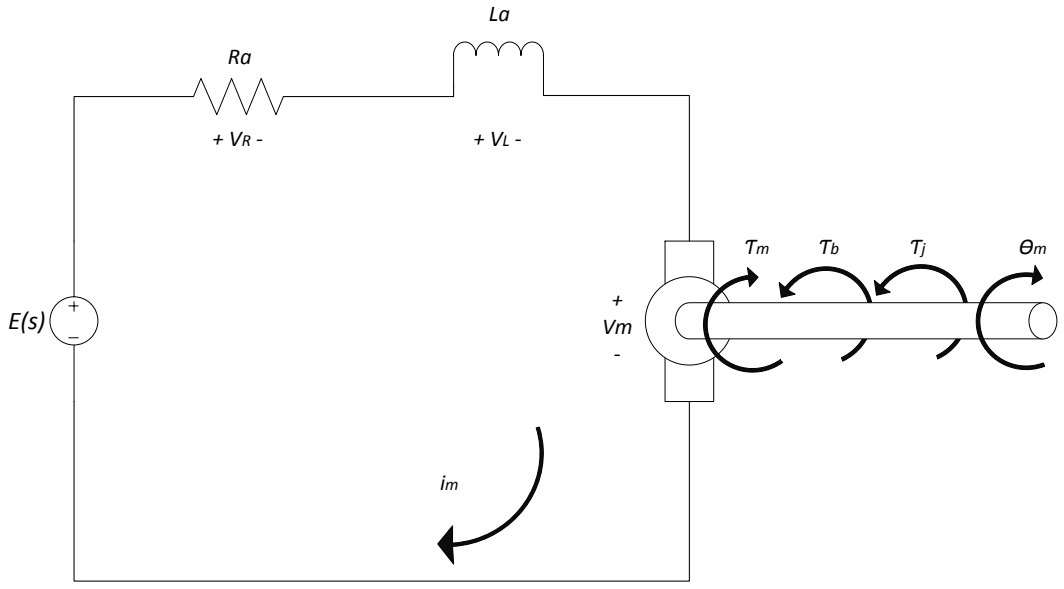


Fig. 2-1 Circuito equivalente del motor de CD.

Los elementos del modelo del motor de CD son descritos en la siguiente tabla.

Tabla 2-1 Elementos del modelado.

Variable	Descripción	Unidades
$E(s)$	Voltaje de entrada (Fuente de tensión)	[V]
R_a	Resistencia de armadura	[Ω]
L_a	Inductancia de armadura	[H]
V_R	Voltaje de la resistencia de armadura	[V]
V_L	Voltaje de la inductancia de armadura	[V]
i_a	Intensidad de corriente en la armadura	[A]
V_m	Fuerza contraelectromotriz (contra FEM)	[V]
τ_m	Par de torsión generado en el rotor del motor	[N m]
J_m	Inercia rotacional del rotor	[kg m^2]
b_m	Coeficiente de fricción rotacional del motor	[N m s/rad]
k_b	Constante de fuerza contraelectromotriz	[V s/rad]
k_t	Constante de par del motor	[N m/A]

Las ecuaciones que forman parte de las relaciones constitutivas del motor son presentadas tanto en el dominio del tiempo, como en el dominio de la frecuencia, después se sustituyen estas ecuaciones en las ecuaciones de equilibrio que representan las relaciones físicas del sistema.

Relaciones constitutivas

El modelo del motor consiste básicamente en una resistencia, conectada en serie con un inductor y el rotor del motor, los cuales corresponden a la *resistencia de armadura, inductancia de armadura y rotor*.

Por ser un circuito en serie se tiene que la intensidad de corriente a través de éste es la misma.

Tabla 2-2 Relaciones constitutivas del motor.

Dominio del tiempo		Dominio de la frecuencia	
$V_R = R_a\, i_a$	(2.1)	$V_R(s) = R_a\ I_a(s)$	(2.2)
$V_L = L_a\, \frac{d}{dt} i_a$	(2.3)	$V_L(s) = L_a\, s\ I_a(s)$	(2.4)
$V_m = k_b\, \frac{d}{dt} \theta_m$	(2.5)	$V_m(s) = k_b\, s\ \theta_m(s)$	(2.6)
$\tau_m = k_t\, i_a$	(2.7)	$\tau_m(s) = k_t\ I_a(s)$	(2.8)

Ecuaciones de equilibrio

$$E(t) = V_R + V_L + V_m \tag{2.9}$$

$$\tau_m = \tau_J + \tau_b \tag{2.10}$$

Donde τ_J es el par inercial generado por la masa del rotor y τ_b es el par generado por la fricción rotacional del motor, donde:

$$\tau_J = J_m \frac{d^2}{dt^2} \theta_m \tag{2.11}$$

$$\tau_b = b_m \frac{d}{dt} \theta_m \tag{2.12}$$

Por lo tanto la ecuación (2.10), la cual indica el par de torsión generado en el rotor[2], puede reescribirse como:

$$\tau_m = J_m \frac{d^2}{dt^2} \theta_m + b_m \frac{d}{dt} \theta_m \tag{2.13}$$

[2] Es importante mencionar que debería considerarse otro par causado por alguna carga presentada en el motor, pero para la aplicación de este trabajo se puede despreciar, puesto que sólo es de interés revisar la posición del rotor.

Transformando las ecuaciones de equilibrio al dominio de la frecuencia tenemos:

$$E(s) = V_R(s) + V_L(s) + V_m(s) \tag{2.14}$$

$$\tau_m = (J_m\, s^2 + b_m\, s)\, \theta_m(s) \tag{2.15}$$

Reescribiendo la ecuación sustituyendo los valores de cada uno de los voltajes se obtiene la siguiente expresión:

$$E(s) = R_a\ I_a(s) + L_a\, s\ I_a(s) + k_b\ s\ \theta_m(s) \tag{2.16}$$

Que se puede simplificar de la siguiente manera:

$$E(s) = (R_a + L_a\, s)\ I_a(s) + k_b\ s\ \theta_m(s) \tag{2.17}$$

Esta ecuación se debe expresar en términos de $\theta_m(s)$. De la ecuación (2.8) se despeja el valor de $I_a(s)$ y se sustituye en la ecuación (2.17).

$$E(s) = \frac{(R_a + L_a\, s)\ \tau_m(s)}{k_t} + k_b\ s\ \theta_m(s) \tag{2.18}$$

Sustituyendo la ecuación (2.15) en la ecuación (2.18) se obtiene:

$$E(s) = \frac{(R_a + L_a\, s)\ (J_m\, s^2 + b_m\, s)\, \theta_m(s)}{k_t} + k_b\ s\ \theta_m(s) \tag{2.19}$$

Sin embargo, usualmente el valor de la inductancia de armadura de un motor de cd es muy pequeña comparada con el valor de la resistencia de armadura, prácticamente cero, por lo cual la ecuación anterior se reduce a:

$$E(s) = \left(\frac{R_a\ (J_m\, s^2 + (b_m + k_b\ k_t)s)}{k_t}\right) \theta_m(s) \tag{2.20}$$

Dejando la ecuación (2.20) en la forma de $G(s)$ se puede expresar como:

$$\frac{\theta_m(s)}{E(s)} = \frac{1}{\frac{R_a J_m}{k_t} s^2 + \left(\frac{b_m R_a + k_b\ k_t}{k_t}\right) s} \tag{2.21}$$

Al sustituir los valores de los parámetros del motor de la Tabla 2-3 en la ecuación (2.21) obtenemos la función de transferencia de motor siguiente:

$$\frac{\theta_m(s)}{E(s)} = \frac{1}{0.00022732 s^2 + 0.0311 s} \tag{2.22}$$

2.3 MOTOR UTILIZADO Y SUS PARÁMETROS

El motor seleccionado fue el Pittman 14204 12.0 V, sus características se describen en la tabla. Se buscaba en particular un motor comercial con *encoder* para ahorrar tiempo en la implementación, puesto que el objetivo principal de este trabajo es el control de posición utilizando una RNA y no la selección del motor.

Tabla 2-3 Parámetros del motor Pittman (Fuente: Pittman Motors).

Motor	14204 12.0V	Unidades
Voltaje de alimentación	12.0	[V]
Constante del par de torsión	0.031	[N m / A]
Constante de voltaje	0.031	[V/rad/s]
Resistencia de armadura	0.27	[Ω]
Inductancia	0.40	[mH]
Factor de fricción viscosa	1.21×10^{-5}	[N m s/rad]
Inercia del rotor	2.61×10^{-5}	[kg m^2]

Fig. 2-2 Motor Pittman 14204 (Fuente: Automation Express).

3. DISEÑO DEL CONTROLADOR DE POSICIÓN DEL MOTOR DE CD

En el capítulo anterior se describió el modelado del motor, donde se obtuvo la siguiente función de transferencia:

$$G(s) = \frac{1}{0.00022732s^2 + 0.0311s} \tag{3.1}$$

Con el propósito de demostrar la capacidad de aprendizaje de las RNA se diseñaron 2 tipos de controladores, P y PD. Se especifican, a continuación, los parámetros de diseño y al final de este trabajo se compararán las respuestas de estos controladores contra los neuronales.

Con el propósito de agilizar el diseño de controladores se reescribe la ecuación (2.21) de la siguiente manera:

$$\frac{\theta_m(s)}{E(s)} = \left(\frac{k_t}{R_a J_m}\right)\frac{1}{s^2 + \left(\frac{b_m R_a + k_b k_t}{R_a J_m}\right)s} \tag{3.2}$$

Que es de la forma:

$$\frac{\theta_m(s)}{E(s)} = g\frac{1}{s^2 + as} \tag{3.3}$$

Donde:

$$g = \frac{k_t}{R_a J_m} \tag{3.4}$$

$$a = \frac{b_m R_a + k_b k_t}{R_a J_m} \tag{3.5}$$

Sustituyendo los valores de los parámetros del motor en las ecuaciones (3.4) y (3.5) se obtienen los siguientes valores: $g = 4399.035$ y $a = 136.8334$

3.1 CONTROLADOR PROPORCIONAL

El controlador proporcional de posición forma parte de la primera etapa de entrenamiento.

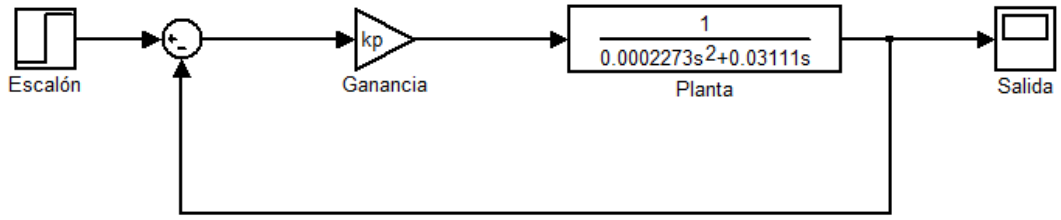


Fig. 3-1 Diagrama de bloques del controlador proporcional

Dada la siguiente función de transferencia de lazo cerrado:

$$H_1 = \frac{gk_p}{s^2 + as + gk_p} \tag{3.6}$$

Se obtuvieron los parámetros del controlador tomando como criterio de diseño el porcentaje de sobrepaso para una ecuación característica de segundo orden:

$$s^2 + 2\xi\omega_n s + {\omega_n}^2 = 0 \tag{3.7}$$

Donde:

$$\xi = \frac{-\ln\frac{\%Sp}{100}}{\sqrt{\pi^2 + \left(\ln\frac{\%Sp}{100}\right)^2}} \tag{3.8}$$

$$\omega_n = \frac{a}{2\xi} \tag{3.9}$$

$$k_p = \frac{{\omega_n}^2}{g} \tag{3.10}$$

Finalmente se sustituyen los valores de a, g, los porcentajes de sobrepaso y el tiempo de asentamiento propuestos en las últimas cuatro ecuaciones para obtener los valores de las constantes del controlador.

La Tabla 3-1 muestra los distintos valores de la constante proporcional para cuatro porcentajes de sobrepaso propuestos, así como los valores máximos de la posición esperados y los valores máximos obtenidos en la simulación del controlador.

Tabla 3-1 Distintos valores de sobrepaso y valores característicos.

$\%Sp$	ξ	ω_n	k_p	Valor máximo de la posición [rad]	Valor máximo de la posición en la simulación [rad]
2.5	0.7613	89.8657	1.835821	0.8050	0.8050
5	0.6901	99.1395	2.234272	0.8247	0.8246
7.5	0.6362	107.5471	2.6293	0.8443	0.8442
10	0.591155	115.73418	3.04485	0.8639	0.8638

Las simulaciones hechas para este controlador se realizaron con una entrada escalón de 45 grados (π/4 radianes) por medio de Simulink®. De la tabla anterior se puede apreciar que los valores de la posición máximos obtenidos en la simulación tienen un error relativo muy pequeño, de $1x10^{-4}$ radianes comparado con los valores teóricos esperados. La excepción es para el valor donde la constante proporcional vale 1.835821, la posición del rotor máxima es la misma que la esperada. Se puede concluir de estos valores, que los porcentajes de sobrepaso son respetados, a pesar de los pequeños errores relativos, causados seguramente por el método utilizado y las aproximaciones numéricas de Simulink® para resolver el sistema (Runge-Kutta).

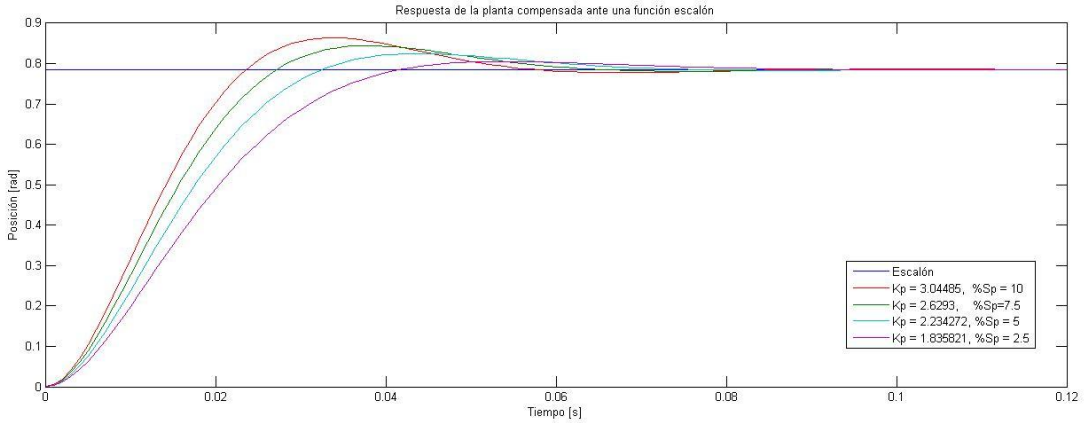


Fig. 3-2 Respuestas del sistema debido a los controladores proporcionales.

El valor de la constante *kp* = 1.835821 es seleccionado para la etapa de pruebas. Para los alcances de este trabajo es suficiente realizar las pruebas para un solo valor, puesto que el objetivo principal es imitar la respuesta de un controlador por medio de una RNA. Por lo tanto, la realización de pruebas para otras constantes proporcionales es adicional. La elección de la *kp* más pequeña permite prevenir oscilaciones en la respuesta del sistema en el caso de una respuesta mayor no deseada, es decir, algún sobrepaso mayor. Tal sobrepaso podría generarse por un retraso en la respuesta del controlador neuronal causado por el número de cálculos que se tengan que efectuar o provocado por la actualización de datos.

3.2 CONTROLADOR PROPORCIONAL - DERIVATIVO

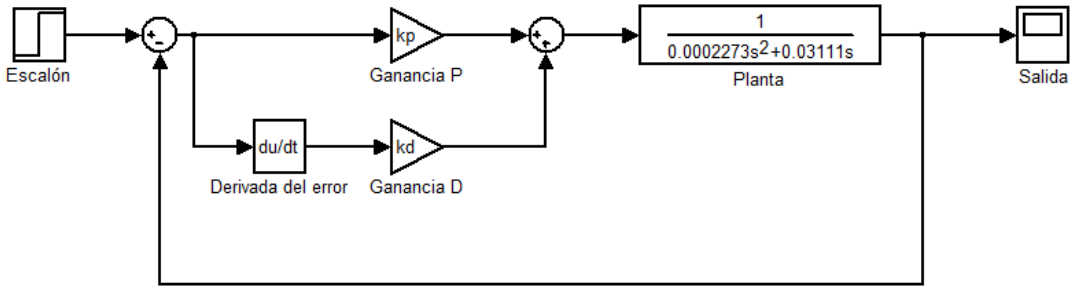


Fig. 3-3 Diagrama de bloques del controlador proporcional derivativo.

El controlador será un PD de la forma:

$$C_2 = K_p + K_d s \tag{3.11}$$

$$G = g\frac{1}{s^2 + as} \tag{3.12}$$

Función de transferencia de lazo cerrado:

$$H_1 = \frac{gK_p + gK_d s}{s^2 + (a + gK_d)s + gK_p} \tag{3.13}$$

Tenemos entonces la siguiente ecuación característica de segundo orden:

$$s^2 + (a + gK_d)s + gK_p = 0 \tag{3.14}$$

Que es de la forma:

$$s^2 + 2\xi\omega_n s + {\omega_n}^2 = 0 \tag{3.15}$$

Donde:

$$\xi = \frac{-\ln\frac{\%Sp}{100}}{\sqrt{\pi^2 + \left(\ln\frac{\%Sp}{100}\right)^2}} \tag{3.16}$$

$$\omega_n = \frac{4}{\xi\, T_s} \tag{3.17}$$

$$K_d = \frac{2\xi\omega_n - a}{g} \tag{3.18}$$

$$k_p = \frac{{\omega_n}^2}{g} \tag{3.19}$$

Finalmente se sustituyen los valores de a, g, los porcentajes de sobrepaso y el tiempo de asentamiento propuestos en las últimas cuatro ecuaciones para obtener los valores de las constantes del controlador.

La tabla siguiente, al igual que para el controlador proporcional, muestra los diferentes valores de las constantes proporcionales y derivativas obtenidas para los distintos porcentajes de sobrepaso propuestos. El tiempo de asentamiento (T_s) planteado es de 50 milisegundos.

Tabla 3-2 Características de los controladores PD.

$\%Sp$	ξ	T_s	ω_n	k_p	k_d	Valor máximo de la posición [rad]	**Simulación**		
							Valor máximo de la posición [rad]	$\%Sp$	T_s
2.5	0.7613	0.05	105.0802	2.510061	0.005266	0.8050	0.8043	2.41	0.0530
5	0.6901	0.05	115.9241	3.054851	0.005266	0.8247	0.8237	4.88	0.0515
7.5	0.6362	0.05	125.7551	3.59496	0.005266	0.8443	0.8432	7.36	0.0475
10	0.591155	0.05	135.3283	4.163128	0.005266	0.8639	0.8628	9.86	0.0435

La Fig. 3-4 muestra la respuesta de cada controlador diseñado ante una función escalón de 45 grados (π/4 radianes).

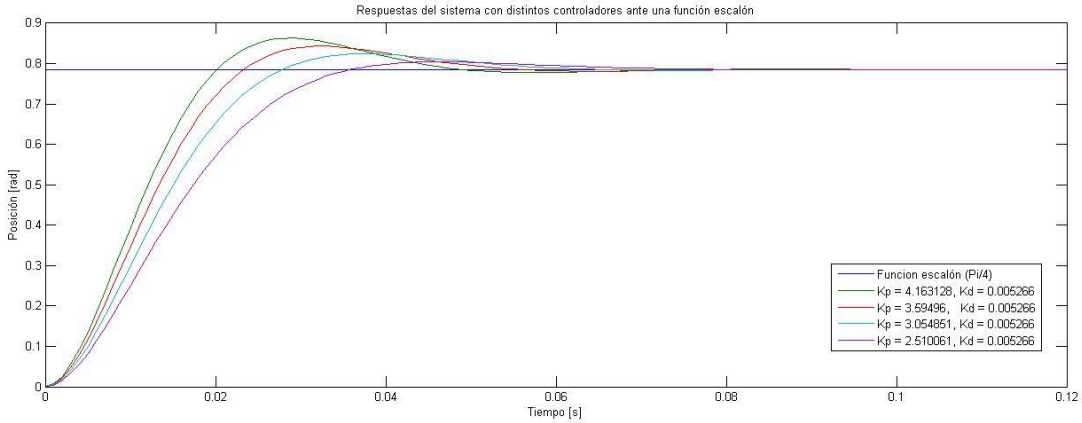


Fig. 3-4 Respuestas del sistema con los controladores PD (%SP = 2.5, 5, 7.5, 10 y T_s = 0.05).

De la Tabla 3-2 se toman los valores de *kd* y *kp* para un porcentaje de sobrepaso del 2.5% para la etapa de pruebas. Al igual que en la sección anterior, el criterio de selección de las constantes se basa en la prevención de oscilaciones causadas por la actualización de datos en el programa de implementación, recordando que aún no se sabe el tiempo que tardarán en ejecutarse los cálculos de la red neuronal. Por otra parte, se puede observar en la misma tabla que los porcentajes de sobrepaso y los tiempos de asentamiento simulados de la respuesta del sistema son diferentes a los diseñados teóricamente, esto se debe al efecto del cero agregado por medio del controlador PD. Esto se puede verificar analizando el *root-locus*. Esta diferencia en los sobrepasos no es crítica para fines de este trabajo.

4. DISEÑO DE LA RED NEURONAL ARTIFICIAL

Este capítulo trata acerca del diseño de la red neuronal artificial, la cual imitará el comportamiento de los controladores estudiados anteriormente. Los pasos de diseño propuestos [20] son los siguientes:

1. Recolectar datos: Una RNA requiere de cierta cantidad de datos para poderse entrenar, depende de la aplicación de la red y, en general, el manejo de éstos es dependiente de la experiencia del diseñador. Mientras más datos se tengan más tiempo tarda en entrenarse la red, este criterio es importante para determinar la cantidad de patrones de entrenamiento.

2. Proponer arquitectura de la red: La arquitectura de la red se establece una vez analizado el problema a resolver. En el caso de las redes multi-capa, siempre es difícil establecer el tamaño de las capas. Sólo la experiencia puede ayudar a definir el tamaño para la capa intermedia en función del número de neuronas de la capa de entrada, para que la red aprenda de forma rápida [21]. Empero, aprender rápido no implica aprender bien, puede tener una convergencia rápida pero no una buena generalización.

3. Configurar la red: Una vez obtenida la arquitectura propuesta se configuran sus parámetros como los valores iniciales de los pesos y sesgos, número de capas, valores iniciales de entrada.

4. Entrenar la red: En esta etapa se implementa el algoritmo de entrenamiento con la finalidad de que la RNA obtenga los valores de los pesos y sesgos que permitan aplicar la red en el sistema de análisis.

5. Validar la red: Se le llama validación de la red a la etapa donde se determina que una red está lista para su aplicación, esto sucede al satisfacer una condición de entrenamiento, en la literatura se señala la utilización el valor del error de convergencia como criterio de validación.

6. Utilizar la red: El último paso es la aplicación de la red, ya sea en un banco de pruebas, en simulación por software o directo sobre el sistema. En esta etapa se puede apreciar si la red tiene una buena generalización o está sobreentrenada.

4.1 ALGORITMO DE RETRO-PROPAGACIÓN

El algoritmo de retro-propagación o *BP*, por sus siglas en inglés, es uno de los más utilizados en el campo de las redes neuronales artificiales, sirve para entrenar redes del tipo perceptrón multicapa. Debido a que una RNA contiene elementos no lineales, tiene la capacidad de implementar prácticamente cualquier función continua. Este algoritmo es una generalización del algoritmo LMS (Least Mean Square), el cual busca obtener el error mínimo del cuadrado de la señal del error [10].

El Algoritmo BP consta de dos etapas:

1. Propagación hacia adelante: Se fijan los parámetros de la red y se presenta una señal de entrada a la red, la cual se propaga hacia adelante para producir el valor de salida.

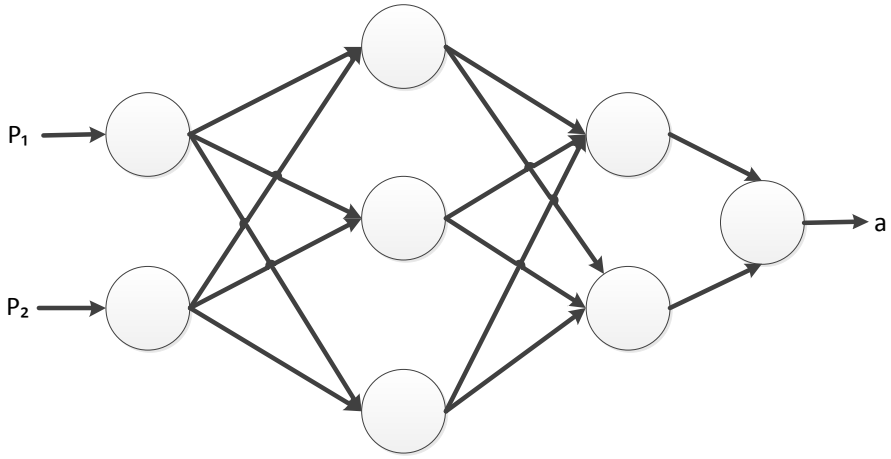


Fig. 4-1 Propagación hacia adelante.

2. Propagación hacia atrás: En esta etapa se obtiene el error o diferencia entre el valor de salida deseado y el valor de la red. Este error se propaga hacia atrás con la finalidad de actualizar los pesos y sesgos de la red.

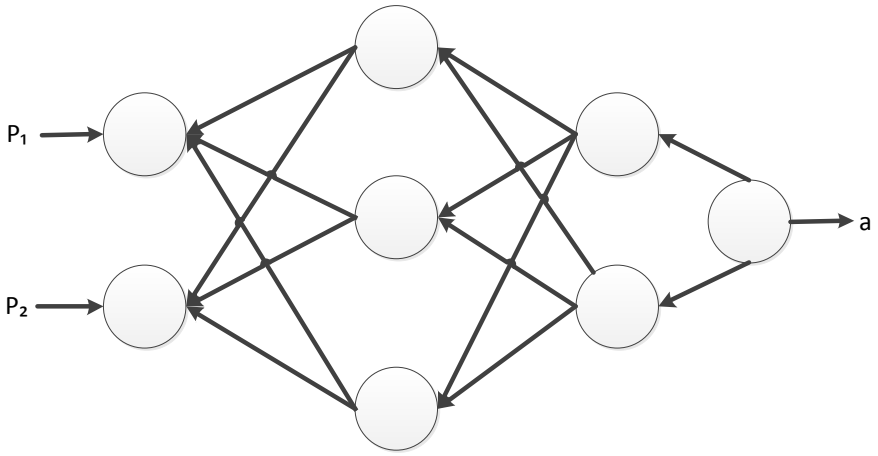


Fig. 4-2 Propagación hacia atrás.

Fig. 4-3 Diagrama de flujo del algoritmo de retro-propagación.

RESUMEN DEL ALGORITMO

A continuación se muestran las acciones tomadas, con base en la Fig. 4-3, en la etapa de simulación de la RNA. La paquetería de software utilizada para este desarrollo fue MATLAB®.

1. Inicialización: En este proceso se determinó la estructura de la red y los valores iniciales de los pesos y sesgos. La literatura dice que cualquier valor inicial permite la convergencia de la red. Sin embargo, se recomienda emplear valores pequeños, la mayoría de los autores recomienda que éstos sean en un rango entre 0.5 y 1. En el caso de la simulación se usarán valores aleatorios.

   Existen pocos criterios de diseño para establecer la estructura de la red y se basan en la experiencia de los diseñadores, no hay algoritmo para su determinación. Autores, como Hagan, consideran el cambio de pendiente como criterio para agregar una neurona en la capa oculta [10]. Otros autores mencionan que una RNA de 4 capas puede imitar cualquier función sin problemas. Un criterio importante es el teorema de aproximación universal [7], el cual sostiene que una sola capa oculta es suficiente para que un perceptrón multicapa logre una aproximación satisfactoria de cualquier transformación continua. Sin embargo, no menciona que tal configuración sea óptima en el sentido del tiempo de aprendizaje. Con esto se puede entender que sea posible que se entrene más rápido una red con más capas y con menos neuronas que una red de una sola capa y con muchas neuronas.

2. Normalización de datos: Este paso no es obligatorio, pero es recomendable cuando se tienen datos con valores mayores del rango de la función de activación.

   Para normalizar los datos se hizo un mapeo (interpolación) por medio de la siguiente ecuación:

$$y = \frac{(x - x_0)(y_1 - y_0)}{(x_1 - x_0)} + y_0 \qquad (4.1)$$

   Donde:

   x: *el número a mapear*
   x_0: *el límite inferior del rango actual del valor*
   y_0: *el límite superior del rango actual del valor*
   x_1: *límite inferior del rango deseado*
   y_1: *límite superior del rango deseado*
   y: *el valor mapeado*

3. Presentar un patrón: Una vez listos los datos para su procesamiento, se presenta un par de entrenamiento (entrada(s), salida(s)) para ejecutarse el algoritmo de entrenamiento.

4. Propagación hacia adelante: En este proceso se propagan los valores de las neuronas hacia adelante, es decir, de la primera capa hasta la última, con el objeto de obtener el valor de salida de la red. Las fórmulas utilizadas para realizar esto son las siguientes:

$$\boldsymbol{a}^0 = \boldsymbol{p} \tag{4.2}$$

$$\boldsymbol{a}^{m+1} = \boldsymbol{f}^{m+1}(\boldsymbol{W}^{m+1}\,\boldsymbol{a}^m + \boldsymbol{b}^{m+1}) \tag{4.3}$$

$para\ m = 0, 1, \ldots, M-1$

$$\boldsymbol{a} = \boldsymbol{a}^M \tag{4.4}$$

Donde M es el número de capas de la RNA.

5. Cálculo del error: Este cálculo está dado por la fórmula del error cuadrático medio, el cual se puede encontrar en el libro *Neural Networks Design* de Martin Hagan [10].

$$\boldsymbol{E} = mse = \frac{1}{N}\sum_{i=1}^{N}(e_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(t_i - a_i)^2 \tag{4.5}$$

Donde N es el número de pares de ejemplos.

El error es:

$$e_i = t_i - a_i$$

6. Propagación hacia atrás: Una vez que se tiene el cálculo del error se procede a propagarlo hacia atrás, es decir, hasta la primera capa de la red neuronal. Este proceso le da el nombre al algoritmo de retro-propagación.

Los valores que se propagan hacia atrás se conocen como *sensibilidades*. Éstos son utilizados para actualizar los pesos, se puede observar que en el caso de que el error sea cero, también los valores de las *sensibilidades* lo son.

$$\boldsymbol{S}^M = -2\dot{\boldsymbol{F}}^M(\boldsymbol{n}^M)(\boldsymbol{t} - \boldsymbol{a}) \tag{4.6}$$

$$\boldsymbol{S}^m = \dot{\boldsymbol{F}}^m(\boldsymbol{n}^m)(\,\boldsymbol{W}^{m+1})^{\boldsymbol{T}}\,\boldsymbol{S}^{m+1} \tag{4.7}$$

Se puede apreciar en las ecuaciones anteriores que se incluyen las derivadas de las funciones de activación de las capas ocultas *m* y para la última capa *M*. Las funciones seleccionadas para este diseño fueron las siguientes:

$$\boldsymbol{F}^{M}(\boldsymbol{n}^{M}) = \boldsymbol{a}^{M} = \boldsymbol{n}^{M} \tag{4.8}$$

$$\boldsymbol{F}^{\boldsymbol{m}}(\boldsymbol{n}^{\boldsymbol{m}}) = \boldsymbol{a}^{m} = \frac{1}{1+e^{-n}} \tag{4.9}$$

Y sus derivadas son:

$$\dot{\boldsymbol{F}}^{M}(\boldsymbol{n}^{M}) = \mathbf{1} \tag{4.10}$$

$$\dot{\boldsymbol{F}}^{m}(\boldsymbol{n}^{m}) = (1-\boldsymbol{a}^{m})(\boldsymbol{a}^{m}) \tag{4.11}$$

7. Actualización de los métodos de la red: Una vez calculadas las *sensibilidades* se actualizan los parámetros de la red. A continuación se muestran las ecuaciones de actualización de pesos y sesgos.

$$\boldsymbol{W}^{m}(k+1) = \boldsymbol{W}^{m}(k) + \Delta\boldsymbol{W}^{m}(k) \tag{4.12}$$

$$\boldsymbol{b}^{m}(k+1) = \boldsymbol{b}^{m}(k) + \Delta\boldsymbol{b}^{m}(k) \tag{4.13}$$

Los valores de los términos que contienen la nomenclatura Δ provienen del algoritmo de entrenamiento, estos se especifican en la siguiente sección de este capítulo.

Una vez actualizados se repite el ciclo desde el paso 3 hasta que se obtenga el valor del error mínimo considerado.

4.2 ALGORITMOS DE ENTRENAMIENTO DE LA RED

Durante la etapa de simulación de este trabajo fueron revisadas tres reglas de aprendizaje: regla delta generalizada, retro-propagación con momentum y retro-propagación Levenberg-Marquardt. Las últimas dos modifican el algoritmo de retro-propagación. Sin embargo, esta modificación sólo es en la actualización de los pesos.

4.2.1 REGLA DELTA GENERALIZADA

La regla delta generalizada es el algoritmo de entrenamiento utilizado en el algoritmo de retro-propagación simple. Las funciones presentadas a continuación se utilizan para actualizar los pesos y sesgos una vez que se ha calculado el error.

Sensibilidades

$$\boldsymbol{S}^{M} = -2\dot{\boldsymbol{F}}^{M}(\boldsymbol{n}^{M})(\boldsymbol{t}-\boldsymbol{a}) = -2(\boldsymbol{t}-\boldsymbol{a}) \tag{4.14}$$

$$\boldsymbol{S}^{m} = (1-\boldsymbol{a}^{m})(\boldsymbol{a}^{m})(\,\boldsymbol{W}^{m+1})^{\boldsymbol{T}}\,\boldsymbol{S}^{m+1} \tag{4.15}$$

Actualización de Pesos y Sesgos

$$\boldsymbol{W}^{m}(k+1) = \boldsymbol{W}^{m}(k) - \alpha\boldsymbol{S}^{m}\,(\boldsymbol{a}^{m-1})^{T} \tag{4.16}$$

$$\boldsymbol{b}^{m}(k+1) = \boldsymbol{b}^{m}(k) - \alpha\boldsymbol{S}^{m} \tag{4.17}$$

El término α es conocido como la razón de aprendizaje, este valor se puede ajustar de manera iterativa o se puede fijar. La literatura recomienda utilizar valores muy pequeños, menores a 1, debido a que mientras más grande sea el valor de la razón de aprendizaje pueden oscilar los valores de actualización de manera indeterminada. Existen algunos criterios para definir este valor, ya sea utilizando reglas basadas en el gradiente de la función del error o simplemente dejando fijo el valor desde el principio. Este último criterio es el que fue utilizado para la etapa de simulación.

4.2.2 RETRO-PROPAGACIÓN CON MOMENTUM

Este algoritmo es una variación heurística de la regla delta, donde se agrega el término γ, conocido como el coeficiente de momentum, el cual debe satisfacer lo siguiente:

$$0 \leq \gamma < 1$$

Esta modificación proviene de la intención de suavizar las oscilaciones en la trayectoria de convergencia de la red. Para ello este algoritmo utiliza un filtro de primer orden paso-bajas, de la siguiente forma:

$$y(k) = \gamma y(k-1) + (1-\gamma)w(k) \tag{4.18}$$

Haciendo una analogía con las ecuaciones de pesos y sesgos se tiene lo siguiente:

$$\Delta \boldsymbol{W}^m(k) = \gamma \Delta \boldsymbol{W}^m(k-1) - (1-\gamma)\alpha \boldsymbol{S}^m \, (\boldsymbol{a}^{m-1})^T \tag{4.19}$$

$$\Delta \boldsymbol{b}^m(k) = \gamma \Delta \boldsymbol{b}^m(k-1) - (1-\gamma)\alpha \boldsymbol{S}^m \tag{4.20}$$

Finalmente, se pueden expresar, de manera reducida, las siguientes ecuaciones:

$$\boldsymbol{W}^m(k+1) = \boldsymbol{W}^m(k) + \Delta \boldsymbol{W}^m(k) \tag{4.21}$$

$$\boldsymbol{b}^m(k+1) = \boldsymbol{b}^m(k) + \Delta \boldsymbol{b}^m(k) \tag{4.22}$$

Es importante mencionar que al tenerse dos coeficientes (de aprendizaje y momentum) se hace más complicado determinar ambos valores, suelen utilizarse valores arriba de 0.5 para γ en algunos ejemplos mostrados en la literatura.

4.2.3 RETRO-PROPAGACIÓN LEVENBERG-MARQUARDT

El uso de este algoritmo de entrenamiento es recomendado por MATLAB®. En su documentación muestra que es uno de los algoritmos que tienen mayor eficiencia en términos del tiempo de convergencia de la red. Es por ello que fue tomado en cuenta para la etapa de simulación. El entrenamiento de la red fue fuera de línea, es decir, con una base de datos generada a partir de las simulaciones de los controladores convencionales.

El algoritmo se resume en los siguientes pasos:

1. Presentar todas las entradas a la red y calcular las salidas correspondientes de la red y los errores $e_q = \boldsymbol{t}_q - \boldsymbol{a}_q^M$. Calcular la suma de los errores cuadrados de todas las entradas, representado por *F(x)*.

$$F(\boldsymbol{x}) = \sum_{q=1}^{N} (v_i)^2 \tag{4.23}$$

Donde:

$$\boldsymbol{v}^T = [v_1\ v_2 \dots\ v_n] = e_{1,1}\ e_{2,1} \dots e_{S^M,1}\ e_{1,2} \dots\ e_{S^M,Q} \tag{4.24}$$

$$N = QxS^M$$

2. Calcular la matriz Jacobiana:

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_{1,1}^1} & \frac{\partial e_{1,1}}{\partial w_{1,2}^1} & \cdots & \frac{\partial e_{1,1}}{\partial w_{S^1,R}^1} & \frac{\partial e_{1,1}}{\partial b_1^1} & \cdots \\ \frac{\partial e_{2,1}}{\partial w_{1,1}^1} & \frac{\partial e_{2,1}}{\partial w_{1,2}^1} & \cdots & \frac{\partial e_{2,1}}{\partial w_{S^1,R}^1} & \frac{\partial e_{2,1}}{\partial b_1^1} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \cdots \\ \frac{\partial e_{S^M,1}}{\partial w_{1,1}^1} & \frac{\partial e_{S^M,1}}{\partial w_{1,2}^1} & \cdots & \frac{\partial e_{S^M,1}}{\partial w_{S^1,R}^1} & \frac{\partial e_{S^M,1}}{\partial b_1^1} & \cdots \\ \frac{\partial e_{1,2}}{\partial w_{1,1}^1} & \frac{\partial e_{1,2}}{\partial w_{1,2}^1} & \cdots & \frac{\partial e_{1,2}}{\partial w_{S^1,R}^1} & \frac{\partial e_{1,2}}{\partial b_1^1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{4.25}$$

Inicializar las relaciones de recurrencia con la ecuación (4.27) y con ellas calcular las sensibilidades:

$$\tilde{\boldsymbol{S}}_q^m = \dot{\boldsymbol{F}}^m(\boldsymbol{n}_q^m)(\boldsymbol{W}^{m+1})^T \tilde{\boldsymbol{S}}_q^{m+1} \tag{4.26}$$

Aumentar las matrices individuales hacia las sensibilidades Marquardt (4.27) utilizando la ecuación (4.28). Calcular los elementos de la matriz Jacobiana con las ecuaciones (4.29) y (4.30).

$$\tilde{\boldsymbol{S}}_q^M = -\dot{\boldsymbol{F}}^M(\boldsymbol{n}_q^M) \tag{4.27}$$

$$\tilde{\boldsymbol{S}}^m = \left[\tilde{\boldsymbol{S}}_1^m \middle| \tilde{\boldsymbol{S}}_2^m \middle| \ldots \middle| \tilde{\boldsymbol{S}}_Q^m\right] \tag{4.28}$$

$$[\boldsymbol{J}]_{h,l} = \tilde{S}_{i,h}^m \times a_{j,q}^{m-1} \quad para\ el\ peso\ x_i \tag{4.29}$$

$$[\boldsymbol{J}]_{h,l} = \tilde{S}_{i,h}^m \quad para\ el\ sesgo\ x_i \tag{4.30}$$

La Sensibilidad Marquardt está dada por la siguiente expresión:

$$S_{i,h}^{-m} \cong \frac{\partial e_{k,q}}{\partial n_{i,q}^m} \tag{4.31}$$

$$h = (q-1)S^M + k \tag{4.32}$$

3. Solucionar la siguiente ecuación para obtener $\Delta \boldsymbol{x_k}$.

$$\Delta \boldsymbol{x}_k = -[\boldsymbol{J}^T(\boldsymbol{x}_k)\ \boldsymbol{J}(\boldsymbol{x}_k) + \mu_k \boldsymbol{I}]^{-1}\, \boldsymbol{J}^T(\boldsymbol{x}_k)\, \boldsymbol{v}(\boldsymbol{x}_k) \tag{4.33}$$

Donde:

$$\boldsymbol{x}^T = [x_1\ x_2\ \ldots\ x_n] \tag{4.34}$$

$$\boldsymbol{x}^T = \left[w_{1,1}^1\ w_{1,2}^1 \ldots w_{S^1,R}^1\ b_1^1 \ldots\ b_{S^1}^1\ w_{1,1}^2 \ldots\ b_{S^M}^M\right] \tag{4.35}$$

$$n = S^1(R+1) + S^2(S^1+1) + \cdots + S^M(S^{M-1}+1) \tag{4.36}$$

4. Recalcular la suma de los errores cuadrados utilizando $\boldsymbol{x_k} + \Delta \boldsymbol{x_k}$. Si la nueva suma de cuadrados es menor que la calculada en el paso 1, entonces se divide μ entre ϑ (factor propuesto mayor que 1), dejando $\boldsymbol{x_{k+1}} = \boldsymbol{x_k} + \Delta \boldsymbol{x_k}$ y regresando al paso 1. Si la suma de cuadrados no está reducida entonces multiplicar μ por ϑ y regresar al paso 3.

4.3 SELECCIÓN DEL ALGORITMO

Con el objetivo de utilizar un solo algoritmo para las diferentes pruebas en la etapa de implementación se propuso una función simple para simular y observar los tiempos de respuesta de cada algoritmo.

La función a simular es la siguiente:

$$y = seno(x) \qquad \text{Ecu. (4.36)}$$

Dentro del rango $[0, \frac{5}{2}\pi]$.

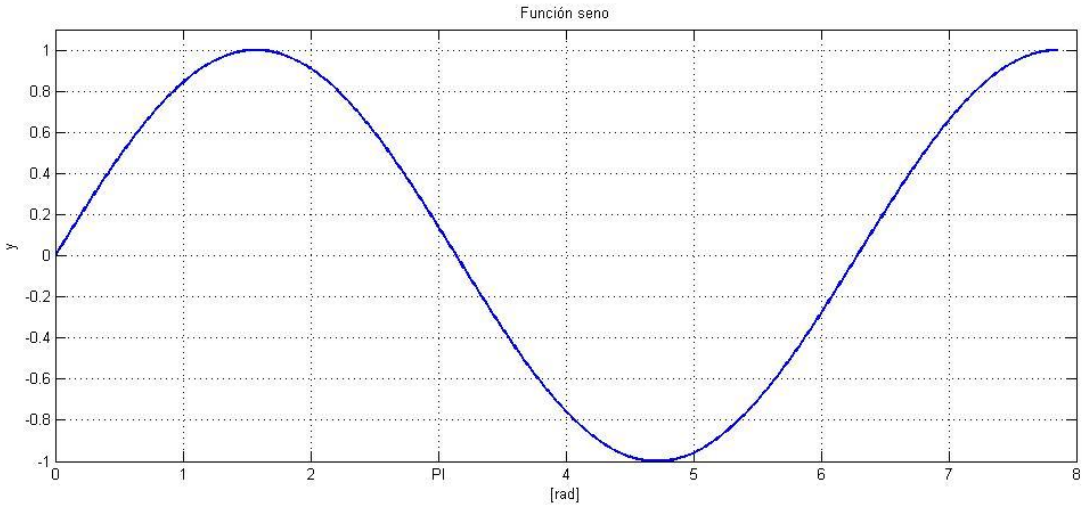


Fig. 4-4 Gráfica de la función seno.

Tomando en cuenta las recomendaciones de algunos autores, como Hagan, que indica que a mayor número de puntos de inflexión en la función tantas neuronas debe tener su capa oculta, se puede proponer una red de 3 neuronas, considerando que al inicio de la gráfica hay otro punto de inflexión. Sin embargo, se propone una red 1-5-1 (una entrada, cinco neuronas en la capa oculta, una salida) para tener una arquitectura sobrada y que permita comparar la convergencia de los algoritmos. El anexo A.8: Efecto del número de neuronas en una red de una sola capa oculta muestra la respuesta de redes con diferente número de neuronas ocultas.

Se realizaron tres entrenamientos con cada algoritmo de aprendizaje presentado con la finalidad de encontrar el algoritmo más eficiente, en términos de tiempo y convergencia. El número de épocas o iteraciones del algoritmo está limitado a 50,000.

Para este proceso se utilizó el RNA Toolbox de MATLAB®, el cual ya tiene los métodos de cada algoritmo y permite visualizar el estatus del entrenamiento. Es importante señalar que el criterio de validación de la red es el valor mínimo del

error mostrado durante la etapa de entrenamiento, esto no indica que se detenga cuando se presenta el error mínimo, esto se limita al número de iteraciones permitidas.

La siguiente tabla muestra los tiempos de convergencia, el número de épocas y el error remanente.

Tabla 4-1 Tabla comparativa de los algoritmos simulados.

Regla de aprendizaje	**Regla Delta generalizada**			**BP con momentum**			**Levenberg-Marquardt**		
Prueba	1	2	3	1	2	3	1	2	3
Lr (alfa)	0.01	0.05	0.1	0.01	0.05	0.05	--	--	--
Mc	--	--	--	0.9	0.8	0.9	--	--	--
Mu inicial	--	--	--	--	--	--	0.001	0.01	0.05
ϑ (incremento)	--	--	--	--	--	--	0.1	0.1	0.1
ϑ (decremento)	--	--	--	--	--	--	10	10	10
Iteraciones	50,000	50,000	50,000	50,000	50,000	50,000	37	61	89
Tiempo [min:seg]	16:47	18:44	17:26	16:38	16:40	16:34	0:01	0:01	0:02
Error cuadrático medio final	1.2 x10^{-3}	3.4761 x10^{-4}	**3.0027 x10^{-4}**	9.9531 x10^{-4}	3.2054 x10^{-4}	**3.1498 x10^{-4}**	4.3189 x10^{-5}	3.7418 x10^{-7}	**2.2347 x10^{-7}**

Como se puede apreciar en la tabla anterior, el algoritmo de retro-propagación Levenberg-Marquardt converge mucho más rápido que los otros algoritmos comparados, además, el número de iteraciones es también menor. Los algoritmos de la Regla Delta generalizada y el de retro-propagación con momentum ocuparon el máximo número de iteraciones establecido y el error remanente es mayor.

En esta etapa de pruebas de los algoritmos se realizó cada entrenamiento tres veces y con alguna modificación de los parámetros iniciales como la razón de aprendizaje, el coeficiente de momentum y μ, todo esto con la finalidad de encontrar la mejor respuesta de cada algoritmo.

En el caso del algoritmo de la regla delta generalizada los valores propuestos para la razón de aprendizaje fueron 0.01, 0.05 y 0.1, ya que son valores pequeños y así se pueden evitar oscilaciones grandes entre cada actualización de parámetros. El entrenamiento realizado con $\alpha = 0.1$ presenta el menor error cuadrático medio comparado con los dos primeros entrenamientos. La siguiente gráfica muestra el acercamiento de la red neuronal hacia la función seno en su última iteración.

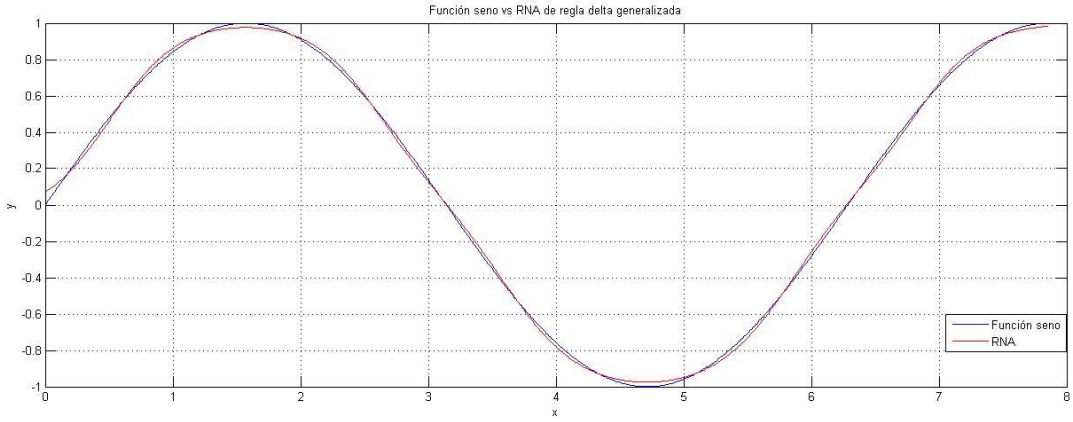


Fig. 4-5 Respuesta de la RNA entrenada con el algoritmo de retro-propagación.

Para el algoritmo de retro-propagación con momentum se propusieron los valores de 0.01 y 0.05 para la razón de aprendizaje y 0.8 y 0.9 para el coeficiente de *momentum*, se puede observar que con el juego de los valores de γ con α=0.05 se obtienen los valores más pequeños del error. Sin embargo, estos dos algoritmos se ven limitados por el número de iteraciones, ya que si hubiese sido mayor éste seguramente se llegaría a un valor menor del error, pero requiere de más tiempo de procesamiento. Como se puede apreciar en la siguiente gráfica, la respuesta de la RNA se parece mucho a la respuesta de la RNA del algoritmo anterior

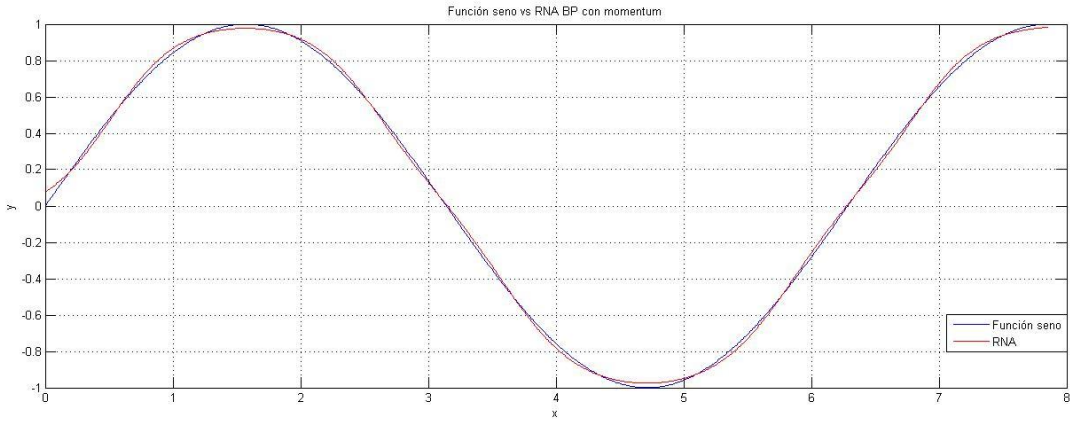


Fig. 4-6 Respuesta de la RNA entrenada con el algoritmo de retro-propagación con momentum.

En la siguiente figura se puede apreciar la respuesta final de la RNA entrenada con el algoritmo Levenberg–Marquardt, donde es notorio un mejor aprendizaje por parte de la red neuronal, puesto que el error es mínimo y no necesitó de tantas iteraciones para encontrar el valor mínimo.

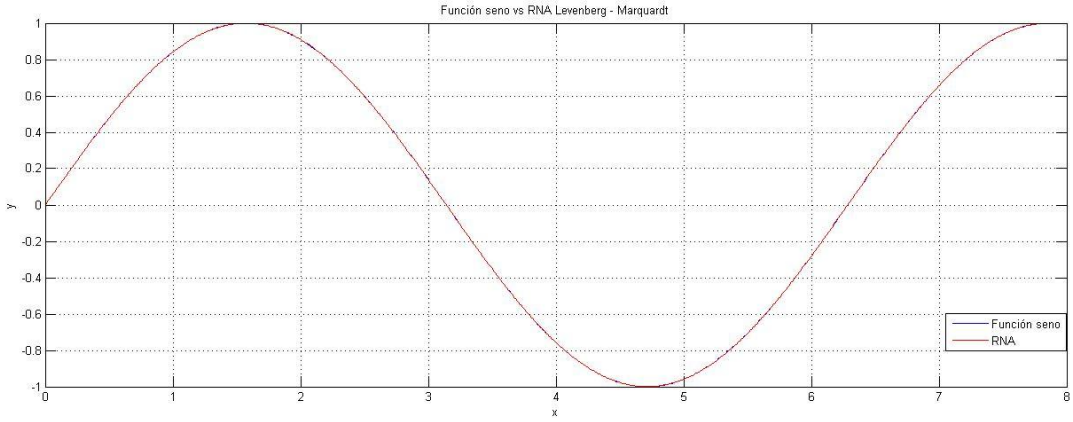


Fig. 4-7 Respuesta de la RNA entrenada con el algoritmo de retro-propagación Levenberg-Marquardt.

Es importante señalar que en este trabajo se implementará un controlador basado en redes neuronales artificiales y estas redes se van a entrenar fuera de línea, por lo que tiene prioridad encontrar los parámetros de la red que permitan tal implementación de la manera más precisa y rápida. Por estas razones se seleccionó el algoritmo Levenberg-Marquardt.

5. IMPLEMENTACIÓN DEL CONTROLADOR

En este capítulo se presentan los procesos que se deben llevar a cabo para la implementación de los controladores evaluados. Ya se habló acerca del motor que se va a utilizar, lo consecuente es hablar de los detalles que se requieren para que se pueda tomar lectura de la posición, así como de las secciones encargadas de la implementación.

En el siguiente diagrama se pueden apreciar los elementos que se necesitan para implementar el control de posición del motor de corriente directa.

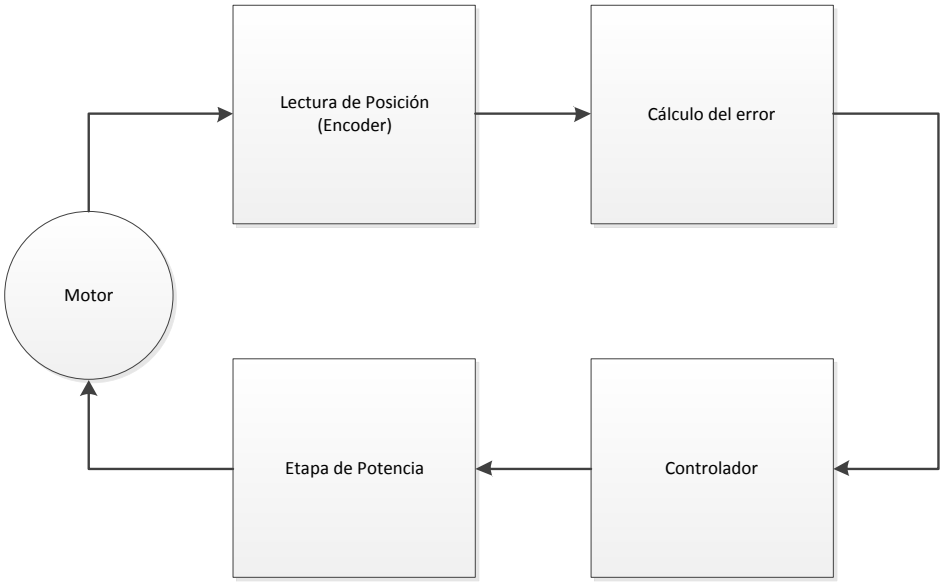


Fig. 5-1 Diagrama de implementación.

Como paso siguiente, se analizaron las opciones para implementar cada subsistema, excepto el motor, el cual ya fue seleccionado desde una etapa inicial.

Lectura de la posición del motor

En el capítulo 2, se describieron los parámetros del motor otorgados por el fabricante, este motor cuenta con un *encoder* diferencial, con una resolución de 1024 pulsos por revolución, cuyos pines de conexión son los siguientes:

Tabla 5-1 Descripción de los pines del *encoder* del motor Pittman 14204.

Pin	Color	Función
1	Negro	Tierra
2	Verde	Índice
3	Amarillo	Canal A
4	Rojo	Vcc
5	Azul	Canal B

Estos pines irán conectados con el dispositivo de implementación, es importante tomar esto en cuenta, ya que se desea llevar los procesos a cabo en un FPGA, sobre todo en el caso de la alimentación, ya que los FPGA pueden tener diferentes voltajes de operación.

Cálculo del error y acción de control

El cálculo del error y la acción de control se llevarán a cabo con el dispositivo seleccionado para implementar la RNA, que se pretende que sea una tarjeta de desarrollo basado en FPGA.

Etapa de potencia

La etapa de potencia va a depender de la selección de la tarjeta de desarrollo, si alguna tiene alguna tarjeta adicional de potencia nos puede ser útil, de otra manera habrá que pensar en algún arreglo que permita darle al motor la potencia requerida para que se pueda controlar, como un puente H.

5.1 SELECCIÓN DEL DISPOSITIVO DE IMPLEMENTACIÓN

Para la implementación de la RNA se evaluaron distintas tarjetas de desarrollo basadas en FPGA. En la siguiente tabla se muestran algunas plataformas que fueron contempladas.

Tabla 5-2 Ejemplos de distintas plataformas FPGA y sus características.

Plataforma	Basys FPGA Board 100K	Nexys2 FPGA Board	Spartan 3E Starter Board 500K	NI CompactRIO NI Crio 9073 NI 9505
Imagen				
Chip	Spartan 3E-100 CP132	Spartan 3E-500 FG320	XC3S500E	Spartan-3
Marca	Xilinx®	Xilinx®	Xilinx®	National Instruments®
Celdas	100K	500K	500K	2000K
Tipo de memoria	XCF02 Platform Flash ROM	16MB Intel StrataFlash Flash R 16MB fast Micron PSDRAM Xilinx Platform Flash ROM	32MB Micron DDR SDRAM 16MB Numonyx StrataFlash 2MB ST Microelectronics Serial Flash	64MB DRAM 128MB de memoria no volátil para registro de datos
Oscilador	25/ 50/ 100 MHz	50 MHz	50MHz	266 MHz
Puertos	Puerto PS/2 Puerto VGA de 8 pines USB 2.0 de velocidad completa	Puerto VGA de 8 pines Puerto de 2 pines RS232 Serial y USB	PS/2 para teclado Dos conectores DB9 RS-232 Ethernet RJ-45 USB 2.0	Puerto Ethernet de 10/100 Mb/s
I/O	32	32	43	Dependiendo de los slots.
Voltajes de operación	3.5-5.5V	5-15V	3.3-5V	<30V
Otras Características	Multiplicadores de 18 bits 8 LEDs 8 Interruptores 4 Botones 4 Displays de 7 segmentos	8 LEDs 8 Interruptores 4 Botones 4 Displays de 7 segmentos	4 Interruptores 8 LEDs DAC Push button switch giratorio Pantalla LCD	Interfaz gráfica Conexión con servidores HTTP y archivos FTP
Software de Trabajo	Xilinx ISE Webpack	Xilinx ISE Webpack	Xilinx ISE Webpack	LabVIEW®
Precio aproximado	$1,550 MXN	$2,440 MXN	$3,420 MXN	$25,000 MXN

La plataforma seleccionada fue NI CompactRIO®. Es un sistema embebido y reconfigurable de control y adquisición. La arquitectura del hardware incluye espacios para módulos de E/S, un chasis FPGA reconfigurable y un controlador embebido de tiempo real. Esta plataforma presenta una velocidad superior de procesamiento a las demás comparadas debido a su oscilador de 800 MHz, cuenta con conexión Ethernet, memoria no volátil y es programable vía LabVIEW®, lo cual permite desarrollar el proyecto con una interfaz gráfica. Por otra parte, se cuenta con este sistema en el laboratorio de pruebas y no se requiere de la creación de una etapa de potencia, ya que se tiene un módulo para esto.

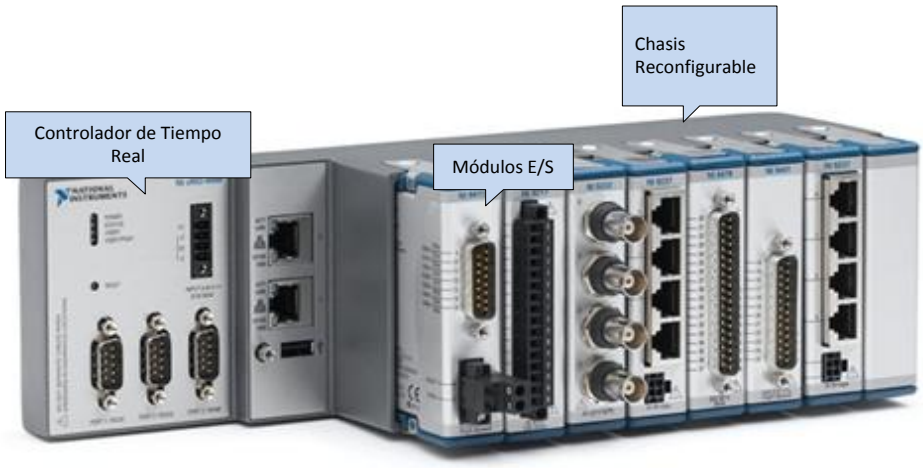


Fig. 5-2 Módulo CompactRIO (Fuente: NI).

La plataforma CompactRIO® permite trabajar de forma conjunta con una computadora de escritorio o ser utilizada de manera independiente. Debido a que cuenta con una memoria no volátil queda guardado el último proyecto cargado sobre el FPGA. La forma de trabajo depende de la aplicación.

Los módulos de entrada/salida de CompactRIO® son diversos, ejemplos de ello son módulos para sensores de temperatura, humedad; controladores de motores con etapa de potencia; módulos de entradas y salidas digitales; módulos de entrada analógica, entre otros. La ventaja de utilizar estos módulos es que tienen la arquitectura para mandar acondicionadas las señales de manera interna, esto se puede apreciar en la Fig. 5-3. Estas señales son leídas y/o mandadas por el FPGA del CompactRIO®, a su vez todo proceso se puede comunicar con el microprocesador de tiempo real. Este sistema es ideal para el monitoreo y control de variables físicas.

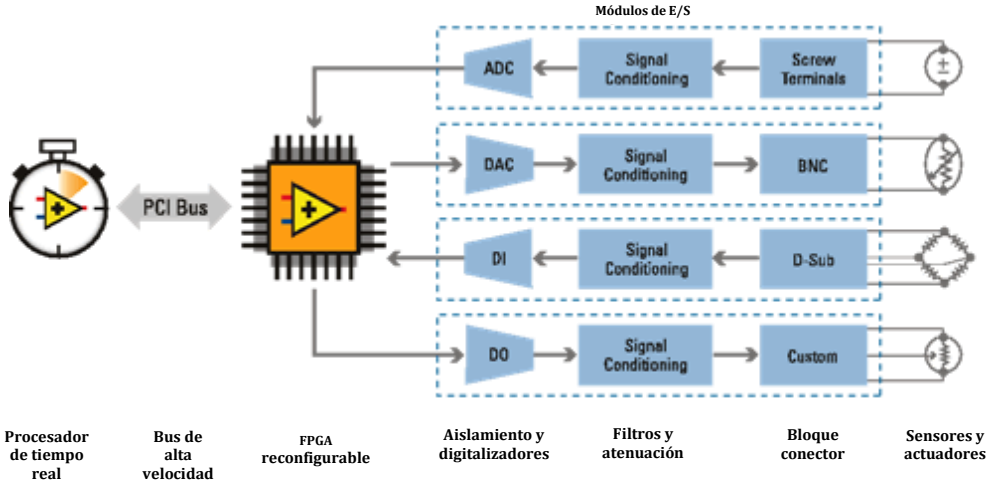


Fig. 5-3 Elementos de la plataforma CompactRIO (Fuente: NI).

Módulo NI 9505

El módulo 9505 de National Instruments para CompactRIO® es un controlador de motores de corriente directa con escobilla. Tiene entradas para leer *encoders* diferenciales y pines para alimentar de manera independiente al motor. Este módulo es ideal como complemento para la etapa de potencia necesaria para mover el motor especificado en este trabajo. En las figuras mostradas a continuación se puede apreciar la imagen del módulo así como su diagrama de conexión.

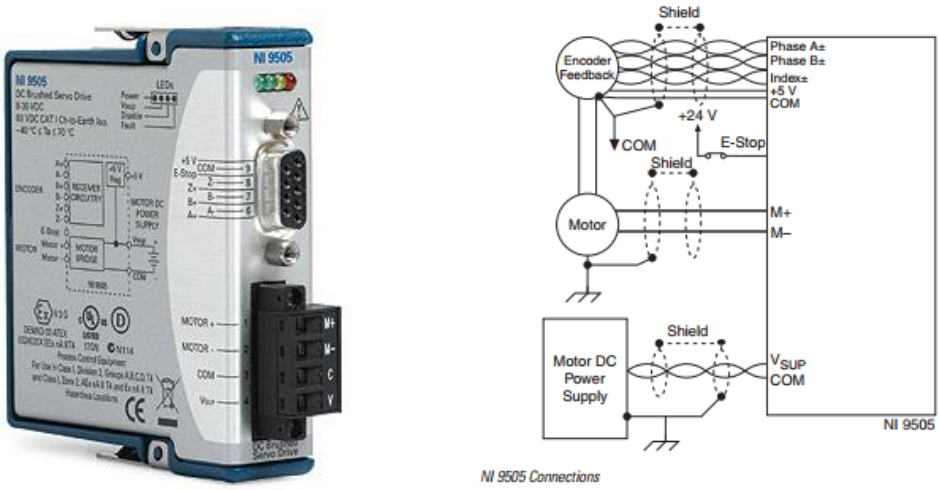


Fig. 5-4 Módulo NI 9505 y su diagrama de conexiones (Fuente: NI).

5.2 ELEMENTOS Y PASOS DE LA IMPLEMENTACIÓN

Tanto para el controlador proporcional como para el control proporcional – derivativo se efectuaron, primeramente, sus simulaciones comparadas con las simulaciones de la red neuronal artificial correspondiente, con el fin de corroborar el buen funcionamiento del programa a implementar.

Los pasos realizados para la implementación fueron los siguientes:

1. Entrenamiento de la RNA.
2. Verificación de la RNA en MATLAB®.
3. Simulación del controlador neuronal en Simulink®.
4. Verificación de la RNA en LabVIEW®.
5. Implementación de la RNA y del controlador P y/o PD en el CompactRIO®.
6. Almacenamiento de datos de la implementación para la generación de gráficas.

La siguiente figura muestra la manera en que se desarrolló la implementación, se utilizó primeramente MATLAB® para entrenar y simular la RNA, después se utilizó CompactRIO® para implementar los controladores y generar los datos para las gráficas, las cuales fueron hechas en MATLAB®.

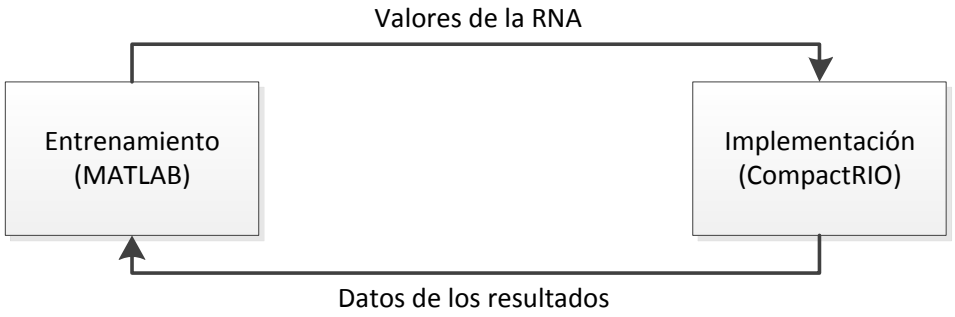


Fig. 5-5 Flujo para la implementación y verificación de las RNA.

Para poder simular las redes neuronales artificiales se generaron los datos de entrenamiento por medio de MATLAB®. Para ello se especificaron los rangos de trabajo, en el caso del controlador proporcional corresponden el valor máximo y el valor mínimo de error, los comandos correspondientes para esto se detallan en la sección de implementación de cada controlador. Posteriormente, los parámetros de la RNA son almacenados y son verificados tanto en MATLAB® como en LabVIEW®.

En el caso de la implementación en CompactRIO® se realizó un proyecto FPGA, con los siguientes elementos principales:

1. My Computer: Permite agregar VI's ejecutados por la computadora, así como su sincronización con el proyecto.

2. Módulo cRIO: En el proyecto aparece el nombre del módulo, así como su dirección IP, se pueden sincronizar datos con la computadora o trabajar de manera independiente. Contiene los siguientes elementos:

    a. Biblioteca de Variables: Esta sección sirve para declarar, agregar y almacenar valores para variables globales, las cuales se pueden compartir con otros VI's del proyecto.

b. Chasis: En el chasis se encuentra el dispositivo FPGA y el registro de módulos conectados. Se puede acceder a los elementos del chasis como el LED del FPGA, la temperatura del chasis, entre otros. También es posible acceder a los elementos de los módulos. En el caso de esta aplicación se puede acceder a las acciones que el módulo de potencia nos permite como el caso de la medida de la intensidad de corriente que circula por el motor, la lectura del encoder o la dirección de giro del rotor del motor. Para acceder a los elementos descritos se necesita crear un VI en el Chasis, el cual se ejecuta en el FPGA.

c. VI's del cRIO: Estos VI's se ejecutan en el procesador de tiempo real que contiene cRIO. Es posible acceder a la información de los módulos accediendo al VI ejecutado por el FPGA.

La siguiente figura muestra un panorama de la arquitectura del proyecto de este trabajo.

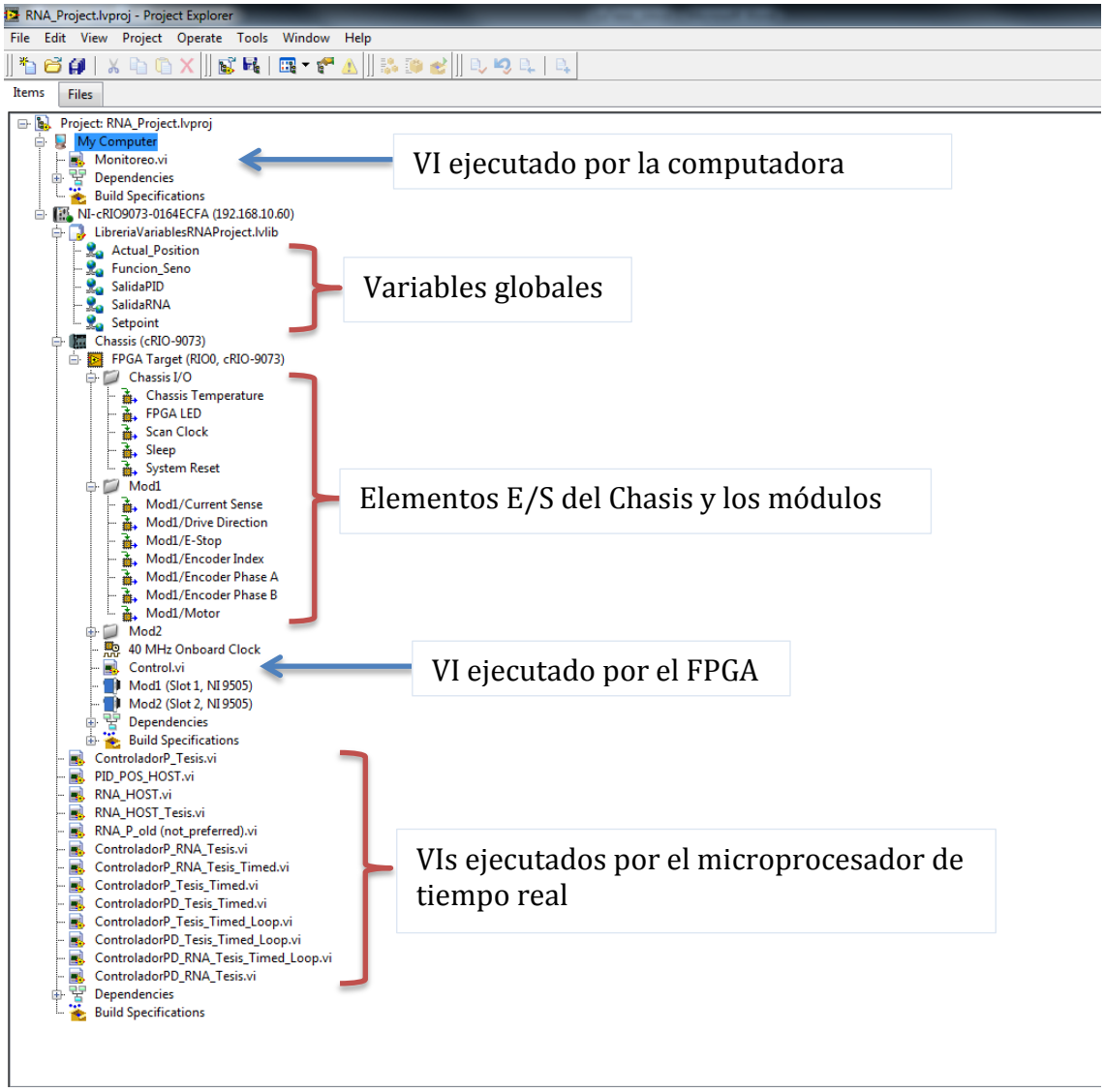


Fig. 5-6 Estructura del proyecto de cRIO.

5.3 IMPLEMENTACIÓN DEL CONTROLADOR PROPORCIONAL

En esta sección se describirá a detalle cada paso realizado para poder llevar a cabo la implementación del controlador proporcional.

5.3.1 ENTRENAMIENTO DE LA RNA PARA EL CONTROLADOR PROPORCIONAL

En el capítulo 4 se mostró que una RNA puede aprenderse una función seno, mientras que en este capítulo se tiene que aprender otra función, que en este caso es la del controlador proporcional. Se tiene que la salida del controlador es directamente proporcional a la entrada de éste, es decir, existe un factor o constante *kp* que afecta la salida de manera lineal.

El primer paso para diseñar la RNA fue identificar el tipo de función que se va a aprender, que en este caso es una línea recta con pendiente *kp = 1.835821*, recordando que el valor de la constante se especificó en el capítulo 3. Se generaron los datos en MATLAB® para entrenar la RNA a partir de la respuesta que puede emitir un controlador proporcional para el rango de valores de entrada (errores de posición del rotor) de [-2π, 2π], restringido para un error máximo en la posición de una revolución y el rango de [-11.5348, 11.5348] para las salidas, las cuales son los voltajes que debe generar el controlador (*kp*error*). Los comandos utilizados se muestran a continuación:

```
%Valores de entrenamiento
kp = 1.835821;              % Coeficiente proporcional
p = [-2*pi:pi/4:2*pi];      % Valores de entrada (Error de posición)
t = kp*p;                   % Valores de salida del controlador
```

Una vez obtenidos estos valores se normalizaron por medio de la función *Map* (véase anexo F). La siguiente figura muestra los valores de entrenamiento y sus valores normalizados para entrenar la RNA.

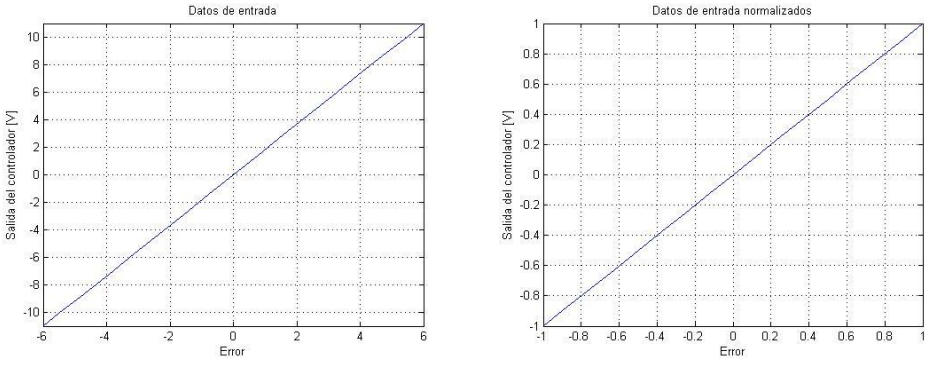


Fig. 5-7 Gráficas con los valores de entrenamiento (lado izquierdo) y datos normalizados (lado derecho) para una $k_p = 1.835821$.

Una vez normalizados los datos se pensó en la estructura adecuada de la RNA, este problema se puede resolver fácilmente sin la necesidad de una red compleja o sin necesidad de una red neuronal, basta una simple neurona con una función lineal de activación para implementar este controlador; sin embargo, el

objetivo de este trabajo es mostrar la capacidad de una RNA de retro-propagación, por ello se consideró una red de una capa oculta y dos neuronas en esta capa (red 1-2-1).

Para facilitar el entrenamiento de esta red y del trabajo en general, se utilizó el *Neural Network Toolbox* de MATLAB®, los comandos para tal entrenamiento son los siguientes:

```
22 -    net = feedforwardnet([2]);              % Red 1 - 2 - 1
23 -    net.trainParam.epochs = 50000;          % Número máximo de épocas
24 -    [net tr] = train(net, pn, tn);          % Entrenamiento de la red
```

Fig. Comandos de entrenamiento para una red de propagación hacia adelante.

El comando *feedforwardnet* define que es una red de retro–propagación, el número "[2]" que se encuentra entre corchetes define que es una red con dos neuronas en su capa oculta. El comando *net.trainParam.epochs* permite definir el número de épocas permitidas, que en este caso es de 50,000 y el comando *train* comienza el entrenamiento; "pn" y "tn" son los valores de entrenamiento normalizados.

5.3.2 VERIFICACIÓN DE LA RNA

Después de realizar el entrenamiento se obtuvieron los siguientes parámetros de la red:

```
W1 = [0.1421; 0.1497];          %Vector de pesos de la capa oculta
b1 = [-0.6515; 0.6702];         %Vector de sesgos de la capa oculta
W2 = [5.4261 4.8939];           %Vector de pesos de la capa de salida
b2 = 0.2440;                    %Vector de sesgos de la capa de salida
```

Con estos valores se probó la RNA, la siguiente gráfica muestra los resultados de la RNA entrenada comparados con los valores de entrenamiento.

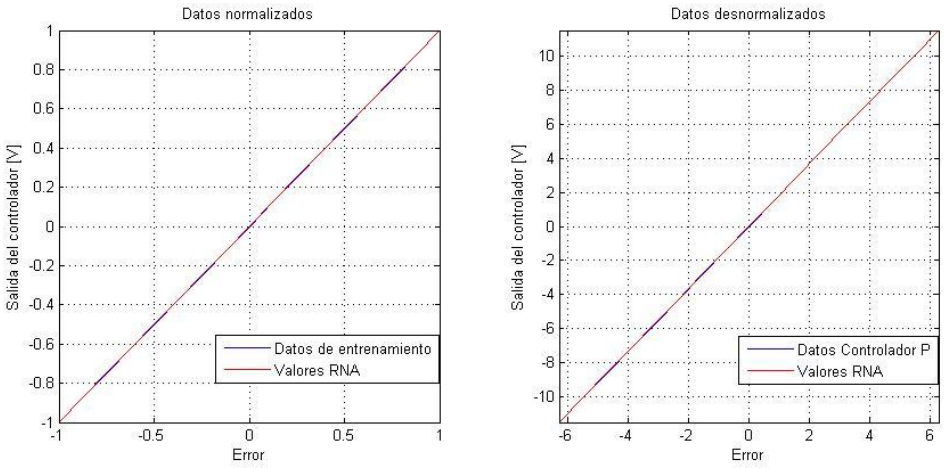


Fig. 5-8 Verificación de la RNA.

5.3.3 SIMULACIÓN DE LA RED NEURONAL

Se utilizó Simulink® para poder comparar la respuesta del motor en lazo cerrado, véase diagrama de bloques en la siguiente figura. El objetivo de la simulación y de la implementación es poder hacer un interruptor que nos permita alternar entre un controlador y otro, con esto se pueden almacenar los datos de la prueba y comparar al final.

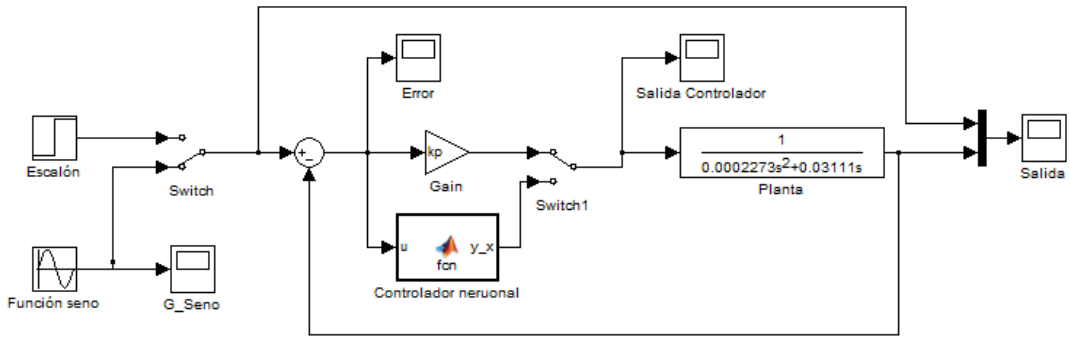


Fig. 5-9 Diagrama de bloques de Simulink®.

Se puede apreciar un bloque representando la RNA en el diagrama de bloques, el cual contiene el código de verificación de la RNA, donde la entrada de la red es el error generado por la diferencia de la posición real menos la deseada y la salida es equivalente a la salida del controlador proporcional; el programa del bloque de la RNA se encuentra en los anexos. El paso siguiente fue comparar los resultados de las simulaciones, tanto para el controlador proporcional, como para el controlador neuronal.

Los parámetros de la simulación realizada son los siguientes:

Tabla 5-3 Parámetros de simulación para el controlador proporcional

Parámetro	**Valor**	**Unidades**
Tiempo de simulación	1	[s]
Valor inicial función escalón	0	[rad]
Valor final de la función escalón	$\frac{\pi}{4}$	[rad]
Amplitud de la función seno	$\frac{\pi}{4}$	[rad]
Frecuencia	10	[Hz]

Tanto en la simulación de la función escalón como en la función de seguimiento sinusoidal, se obtuvo una muy buena aproximación, como se puede apreciar en el siguiente par de gráficas.

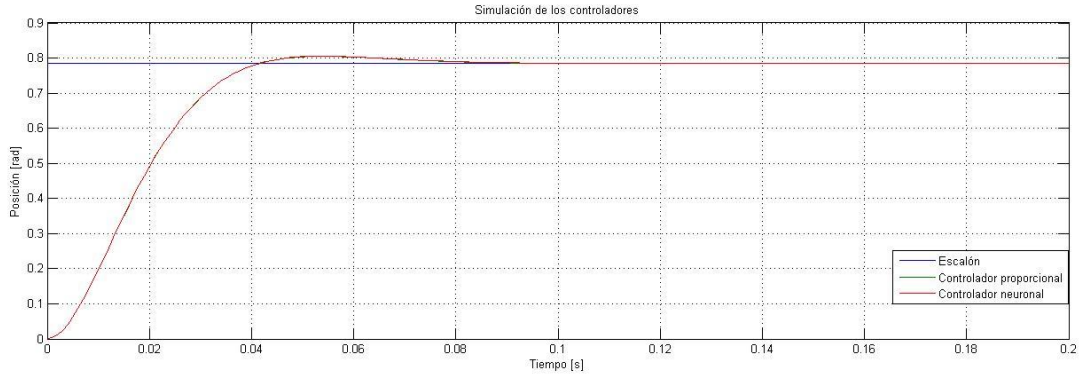


Fig. 5-10 Comparación por simulación de los controladores ante una función escalón.

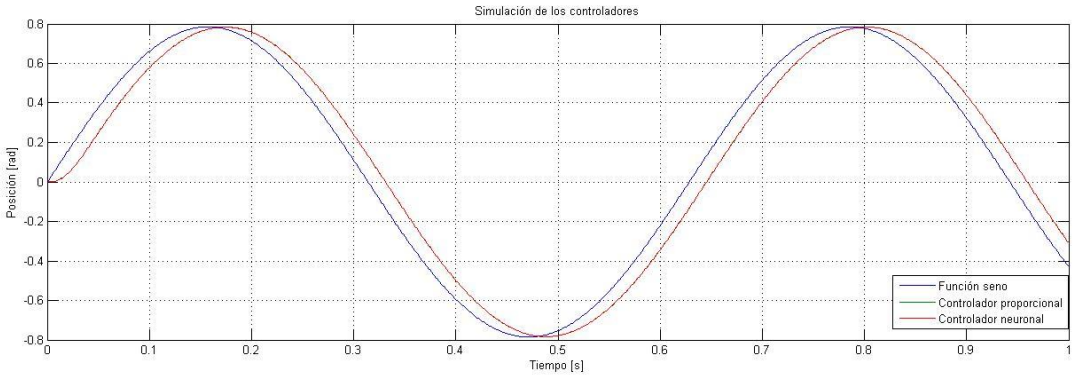


Fig. 5-11 Comparación por simulación de los controladores ante una función seno.

5.3.4 VERIFICACIÓN DE LA RNA EN LabVIEW®

Debido a la selección del dispositivo se tuvo que trabajar con la plataforma CompactRIO®, por lo que el software utilizado fue LabVIEW®. Esta etapa fue la que costó más trabajo, debido a que esta paquetería cuenta con diversos elementos que pueden facilitar la programación, sin embargo, es importante tener en cuenta la capacidad computacional que demanda cada elemento.

Una parte fundamental de este trabajo fue verificar el funcionamiento de la RNA, por ello se realizó un programa independiente de la implementación para revisar que la implementación pueda llevarse a cabo satisfactoriamente.

La siguiente secuencia de imágenes muestra la estructura del programa, así como las actividades o eventos a realizar.

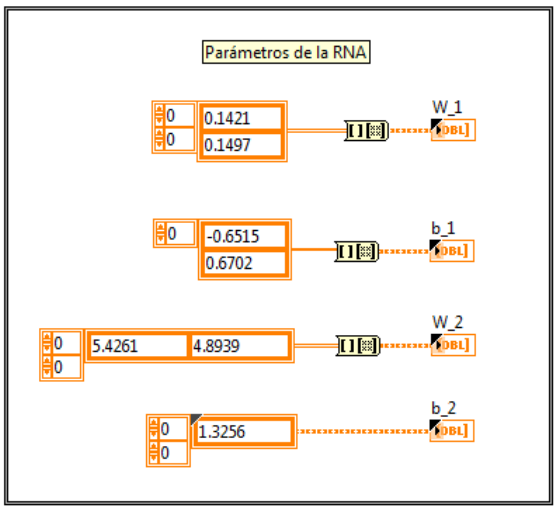


Fig. 5-12 Inicialización de parámetros de la RNA.

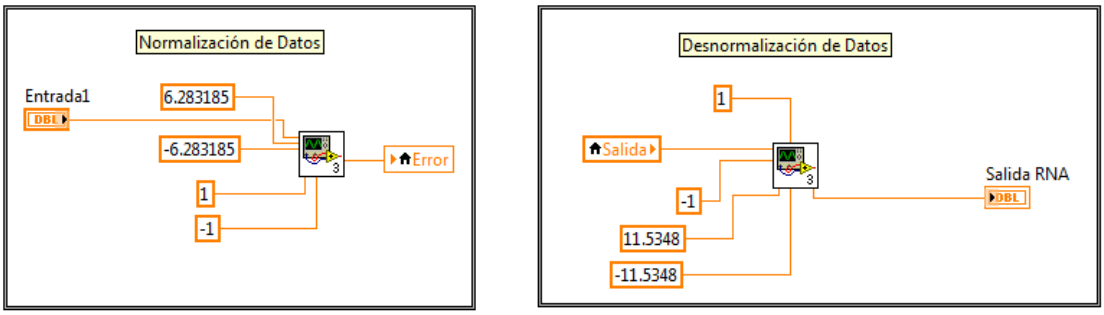


Fig. 5-13 Acondicionamiento de los datos de entrada y salida de la RNA.

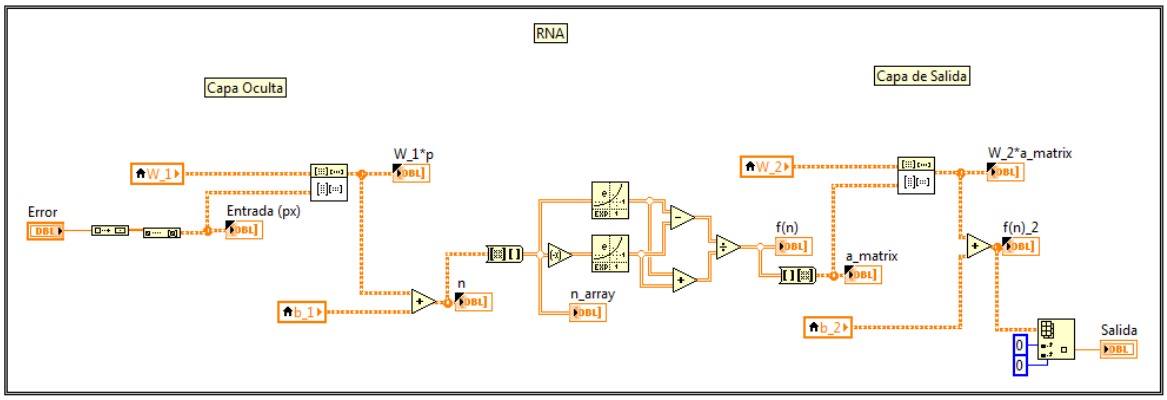


Fig. 5-14 Red neuronal en LabVIEW®.

Los resultados fueron favorables, existieron algunos errores en los valores finales, esto es debido a que el VI que ejecuta la función exponencial difiere de los valores que da una función exponencial de MATLAB®. Sin embargo, a pesar de estos errores se puede verificar el funcionamiento de la RNA.

5.3.5 IMPLEMENTACIÓN DE LA RNA EN CompactRIO

En esta etapa se replicó el programa de LabVIEW® utilizado para verificar la RNA en un proyecto para FPGA. La siguiente tabla muestra las acciones ejecutadas por cada subsistema del proyecto FPGA.

Tabla 5-4 Elementos del proyecto FPGA.

Subsistema	Acciones
FPGA	1. Lectura del *encoder* para determinar la posición del eje del motor. 2. Ejecución del PWM. 3. Paro de emergencia. 4. Almacenamiento de datos en variables internas del FPGA.
Módulo de Tiempo Real	1. Implementación del controlador proporcional. 2. Acondicionamiento de datos. 3. Implementación del controlador neuronal.
Computadora	1. Almacenamiento de datos. 2. Inicialización del s*etpoint* del controlador. 3. Ejecutar función de seguimiento. 4. Generación de base de datos para graficar.

5.3.6 ALMACENAMIENTO DE DATOS

En el caso del almacenamiento de los datos se generó una base de datos en un archivo de texto con extensión ".txt" para poder graficarlos en MATLAB®, ya que nos permite manejar las gráficas de una manera más sencilla, e incluso poder utilizar estos valores para simulaciones futuras.

La figura Fig. 5-15 muestra las variables que fueron almacenadas como el tiempo en segundos, el *setpoint*, la posición real, el error de la posición y la salida de la RNA.

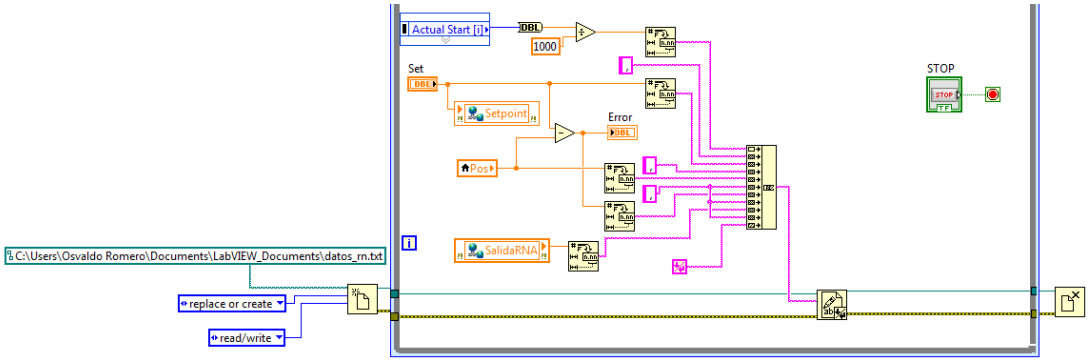


Fig. 5-15 Almacenamiento de Datos en LabVIEW®[3].

[3] En los anexos se puede encontrar el código completo de LabVIEW® para cada proceso.

5.4 IMPLEMENTACIÓN DEL CONTROLADOR PROPORCIONAL-DERIVATIVO

Durante el desarrollo de esta parte del trabajo se fueron presentando ciertas complicaciones que dificultaban la implementación del controlador neuronal, algunas veces por el tipo de entradas que se querían utilizar y otras por el tiempo de entrenamiento, así como el tiempo de procesamiento de la implementación. Uno de los problemas más importantes fue el tiempo, a pesar de utilizar bloques simples en el software de implementación, se presentaban variaciones en el tiempo de respuesta del controlador.

En las siguientes subsecciones se detallan las acciones realizadas para poder desarrollar el controlador proporcional-derivativo.

5.4.1 ENTRENAMIENTO DE LA RNA

El primer paso para realizar esta implementación fue definir la topología de la RNA, el criterio principal fue diseñar una red capaz de realizar las operaciones que se tienen que llevar para ejecutar un controlador proporcional - derivativo, es decir, que permita realizar la siguiente ecuación:

$$U_c = K_p(E(k)) + K_d(\frac{\Delta \mathrm{E}}{\Delta t}) \qquad \text{Ecu. (6.1)}$$

Que se puede reescribir como:

$$V_c = K_p E(k) + K_d \frac{E(k) - E(k-1)}{t(k) - t(k-1)} \qquad \text{Ecu. (6.2)}$$

Donde $K_p = 2.510061$ y $K_d = 0.005266$, que corresponden al diseño del controlador proporcional cuyo porcentaje de sobrepaso es igual a 2.5%.

Se evaluaron primeramente las dos topologías siguientes:

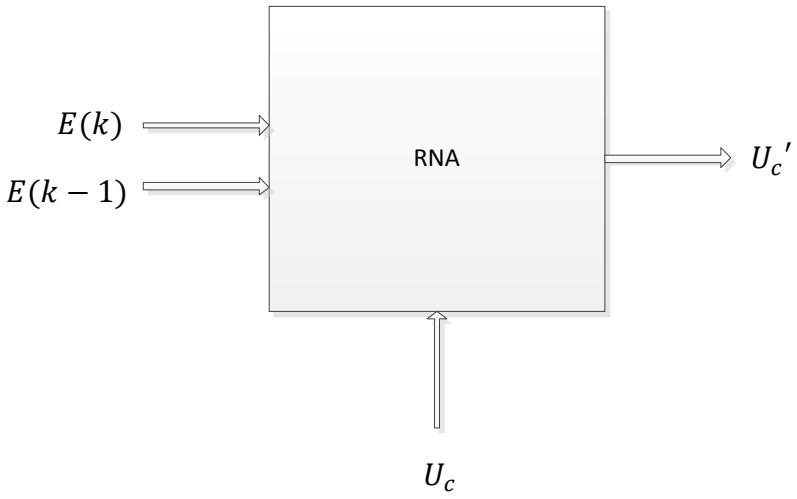


Fig. 5-16 RNA con dos entradas, una salida.

La figura anterior muestra una red neuronal con un par de entrenamiento que consta de dos entradas $p_1 = E(k)$ y $p_2 = E(k-1)$, relacionadas con un vector t = U_c. La intención de esta configuración fue aprender la relación de un controlador proporcional derivativo contra la diferencia del error generado. Es importante destacar que la diferencial del tiempo se consideró como una constante de tiempo, para determinar esta constante se verificó la duración máxima de la ejecución del programa de la red neuronal en LabVIEW® y se compensó el tiempo de espera para que durara 30 milisegundos, es decir, $\Delta t = 0.03\,[s]$. Estos 30 milisegundos es lo que tardaba usualmente en correr el programa utilizado.

En la simulación de esta red se obtuvieron resultados favorables, el aprendizaje de estos valores fue bueno y la diferencia de los errores la ejecutó sin problema alguno. Sin embargo, al momento de implementar esta red no se obtuvieron los resultados esperados, ya que el delta de tiempo especificado no era del todo preciso y su valor era muy grande, por lo que la actualización de los valores del error llegaba muy tarde y la respuesta del sistema presentaba oscilaciones. Se optó revisar otra topología, donde la RNA calculara la derivada total internamente. La siguiente figura muestra la configuración para su cálculo.

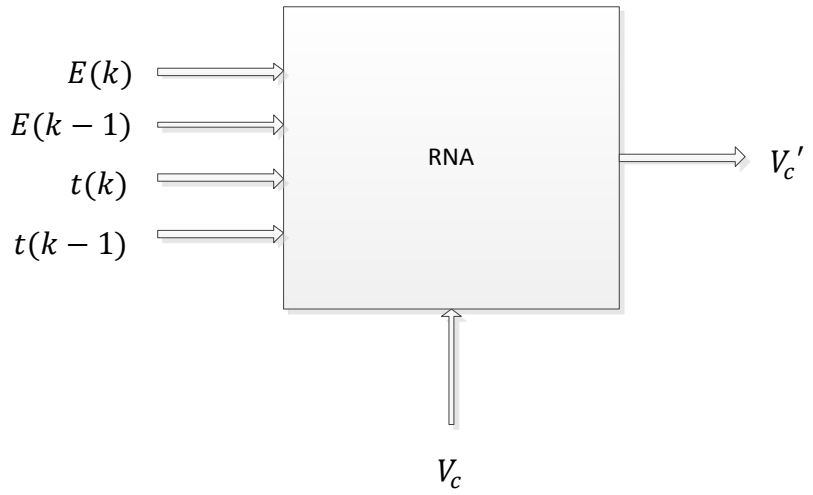


Fig. 5-17 RNA con cuatro entradas, una salida.

Esta configuración mostró buenos resultados para los valores del entrenamiento dados, sin embargo, faltaron datos para entrenarla adecuadamente y el incremento de las variables de entrada provocó un aumento exponencial en el tiempo de entrenamiento.

A pesar del buen entrenamiento que se tuvo, existieron los siguientes inconvenientes:

- Datos insuficientes para el par de entrenamiento: En el caso del controlador proporcional se entrenó esa red con un par de entrenamiento de 17 datos cada vector. Para dos entradas se incrementó a 289 datos de entrenamiento, para las cuatro entradas de esta arquitectura se utilizaron 83,521 datos. Sin embargo, fueron muy pocos, puesto que faltaban valores intermedios, además, al normalizar los datos se perdía información.

- Valores de salida inesperados: A pesar de encontrar una buena relación diferencial en la salida de la red para los valores de entrenamiento se encontraron valores inapropiados para la salida de la red, como valores con signo contrario o fuera de rango.

- Aumento de las neuronas de la capa oculta: Con la finalidad de lograr una mejor convergencia en los resultados de la red, se diseñaron un par de redes, una con 10 neuronas y otra con 15. Esta última red mostró un mejor desempeño, pero la cantidad de datos seguía siendo insuficiente.

Debido a estos inconvenientes se optó por cambiar a otra topología que nos permitiera implementar con mayor facilidad la red neuronal. La arquitectura final propuesta fue la siguiente:

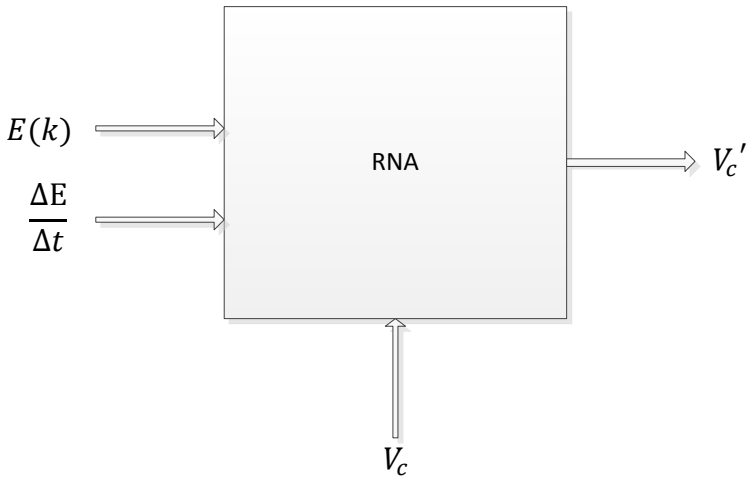


Fig. 5-18 Topología final de la RNA.

Esta topología permitió llevar a cabo la RNA de una manera más sencilla, ya que antes de la implementación se ejecutan los cálculos para determinar el error de posición del motor y la razón de cambio de este error con respecto al tiempo. Esta relación existente entre el error y su derivada puede aprenderla la RNA rápidamente y no requiere de tantos datos para ello.

Se utilizaron 20,301 datos por cada valor, generados por la combinación de 101 valores del error contra 201 valores de su derivada. Estos datos fueron suficientes para entrenar la RNA, que en este caso fue una red 2-3-1 (tres neuronas en su capa oculta); otra ventaja de utilizar esta estructura fue que en *Simulink*® se puede observar de manera sencilla el funcionamiento de la red, mientras que con las topologías anteriores es más complicado realizar la simulación.

La generación de los datos de entrenamiento fue generada por MATLAB® por medio de las siguientes líneas de programación:

```
A = [-2*pi: 2*pi/50: 2*pi];
B = [-100: 1: 100];
```

Donde A y B son los vectores del error y la derivada del error respectivamente o las entradas p_1 y p_2 de la RNA, en el programa utilizado se hace la combinación de todos los valores y se crean los vectores de entrada por medio de los siguientes comandos de MATLAB®:

```
kp = 2.510061;
kd = 0.005266;
e= p(:,1);
dedt = p(:,2);
t = kp*e + kd* dedt;
```

Los valores de `p(:,1)` y `p(:,2)` son los valores de la combinación de A y B, que son multiplicados por `kp` y `kd`, respectivamente. En los anexos se puede ver el programa completo.

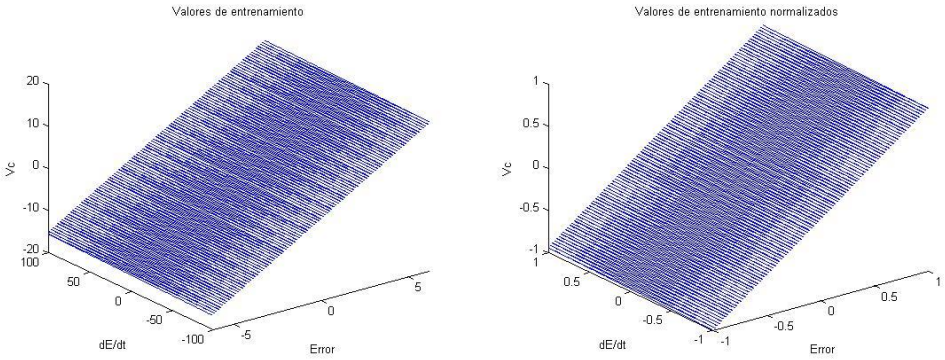


Fig. 5-19 Datos de entrenamiento.

Los datos de entrenamiento, mostrados en la figura anterior, forman un plano, lo cual resulta obvio puesto que se tienen dos ejes para los valores de entrada (el error y la derivada del error) y un eje que representa la salida del controlador (voltaje). Como analogía de ello se puede ver que el comando "`t = kp*e + kd* dedt`", es de la forma $z = ax + by$.

La simulación fue relativamente rápida, la red neuronal convergió en la iteración 2320 en 15 minutos con 23 segundos logrando obtener un error mínimo cuadrático de 9.9 x 10^{-6}.

5.4.2 VERIFICACIÓN DE LA RNA

Después de realizar el entrenamiento se obtuvieron los siguientes parámetros de la red:

```
%Vector de pesos de la capa oculta
W1 = [0.4273 2.5738; 0.0185 0.0006; -0.5967 -0.0199];
%Vector de sesgos de la capa oculta
b1 = [-1.5072; -0.0254; -1.2261];
%Vector de pesos de la capa de salida
W2 = [0.0000 52.2900 -0.0052];
%Vector de pesos de la capa de salida
b2 = 1.3256;
```

Estos parámetros, al ser utilizados en la red de implementación (red de propagación hacia adelante) dan una gran aproximación a los valores de salida dados por los datos de entrenamiento. En la Fig. 5-20 se pueden ver que las gráficas son similares para los datos de entrenamiento y para los valores de la red neuronal entrenada. Cabe mencionar que los datos son puntos ubicados dentro de las ejes X, Y y Z correspondientes al error, derivada del error y salida del controlador.

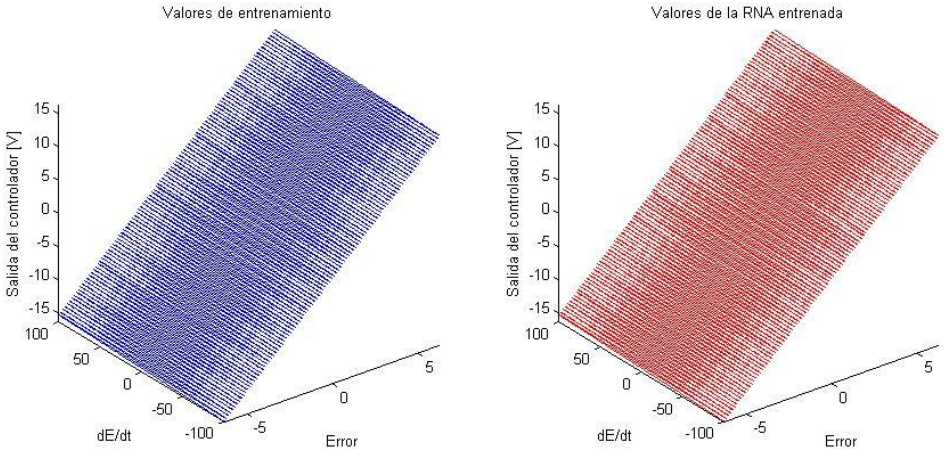


Fig. 5-20 RNA entrenada con los valores del controlador PD.

5.4.3 SIMULACIÓN DE LA RNA

Antes de pasar a la etapa de implementación se simuló el PD en Simulink®. El diagrama de bloques se muestra en la imagen inferior.

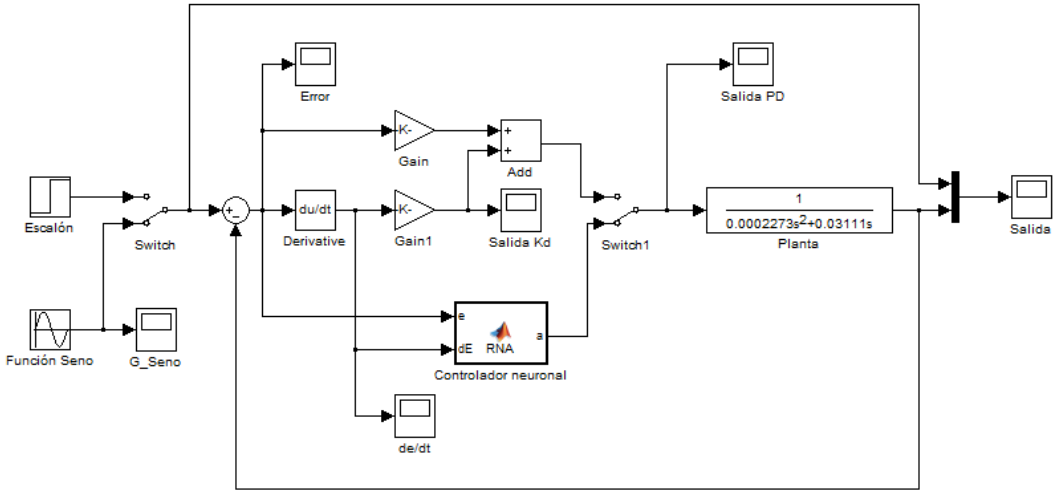


Fig. 5-21 Diagrama de bloques de Simulink® de la simulación de los controladores PD.

El flujo de simulación se repite para este controlador, primeramente se almacenaron los datos de la respuesta del motor debido al controlador PD ante una función escalón y una función sinusoidal; posteriormente, se repitieron este

par de simulaciones usando el controlador neuronal. La Fig. 5-22 permite comparar la respuesta del motor debido a cada controlador.

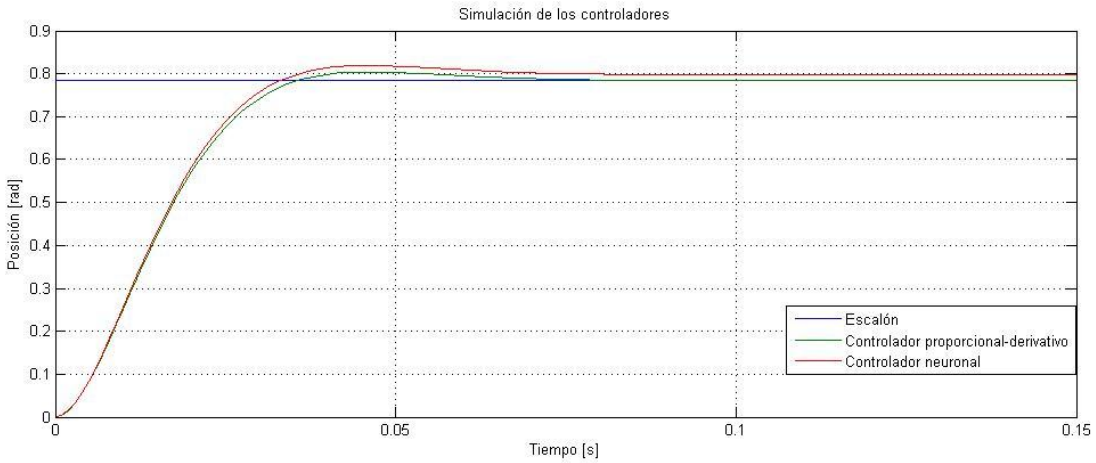


Fig. 5-22 Comparación por simulación de los controladores PD ante una función escalón.

Como se puede apreciar en la figura anterior, la respuesta debida al controlador neuronal no sigue perfectamente la respuesta del controlador PD, solamente al principio y al acercarse al aproximadamente el 70% del error comienza a divergir de los valores del PD.

En el caso de la función de seguimiento, los valores obtenidos de la RNA se asemejan a los del controlador PD, donde el error promedio entre ellos es de 0.0134 [rad] y su diferencia máxima es de 0.0139 [rad].

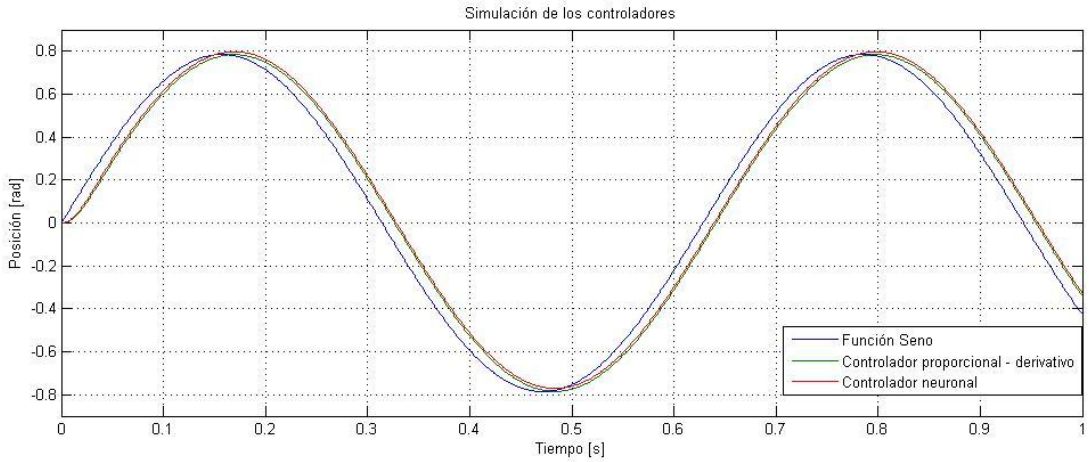


Fig. 5-23 Comparación por simulación de los controladores PD ante una función seno.

Comparando las gráficas generadas en las simulaciones del controlador P y del controlador PD, precisamente en el caso de las entradas escalón, se llega al valor esperado, excepto en el caso del controlador neuronal PD. Durante las pruebas de programación se notó que al incluir como entrada la derivada del error hay desfasamientos en la entrada de la RNA. Este error en estado permanente puede representar una falta de entrenamiento o de otra estructura. También puede representar un error natural del compilador puesto que Simulink® utiliza un depurador de código independiente de MATLAB®, que en este caso fue una herramienta de Microsoft Visual Studio®.

5.4.4 IMPLEMENTACIÓN DE LA RNA EN CompactRIO® Y ALMACENAMIENTO DE DATOS

La implementación de la RNA en cRIO fue de manera similar a la del controlador proporcional, la estructura es la misma que la mostrada en la Tabla 5-4, sin embargo, el VI ejecutado es diferente. El almacenamiento de datos se llevó a cabo también de la misma manera que para el controlador proporcional, en la sección de pruebas y resultados se muestran los detalles de las mismas.

6. PRUEBAS Y RESULTADOS

En este capítulo se muestran los resultados de las pruebas realizadas para demostrar el funcionamiento de las redes neuronales artificiales en el área de sistemas de control. Además, estos resultados permitirán corroborar la teoría analizada en este trabajo.

En el capítulo 3 se mostraron los resultados de las simulaciones de los controladores, estos resultados se compararán con los obtenidos en la plataforma de implementación (CompactRIO), descrita en el capítulo anterior.

Las pruebas realizadas para los controladores (proporcional y proporcional – derivativo) consistieron básicamente en el análisis de dos respuestas, la primera ante una entrada escalón unitario y la segunda por medio de una función de seguimiento; esta función de seguimiento consistió en una función seno a una frecuencia de 125.5 mHz, es decir, un periodo por cada 80 segundos.

6.1 PRUEBAS DEL CONTROLADOR PROPORCIONAL

El controlador proporcional implementado fue el diseñado para obtener un sobrepaso del 2.5%, es decir, para una constante *kp* = 1.835821. La siguiente figura muestra la respuesta del motor debido al controlador ante una función escalón.

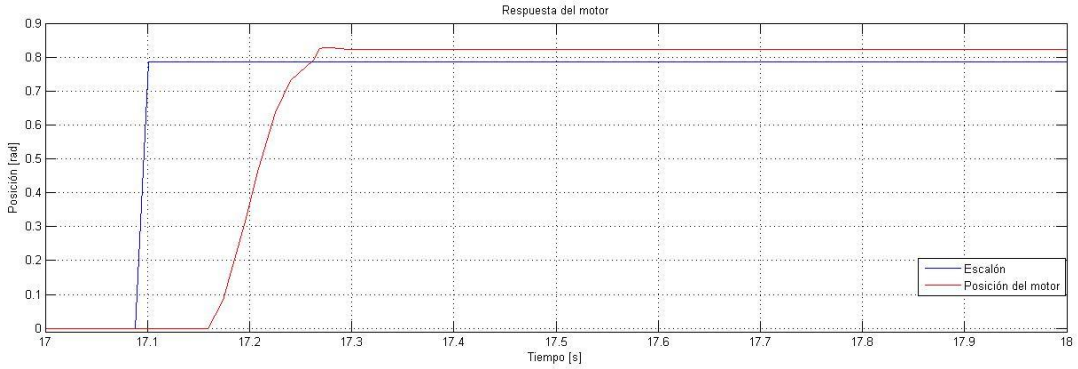


Fig. 6-1 Respuesta del motor de CD con el controlador proporcional (kp = 1.835821).

En la figura anterior se puede apreciar cierto error en estado permanente. Esto es normal, puesto que un voltaje pequeño en la salida del controlador no puede vencer la inercia del rotor del motor, para ello se requiere de cierta compensación, que en este caso, puede ser una parte integral.

Al igual que las pruebas del controlador proporcional, se hicieron las pruebas para el controlador neuronal, ante una función escalón. Se mostraron diferentes comportamientos al realizar 3 pruebas, primeramente mostraba una mayor ganancia, pero en otras pruebas mostraba un resultado muy parecido al controlador proporcional.

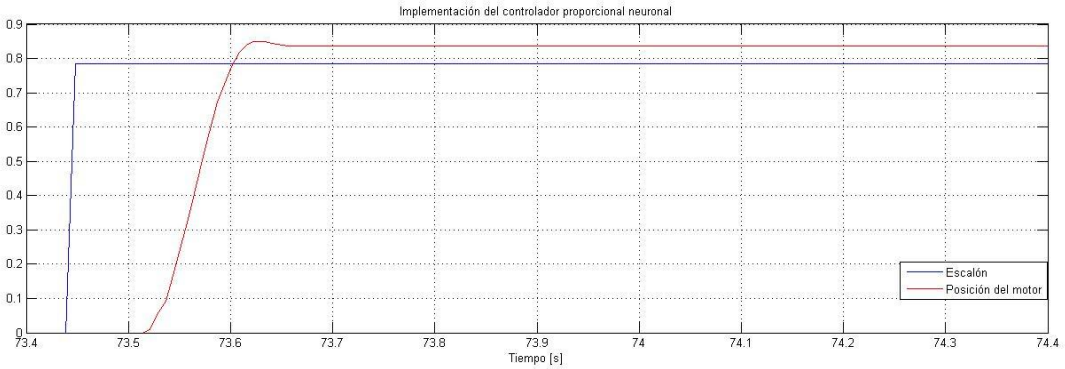


Fig. 6-2 Respuesta del motor de CD con el controlador neuronal (kp = 1.835821)

En la siguiente gráfica se pueden comparar las respuestas de ambos controladores, el controlador proporcional tiene un porcentaje de sobrepaso mayor que el de la red neuronal artificial. Dichas respuestas tienen una forma similar. Sin embargo, la red neuronal responde más tarde que el controlador proporcional; esto se debe a que el cálculo que se debe realizar en la RNA es mayor comparado con la multiplicación correspondiente del error con el constante proporcional k_p. A pesar del retraso inicial, se puede ver que el controlador neuronal hace que la respuesta del motor llegue al valor de sobrepaso máximo más rápido que el controlador proporcional.

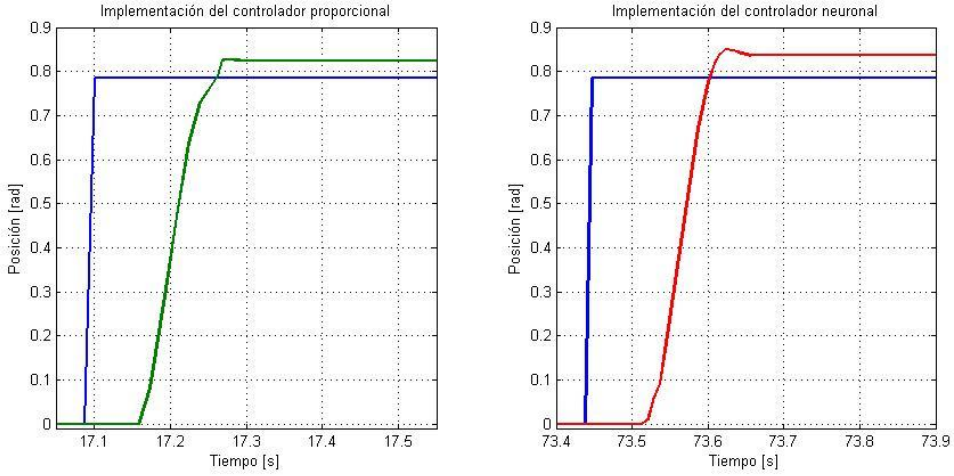


Fig. 6-3 Comparación entre controlador proporcional y neuronal para un porcentaje de sobrepaso de 2.5% (kp = 1.835821).

Tabla 6-1 Comparación de resultados obtenidos para el primer controlador.

Controlador	%Sp	Valor máximo (rad)	Posición en estado permanente (rad)	Error relativo en estado permanente (rad)	Error relativo (°)
Teórico (P)/RNA	2.5	0.8050	0.7854	0	0
Real (P)	5.27	0.8268	0.8237	-0.0383	-2.194
Real (*RNA*)	8.2	0.8498	0.8360	-0.0506	-2.899

En la tabla mostrada anteriormente se puede observar y detallar la diferencia existente entre los sobrepasos reales comparados con el teórico esperado por ambos controladores ante una función escalón. La respuesta debida al controlador proporcional tiene un sobrepaso más grande, 2.77% más que el esperado. En el caso de la respuesta debida al controlador neuronal se tiene un sobrepaso del 5.7%, que representa más del doble del sobrepaso causado por el controlador proporcional implementado. Otra diferencia entre controladores es el tiempo de respuesta, la RNA tarda poco más de 10 milisegundos que el controlador proporcional en responder, esto se entiende debido al tiempo de procesamiento de la red.

En estado permanente se tienen resultados mucho mejores, puesto que en los resultados del error en la posición (en grados) son similares en ambos controladores implementados y la diferencia es de menos de un grado, por lo que se puede asegurar que una buena implementación de una RNA puede asegurar una respuesta muy aproximada a la de un controlador proporcional.

La siguiente gráfica nos permite comparar las respuestas del controlador proporcional y el neuronal ante una función de seguimiento, en este caso una función sinusoidal a una frecuencia de 0.014 Hz.

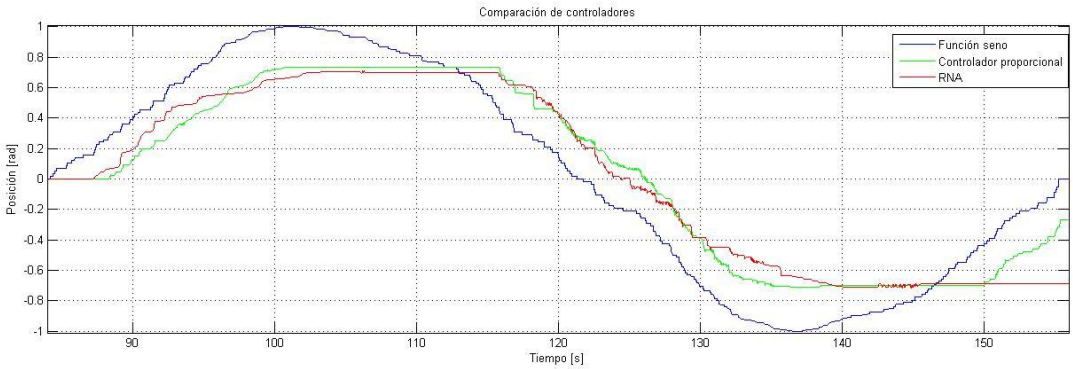


Fig. 6-4 Respuestas de control proporcional y neuronal ante una función seno.

Se puede apreciar que la RNA responde más rápido que el controlador proporcional y presenta un comportamiento similar en las zonas donde el controlador no puede actuar, debido a la inercia del eje del motor. Sin embargo, para criterios de este trabajo, no se toma en cuenta este detalle, puesto que uno de los objetivos particulares es imitar la respuesta de un controlador clásico. Una manera de compensar ese inconveniente es agregando una parte integral al controlador o establecer umbrales en la salida del controlador, pero esto puede provocar una oscilación permanente.

Un detalle importante que se puede señalar es que el uso de las variables globales representó un retraso considerable en la respuesta del controlador, debido a la sincronización de datos en cada subsistema; sin embargo, esto no representó un obstáculo para verificar la convergencia de una RNA, a pesar de no tener una curva se seguimiento muy suave, la respuesta entre ambos controladores fue similar.

6.2 PRUEBAS DEL CONTROLADOR PROPORCIONAL - DERIVATIVO

Al igual que en la prueba para el controlador proporcional se hicieron pruebas para una función escalón y una función de seguimiento sinusoidal. La siguiente gráfica muestra la respuesta ante un escalón con valor de π/4.

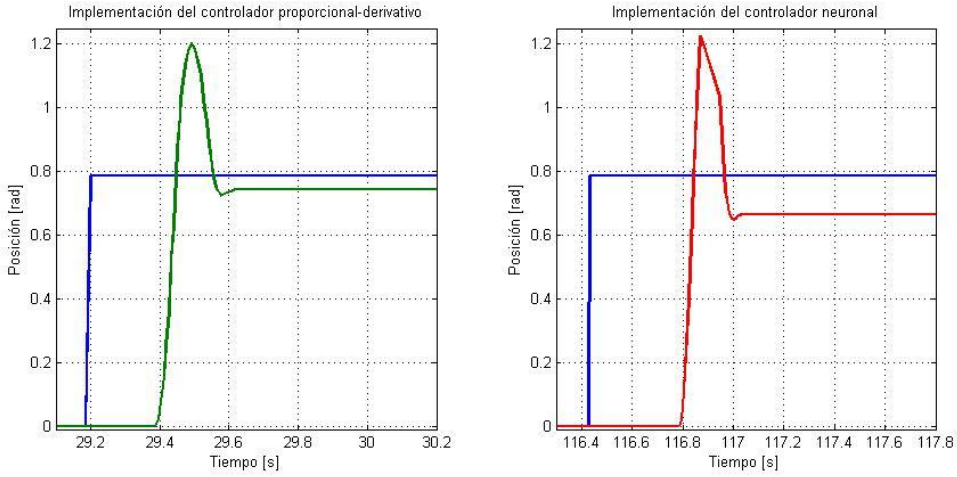


Fig. 6-5 Comparación entre implementación de controlador PD y de controlador neuronal.

De la gráfica anterior se puede observar un retraso en la respuesta del controlador neuronal de aproximadamente 350 milisegundos, comparado con el retraso del controlador proporcional derivativo (70 milisegundos) es un retraso considerablemente grande a pesar de que la amplitud alcanzada por la RNA es prácticamente la misma del otro controlador. El error en estado permanente está presente en ambas respuestas, ninguno de los dos controladores puede vencer la inercia del eje del motor.

Por otra parte, debido al cálculo de las matrices en el programa y a la sincronización de datos se presentaban retrasos en la actualización de los valores para la salida del PWM, por lo que existían, en ocasiones, respuestas con sobrepasos mayores de los diseñados en este trabajo.

Tabla 6-2 Comparación de resultados obtenidos para el segundo controlador.

Controlador	%Sp	Posición máxima (rad)	Posición en estado permanente (rad)	Error relativo en estado permanente (rad)	Error relativo (°)
Teórico (PD)	2.41	0.8043	0.7854	0	0
Teórico (*RNA*)	4.28	0.8190	0.7990	-0.1360	-0.7793
Real (*PD*)	53.11	1.2026	0.7424	-0.0429	-2.4636
Real (*RNA*)	54.49	1.2272	0.6642	-0.1349	-7.7348

La tabla anterior muestra los errores obtenidos en la respuesta debida al controlador PD y al controlador neuronal. Los errores en estado estable son aceptables considerando que en el diseño no se tiene la respuesta buscada (% Sp =

2.5). Los porcentajes de sobrepaso tomados en la implementación rebasan por más del 50% el valor esperado, esto puede ser debido al tiempo de respuesta del controlador, ya que si se hacen pruebas ingresando datos aleatorios, se tienen los resultados esperados en el controlador. Sin embargo, es muy similar el porcentaje de sobrepaso de ambos controladores implementados.

En general, los subprocesos adicionales a la implementación provocan un desfasamiento en la respuesta, por ello se puede entender un sobrepaso más grande de lo esperado, ya que la respuesta de control es más grande cuando se actualiza el estado del error y de su derivada.

En estado permanente existen errores relativos de aproximadamente 2.5 y 7.7 grados. La RNA presenta más dificultades para llegar a la posición deseada, también se entiende esto debido al tiempo de procesamiento mayor para ejecutarse. Puesto que el objetivo de este trabajo no es hacer un controlador muy sensible no se busca añadir algún tipo de compensación para obtener una mejor convergencia en la respuesta final.

La siguiente gráfica muestra la respuesta de ambos controladores ante una función seno.

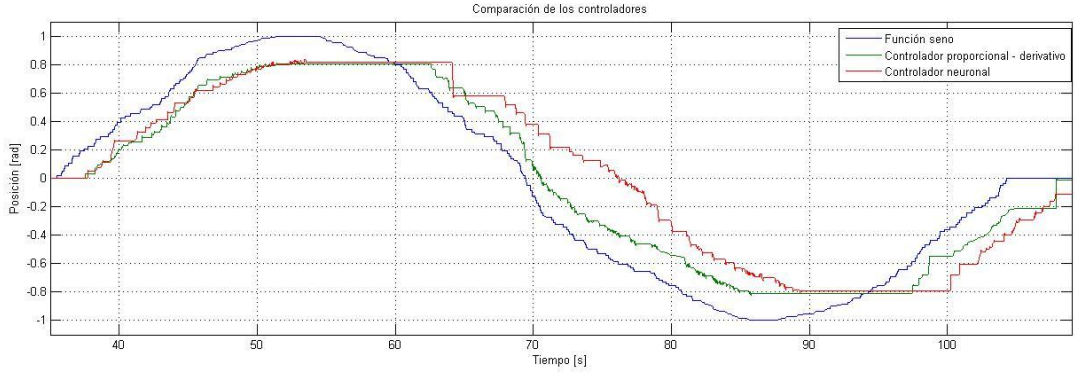


Fig. 6-6 Comparación entre controlador PD y neuronal para un porcentaje de sobrepaso de 2.5% (kp = 1.835821).

La RNA hace un seguimiento muy interesante: desde el comienzo del cambio de la función hasta el valor máximo que alcanza el controlador proporcional - derivativo, ésta se asemeja bastante a la respuesta deseada. Al cambiar de sentido la función, la respuesta debido al controlador neuronal presenta un error mucho más grande y tarda en alcanzar el valor del controlador PD. Lo anterior puede deberse al tiempo de procesamiento, aunado a la inercia del rotor del motor que no logra vencerla después de todo. Es visible, además, que en los puntos más altos y bajos la RNA alcanza los valores esperados.

7. CONCLUSIONES Y TRABAJO A FUTURO

El objetivo principal de esta tesis se cumple, la RNA permitió controlar la posición del rotor del motor de corriente directa, tanto en simulación como en implementación. Con respecto a los resultados obtenidos durante el desarrollo de este proyecto se concluye que las RNA pueden aprender las características de un sistema, por ejemplo, una planta o un controlador. Dichas características aprendidas pueden ser útiles en proyectos futuros, donde se desconoce el modelo de la planta o el tipo de controlador utilizado. Por otra parte, la capacidad de aprendizaje de las redes neuronales puede servir para crear dispositivos que emulen el comportamiento de algún otro sistema. Esto puede ser a partir de un modelo matemático o del almacenamiento de datos de la respuesta del sistema.

Durante las pruebas realizadas se pudo observar el comportamiento de distintas arquitecturas de las redes, así como el tiempo requerido para realizar su entrenamiento. El algoritmo Levenberg-Marquardt mostró ser mucho más eficiente en términos de tiempo y convergencia comparado con la regla delta generalizada y el algoritmo de retro-propagación con *momentum*. El uso de este algoritmo cubre el objetivo particular de convergencia rápida del entrenamiento de la red. La implementación de este algoritmo, en un dispositivo con alta capacidad de procesamiento, a nivel hardware representa una buena opción para crear redes neuronales que se entrenen en línea.

No está de más mencionar que la complejidad de la red se determina a partir de la experiencia del diseñador y/o experiencias externas. La literatura sugiere algunos criterios, mas no impera con leyes o normas de diseño sobre la determinación de la arquitectura. Estas sugerencias fueron seguidas para facilitar la implementación de las RNA. En la etapa de pruebas se mostró que con arquitecturas simples, con pocas neuronas en la capa oculta, puede trabajar bien la red y se logró utilizar redes relativamente pequeñas para aprender la respuesta de los controladores.

De los resultados obtenidos en la implementación se puede afirmar, que las redes neuronales artificiales representan una buena alternativa en el área de control. Sin embargo, es importante señalar que en su implementación se debe contar con un equipo que permita su procesamiento de manera rápida y/o en paralelo, ya que, mientras más grande es la red, se requiere de mayor capacidad del sistema para su implementación. En el caso de este trabajo, a pesar de no contar con redes neuronales grandes, fue difícil el correcto funcionamiento debido a los recursos de programación utilizados, en parte por el tipo de métodos utilizados y por eventos para llevar a cabo la interfaz gráfica y el almacenamiento de datos.

El objetivo de implementar la RNA en un dispositivo de tiempo real se cubre parcialmente, puesto que el objetivo final era ejecutar la RNA en un FPGA. La elección del sistema CompactRIO® como sistema de implementación cubrió la totalidad de criterios para su selección, sin embargo, debido al tipo de datos que maneja y sobre todo a la exclusión del manejo de datos flotantes repercutió en la

decisión de utilizar el procesador de tiempo real para implementar la RNA en vez de hacerlo directamente sobre el FPGA, más que nada por la duración de la síntesis del programa y por la ejecución de más procesos como el PWM y la lectura del encoder. Cada edición del programa requiere repetir el proceso de síntesis y aumenta la duración de este proceso conforme aumenta la complejidad del programa, por lo cual se deja como trabajo a futuro su implementación completa en el FPGA o de otra manera, se recomienda hacer los cálculos de la red neuronal y las acciones de control en un FPGA y la etapa de potencia de manera independiente.

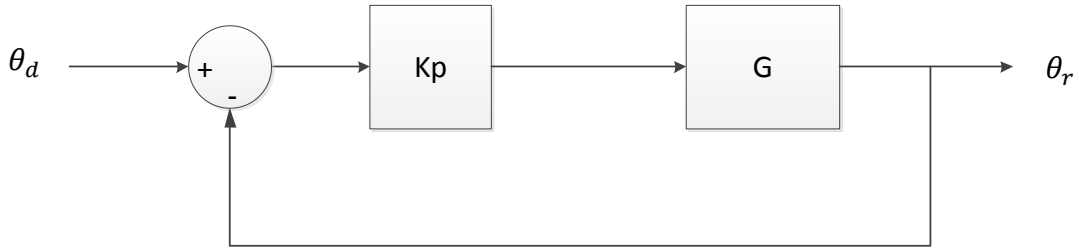
Como comentario adicional, es necesario que se acondicionen bien las señales o los valores de entrada de la RNA, ya que pueden existir errores en los valores finales, más cuando éstos se normalizan y/o su proceso inverso (llevar los valores a la escala original). En el caso de que los valores de entrenamiento sean muy grandes (en orden de miles) y valores muy pequeños (en orden de 10^{-3}) seguramente se requerirá de tener mayor precisión en el factor de escalamiento. Además, es necesario definir el rango de valores para el entrenamiento de la RNA, pues esto también es un factor importante para la duración del entrenamiento de la red así como para la definición del rango de escalamiento.

Considerar las RNA como un controlador único puede representar un gran reto en la implementación, sin embargo, es posible lograrlo con la programación adecuada y depurada para llevar a cabo esto en paralelo, además de un entrenamiento óptimo (entiéndase por ello un entrenamiento con la cantidad de datos suficiente para que la RNA pueda ser generalizada adecuadamente y con una arquitectura que permita realizar esto). Por otra parte, se pueden utilizar las RNA como elementos de apoyo a controladores clásicos y verse como una alternativa para mejorar su desempeño. Incluso, tomando las RNA como un elemento complementario para el área de control de sistemas dinámicos, se pueden utilizar las redes neuronales para instrumentar sensores, sobre todo como alternativa para sensores no lineales. Una RNA pequeña puede representar una opción para obtener mediciones precisas sin la necesidad de linealizar un sensor, por ejemplo, un termistor comercial, en el anexo A.9: Aplicación de una RNA para instrumentar un sensor se muestra el diseño de una RNA para su instrumentación.

Finalmente, se concluye que las RNA tienen una gran capacidad de aprendizaje y pueden tener diversas aplicaciones para el área de control de sistemas dinámicos. Es por ello que en un futuro no debe parecer extraño que sean utilizadas con mayor frecuencia, pues los retos en el área de ingeniería son mayores con el paso del tiempo.

ANEXOS

A.1: Controlador proporcional



$$C_1 = K_p \qquad \text{Ecu. (A.1)}$$

$$G = g\frac{1}{s^2 + as} \qquad \text{Ecu. (A.2)}$$

Obtención de la función de transferencia en lazo cerrado:

$$H_1 = \frac{GC_1}{1 + GC_1} \qquad \text{Ecu. (A.3)}$$

$$H_1 = \frac{\frac{gk_p}{s^2 + as}}{1 + \frac{gk_p}{s^2 + as}} \qquad \text{Ecu. (A.4)}$$

$$H_1 = \frac{gk_p}{s^2 + as + gk_p} \qquad \text{Ecu. (A.5)}$$

La ecuación característica de la función de transferencia H_1 está dada por:

$$s^2 + as + gk_p = 0 \qquad \text{Ecu. (A.6)}$$

Que es similar a la ecuación característica de un sistema de segundo orden.

$$s^2 + 2\xi\omega_n s + {\omega_n}^2 = 0 \qquad \text{Ecu. (A.7)}$$

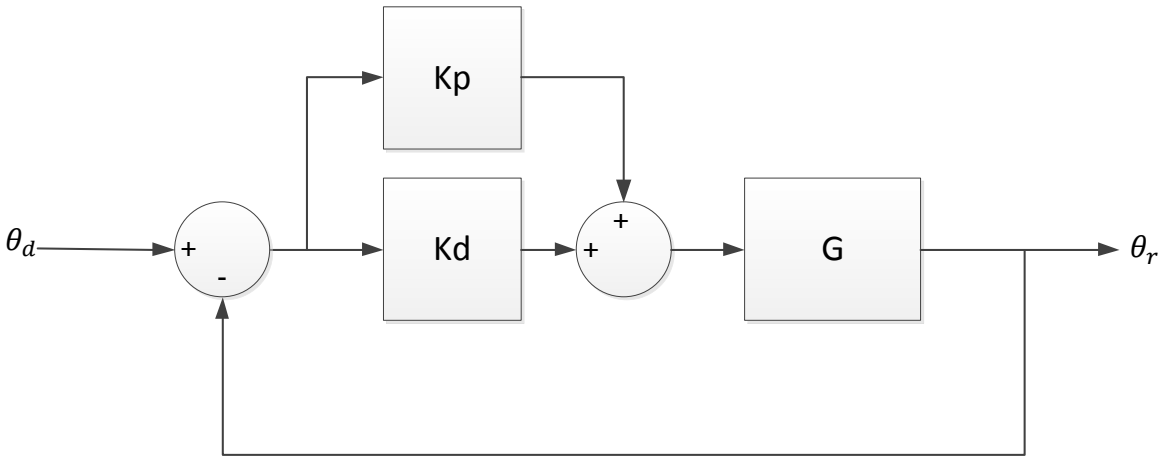
Donde:

$$2\xi\omega_n = a \qquad \text{Ecu. (A.8)}$$

$$\xi = \frac{-\ln\frac{\%Sp}{100}}{\sqrt{\pi^2 + \left(\ln\frac{\%Sp}{100}\right)^2}} \qquad \text{Ecu. (A.9)}$$

$${\omega_n}^2 = gk_p \qquad \text{Ecu. (A.10)}$$

A.2: Controlador proporcional - derivativo



El controlador será un PD de la forma:

$$C_3 = K_p + K_d s \qquad \text{Ecu. (B.1)}$$

$$G = g\frac{1}{s^2 + as} \qquad \text{Ecu. (B.2)}$$

Función de transferencia de lazo cerrado:

$$H_1 = \frac{g\dfrac{K_p + K_d s}{s^2 + as}}{1 + g\dfrac{K_p + K_d s}{s^2 + as}} \qquad \text{Ecu. (B.3)}$$

Simplificando la ecuación anterior tenemos:

$$H_1 = \frac{gK_p + gK_d s}{s^2 + (a + gK_d)s + gK_p} \qquad \text{Ecu. (B.4)}$$

Tenemos entonces la siguiente ecuación característica de segundo orden:

$$s^2 + (a + gK_d)s + gK_p = 0 \qquad \text{Ecu. (B.5)}$$

Que es de la forma:

$$s^2 + 2\xi\omega_n s + {\omega_n}^2 = 0 \qquad \text{Ecu. (B.6)}$$

Donde:

$$\xi = \frac{-\ln\frac{\%Sp}{100}}{\sqrt{\pi^2 + \left(\ln\frac{\%Sp}{100}\right)^2}} \qquad \text{Ecu. (B.7)}$$

$$\omega_n = \frac{4}{\xi\, T_s} \qquad \text{Ecu. (B.8)}$$

$$K_d = \frac{2\xi\omega_n - a}{g} \qquad \text{Ecu. (B.9)}$$

$$k_p = \frac{{\omega_n}^2}{g} \qquad \text{Ecu. (B.10)}$$

Los pasos a seguir para calcular los parámetros del controlador son:

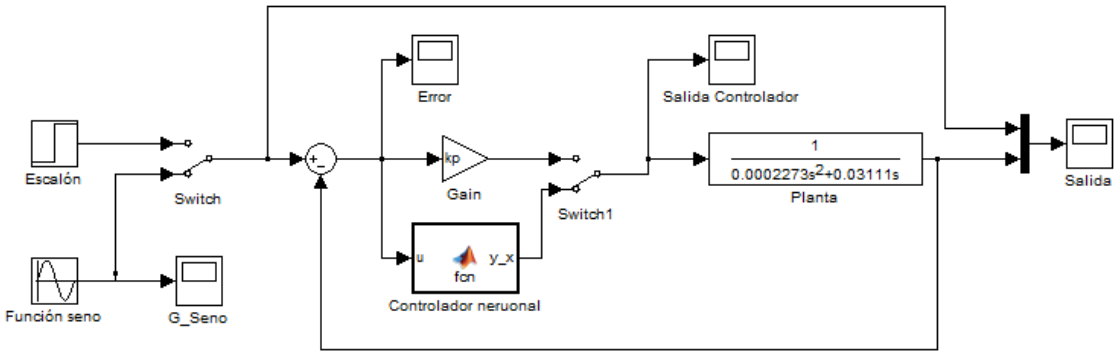
1. Se proponen el porcentaje de sobrepaso *%Sp* y el tiempo de asentamiento *Ts.*
2. Se obtienen los valores de ξ y de ω_n.
3. Se construye la ecuación característica de segundo orden.
4. Se despejan los valores de las constantes de los controladores K_p y K_d.

A.3: Código de MATLAB para entrenamiento de la RNA proporcional

```
% Red de entrenamiento de N capas
clear
clc;

%kp = 1.835821 kp = 2.234272 kp = 2.6293 kp = 3.04485;

%Valores de entrenamiento
kp = 1.835821;              % Coeficiente proporcional
p = [-2*pi:pi/4:2*pi];      % Valores de entrada (Error de posición)
t = kp*p;                   % Valores de salida del controlador

pn = mapminmax(p);          % Normalización del vector p
tn = mapminmax(t);          % Normalización del vector t

%-------Valores para desnormalizar los datos de salida------------
Xo = -1;
X1 = 1;
Yo = min(t);
Y1 = max(t);
%-----------------------------------------------------------------
net = feedforwardnet([2]);              % Red 1 - 2 - 1
net.trainParam.epochs = 50000;          % Número máximo de épocas
[net tr] = train(net, pn, tn);          % Entrenamiento de la red

a = net(pn);                            % Salida de la red
%------------- Almacenamiento de los vectores de pesos y sesgos----
W1 = net.IW{1};
W2 = net.LW{2};
IW = net.IW;
LW = net.LW;
b1 = net.b{1};
b2 = net.b{2};
%-----------------------------------------------------------------

% Desnormalización de los datos de salida
Y = Map(a, Xo, X1, Yo, Y1);

plot(p,t,'blue', p, Y, 'magenta');  % Gráficas
```

A.4: Código de simulación de la RNA proporcional (bloque de Simulink)

Bloque de Simulink® para simular el controlador neuronal.



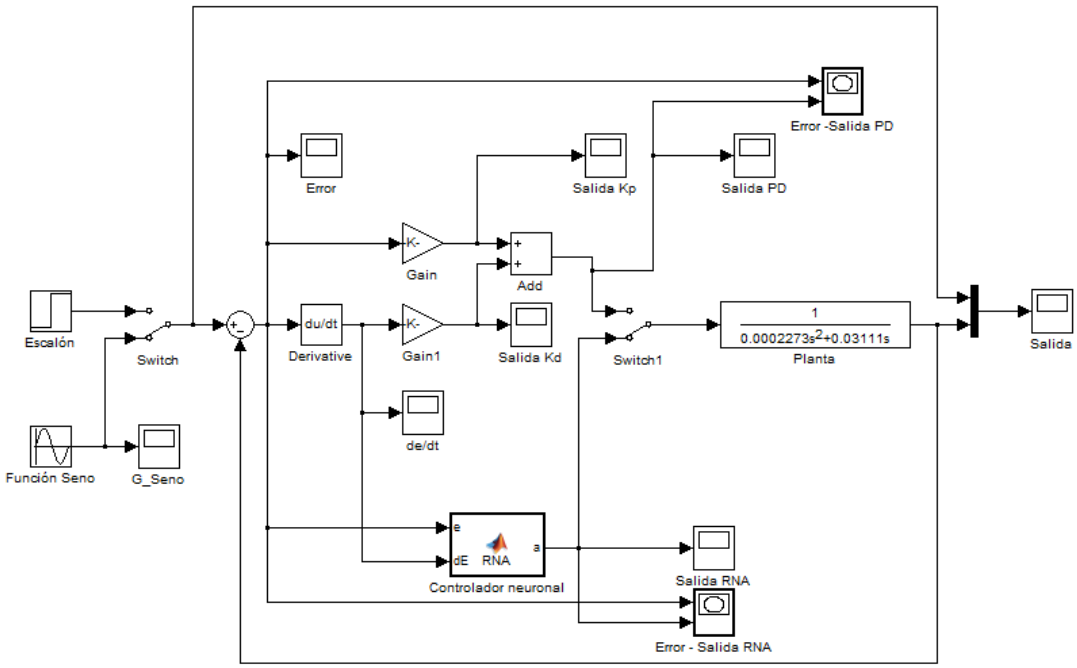
Código del bloque fcn:

```
function y_x = fcn(u)
%#codegen
k_p = 1.835821;                % Valor de kp

p = (-2*pi: pi/4: 2*pi);       % Vector de entrenamiento p,
                               % usado para obtener valores máximos y
mínimos
t = k_p*p;                     % Vector de entrenamiento t,
                               % usado para obtener valores máximos y
mínimos

up = Map(u,min(p),max(p),-1,1); % Normalización de datos

%-------------------- Propagación hacia adelante ---------------------
W1 = [0.1421; 0.1497];         %Vector de pesos de la capa oculta
b1 = [-0.6515; 0.6702];        %Vector de sesgos de la capa oculta

W2 = [5.4261 4.8939];          %Vector de pesos de la capa de salida
b2 = 0.2440;                   %Vector de sesgos de la capa de salida

n = W1*up + b1;
y1 = (exp(n)-exp(-n))./(exp(n)+exp(-n)); % Valor de salida de la capa
oculta
y2 = W2*y1 + b2;                         % Valor de salida de la RNA
%----------------------------------------------------------------------


%-------------------- Re-escalamiento de valores --------------
Xo = -1; X1 = 1; Yo = min(t);Y1 = max(t);
y_x = Map(y2, Xo, X1, Yo, Y1);       % Salida del controlador neuronal
```

A.5: Código de entrenamiento de la RNA de MATLAB para el controlador PD

```
% Red de entrenamiento de N capas
clear
clc;

A = [-2*pi:2*pi/50:2*pi];    % Rango de valores del error
B = [-100:1:100];            % Rango propuesto para los valores de
                             % la derivada del error
n=1;
for i=1:length(A)
    for j=1:length(B)
        V2(n,:)=([A(i),B(j)]);
        n=n+1;
    end
end

p = V2;
de=p(:,1);
dt=p(:,2);

%----------------- Controlador proporcional derivativo ------
kp = 2.510061;       % Constante proporcional
kd = 0.005266;       % Constante derivativo
e= p(:,1);           % El error es la entrada p1
dedt = p(:,2);       % La derivada del error es la entrada p2
t = e*kp + dedt*kd; % Salida del controlador
%-------------------------------------------------------------
[p t];               % Par de entrenamiento

p = p';
t = t';
pn = mapminmax(p);                  % Normalización del vector p
tn = mapminmax(t);                  % Normalización del vector t


net = feedforwardnet([3]);          % Definición de red 2-3-1
net.trainParam.epochs = 50000;      % Número máximo de épocas
[net tr] = train(net, pn, tn);      % Entrenamiento de la red
a = net(pn);

W1 = net.IW{1};                % Vector de pesos de la capa oculta
W2 = net.LW{2};                % Vector de sesgos de la capa oculta

b1 = net.b{1};                 % Vector de pesos de la capa de salida
b2 = net.b{2};                 % Vector de sesgos de la capa de salida

Xo = -1;X1 = 1;Yo = min(t); Y1 = max(t);
Y = Map(a, Xo, X1, Yo, Y1);         % Re-escalamiento de datos

plot(p,t,'blue', p, a, 'magenta');  % Gráficas
```

A.6: Código de simulación de la RNA proporcional (bloque de Simulink)



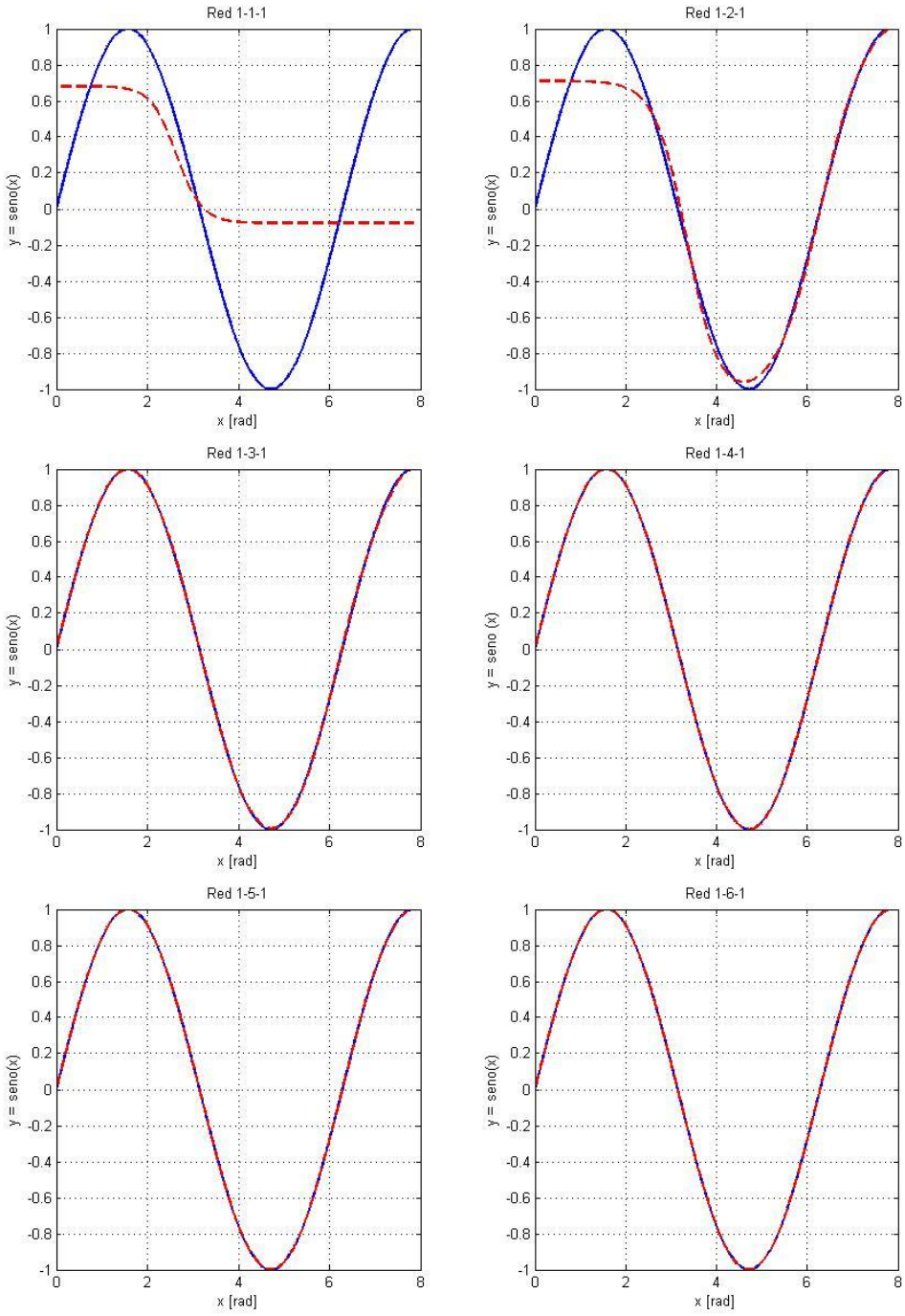

```
function a = RNA(e,dE)
up = Map(e,-2*pi,2*pi,-1,1);     % Normalización del error
vp = Map(dE,-100,100,-1,1);      % Normalización de la derivada
px = [up; vp];
maxt = 16.297778395324499;       % Valor máximo esperado del vector t
%-------------------- Par de entrenamiento -------------------------

%----------------- Propagación hacia adelante----------------------
W1 = [0.4273 2.5738; 0.0185 0.0006; -0.5967 -0.0199];
b1 = [-1.5072; -0.0254; -1.2261];

W2 = [0.0000 52.2900 -0.0052];
b2 = [1.3256];

n = W1*px + b1;
y1 = (exp(n)-exp(-n))./(exp(n)+exp(-n));
y2 = W2*y1 + b2;
%-------------------------------------------------------------

%-------------------- Re-escalamiento de valores ---------------
Xo = -1; X1 = 1; Yo = -maxt; Y1 = maxt;
a = Map(y2, Xo, X1,Yo, Y1);
```

A.7: Código de la función Map

```
% Función de Mapeo
function Y = Map(X,Xo,X1,Yo,Y1)
Y = (X-Xo)*(Y1-Yo)/(X1-Xo)+Yo;
```

A.8: Efecto del número de neuronas en una red de una sola capa oculta

En este anexo se muestra el efecto del número de neuronas para el aprendizaje de la función seno que fue analizada en el capítulo 4 para la selección del algoritmo de entrenamiento. Tal efecto se puede apreciar de manera gráfica en las siguientes imágenes, cada una señala en el título superior la arquitectura de la red.



De las gráficas anteriores se puede observar que para esta función 3 neuronas en la capa oculta son suficientes para aproximarse muy bien a la función, pero aumentando la cantidad de las neuronas se tiene una precisión mucho mayor. Se puede ver, además, que los puntos de inflexión dan una referencia para escoger el número mínimo de neuronas en la capa oculta.

A.9: Aplicación de una RNA para instrumentar un sensor

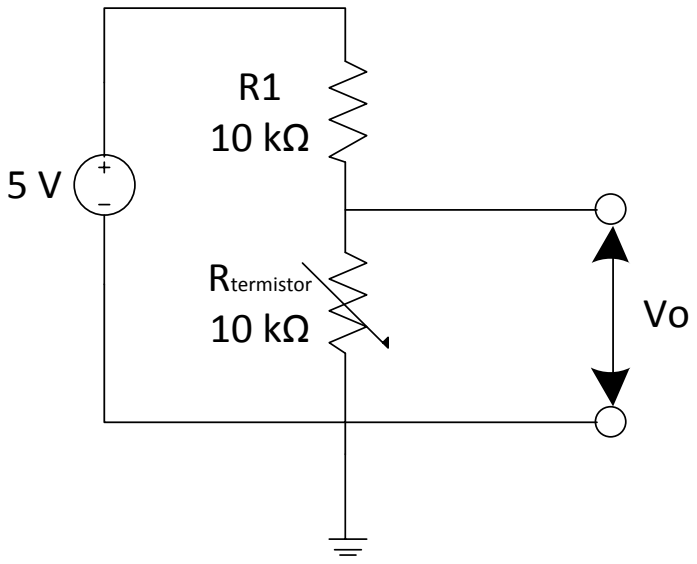
En este anexo se muestra una aplicación de las redes neuronales artificiales en el área de Instrumentación. Se utiliza como ejemplo un termistor NTC comercial 103AT de Semitec, el cual es utilizado frecuentemente en la industria. Su resistencia nominal es de 10 kΩ (a 25°C).

La desventaja de utilizar este termistor es que su salida no es lineal, por lo que se necesita de un método para linealizarlo, de tal manera, que nos facilite el cálculo para determinar la temperatura que se está midiendo.

Existen métodos matemáticos que nos permiten hacerlo, como los métodos numéricos o ecuaciones como la Steinhart-Hart, que es un modelo matemático de la resistencia de un semiconductor a diferentes temperaturas.

Otro método, motivo de este anexo, es entrenar una RNA e implementarla directo en un micro-controlador, FPGA o incluso con amplificadores operacionales. Este método representa una alternativa a los métodos tradicionales.

La siguiente figura muestra la conexión del termistor. Se puede apreciar que está conectada a una resistencia de *pull–up* de 10 kΩ. Se asume que la fuente es de 5 volts.



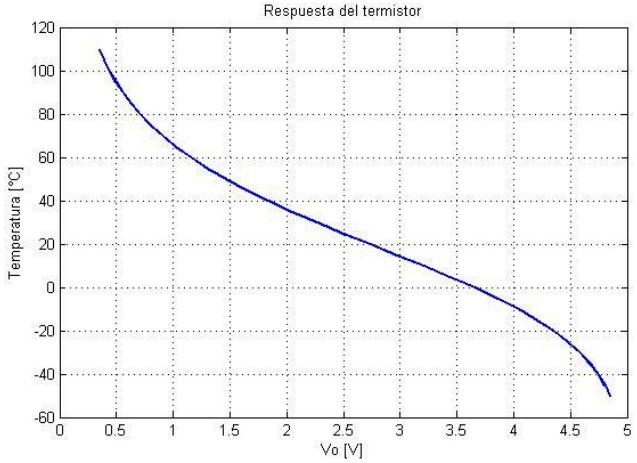
V_o es el voltaje que se obtiene a la salida del divisor de voltaje. Su valor está dado por la siguiente igualdad:

$$V_o = (5\,[V])\frac{R_{termistor}}{R1 + R_{termistor}}$$

La tabla siguiente muestra la respuesta del voltaje de lectura con base en la configuración mostrada anteriormente. Los datos obtenidos son de la hoja de especificaciones del termistor 103AT.

Temperatura [°C]	$R_{termistor}$ [Ω]	V_o [V]	Temperatura [°C]	$R_{termistor}$ [Ω]	V_o [V]
-50	329,000	4.85250737	35	6,940	2.04840614
-45	247,700	4.80597594	40	5,827	1.8408416
-40	188,500	4.74811083	45	4,911	1.64677084
-35	144,100	4.67553537	50	4,160	1.46892655
-30	111,300	4.58779885	55	3,536	1.30614657
-25	86,430	4.48148916	60	3,020	1.15975422
-20	67,770	4.35707856	65	2,588	1.02796314
-15	53,410	4.21148084	70	2,228	0.91102388
-10	42,470	4.04707452	75	1,924	0.80677625
-5	33,900	3.86104784	80	1,668	0.71477545
0	27,280	3.65879828	85	1,451	0.63356912
5	22,050	3.4399376	90	1,266	0.56186757
10	17,960	3.21173104	95	1,108	0.49873965
15	14,690	2.97488862	100	973.1	0.4434025
20	12,090	2.73653237	105	857.2	0.39476108
25	10,000	2.5	110	757.6	0.35212315
30	8,313	2.26969912	--	--	--

En la gráfica siguiente se muestra la relación del divisor de voltaje con las temperaturas mostradas en la tabla anterior.



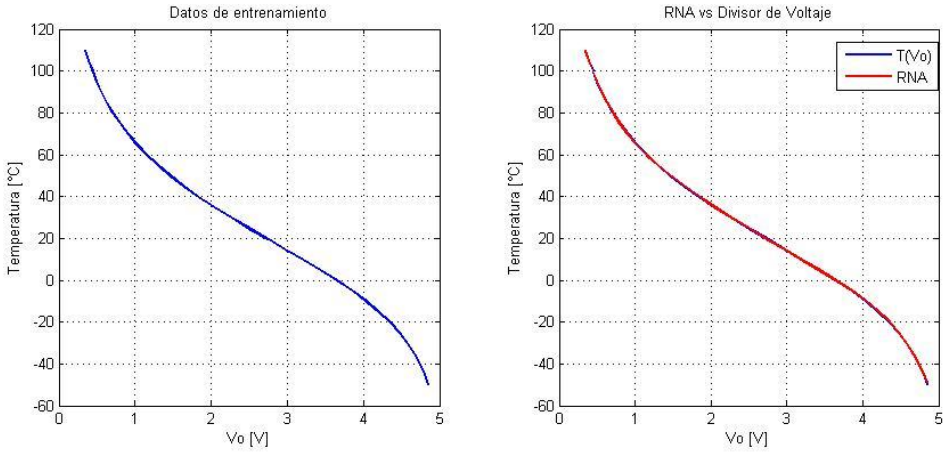
La arquitectura propuesta de la RNA es de una capa oculta con tres neuronas, es decir, una red 1-3-1. No se detalla mucho en la decisión de la arquitectura de la RNA, pues el objetivo de esta sección es mostrar la aplicación de una red neuronal para el área de instrumentación. Como comentario adicional con base en la experiencia del diseñador, una RNA de dos neuronas en su capa oculta puede ser más que suficiente, pero puede tardar mayor tiempo para su entrenamiento que

una de 3 neuronas en su capa oculta. El método de entrenamiento usado fue el algoritmo Levenberg-Marquardt.

Los pesos y sesgos obtenidos en el entrenamiento se muestran a continuación:

```
W1= [-3.0463; 0.1249; 2.8626];  % Vector de pesos de la capa oculta
b1= [5.2064; 0.7471; 4.9139];   % Vector de sesgos de la capa oculta

W2= [12.7599 -8.0822 -13.3474]; % Vector de pesos de la capa de salida
b2= [5.6160];                   % Vector de sesgos de la capa de salida
```

La siguiente figura muestra la gráfica de los datos de entrenamiento comparado con la gráfica de la red neuronal.



La arquitectura propuesta permite un buen entrenamiento de la RNA, se alcanza a apreciar errores ligeros en el empalme de la gráfica, sin embargo, tiene una buena generalización. En la tabla siguiente se muestran los resultados de la RNA a ciertos valores y se comparan con los valores reales de temperatura.

Vo	**Treal**	**T_2 (Salida de la RNA)**	**Error**
4.85250737	-50	-48.6452	-1.3548
4.48148916	-25	-24.9023	-0.0977
3.65879828	0	-0.1857	0.1857
3	**--**	**14.1808**	**--**
2.5	25	25.0570	-0.0570
1.46892655	50	50.1377	-0.1377
1	**--**	**65.9076**	**--**
0.80677625	75	74.8373	0.1627
0.4434025	100	100.3215	-0.3215
0.35212315	110	109.6940	0.3060
0	**--**	**166.9418**	**--**

Los valores mostrados en negritas muestran que la RNA responde a valores intermedios, que no fueron utilizados en su entrenamiento. Esto quiere decir que no es necesario entrenar la red con todos los valores posibles, pueden omitirse

algunos valores, esto también depende de la aplicación. En el caso de funciones sencillas no es crítico tener una gran cantidad de pares de entrenamiento.

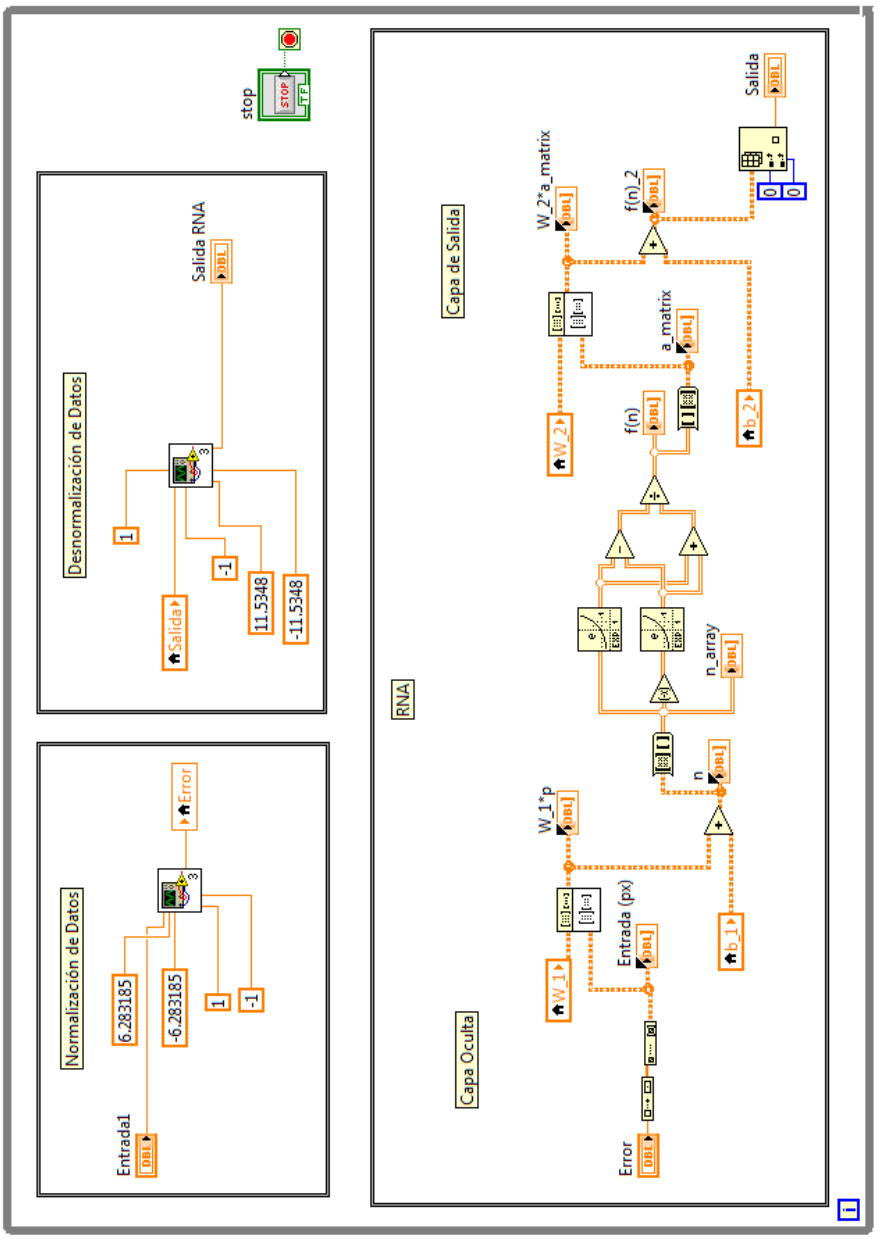
Analizando el último resultado, cuando el voltaje vale "cero" que la RNA estima un valor de la temperatura a pesar de que el último valor de entrenamiento fue de "0.35212315", que corresponde a una temperatura de 110 [°C]. Trasladando esta aplicación al área de probabilidad y estadística, se pueden predecir eventos con base datos registrados en un periodo.

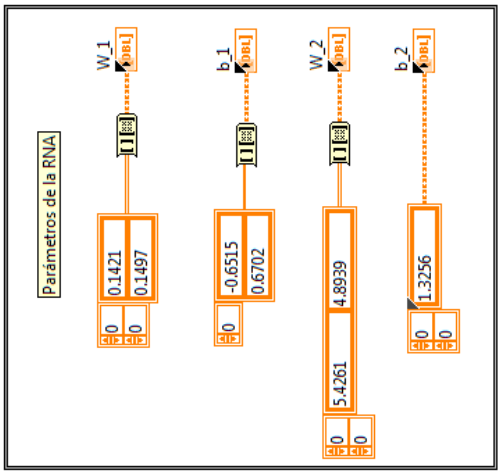
En el caso de esta red, al ser muy pequeña, puede bastar un micro-procesador pequeño para implementarla, ya que el cálculo matricial no es muy grande.

Por último, se muestra un código de MATLAB que permite llevar a cabo la implementación de la RNA.

```
entrada = -1;                          % Valor de entrada
Vomin = 0.3521;                       % Valor mínimo del divisor
VoMax = 4.8525;                       % Valor máximo del divisor
pt = Map(entrada,Vomin,VoMax,-1,1);    % Normalización de la entrada

W1= [-3.0463; 0.1249; 2.8626];  % Vector de pesos de la capa oculta
b1= [5.2064; 0.7471; 4.9139];   % Vector de sesgos de la capa oculta

W2= [12.7599 -8.0822 -13.3474]; % Vector de pesos de la capa de salida
b2= [5.6160];                  % Vector de sesgos de la capa de salida

red = (W2*tansig(W1*pt+b1))+b2;     % Valor de salida de la red

Tmin = -50;                           % Valor mínimo de la temperatura
Tmax = 110;                           % Valor máximo de la temperatura
salida = Map(red,-1,1,Tmin,Tmax)     % Valor de salida mapeado
```

A.10: Programa Matrices





A.11: Interfaz de LabVIEW para implementación del controlador proporcional/ RNA

PANEL FRONTAL

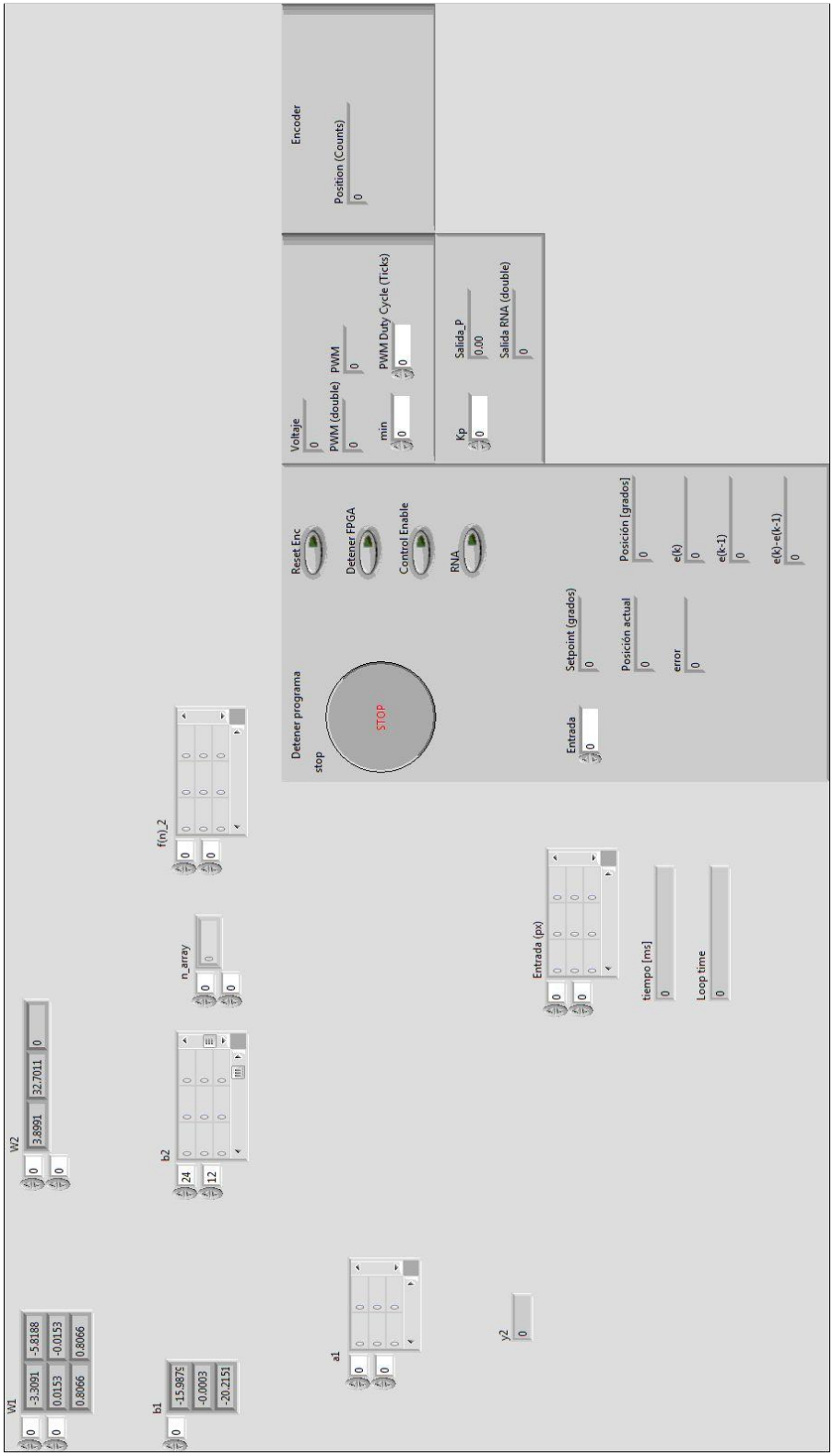


DIAGRAMA DE BLOQUES

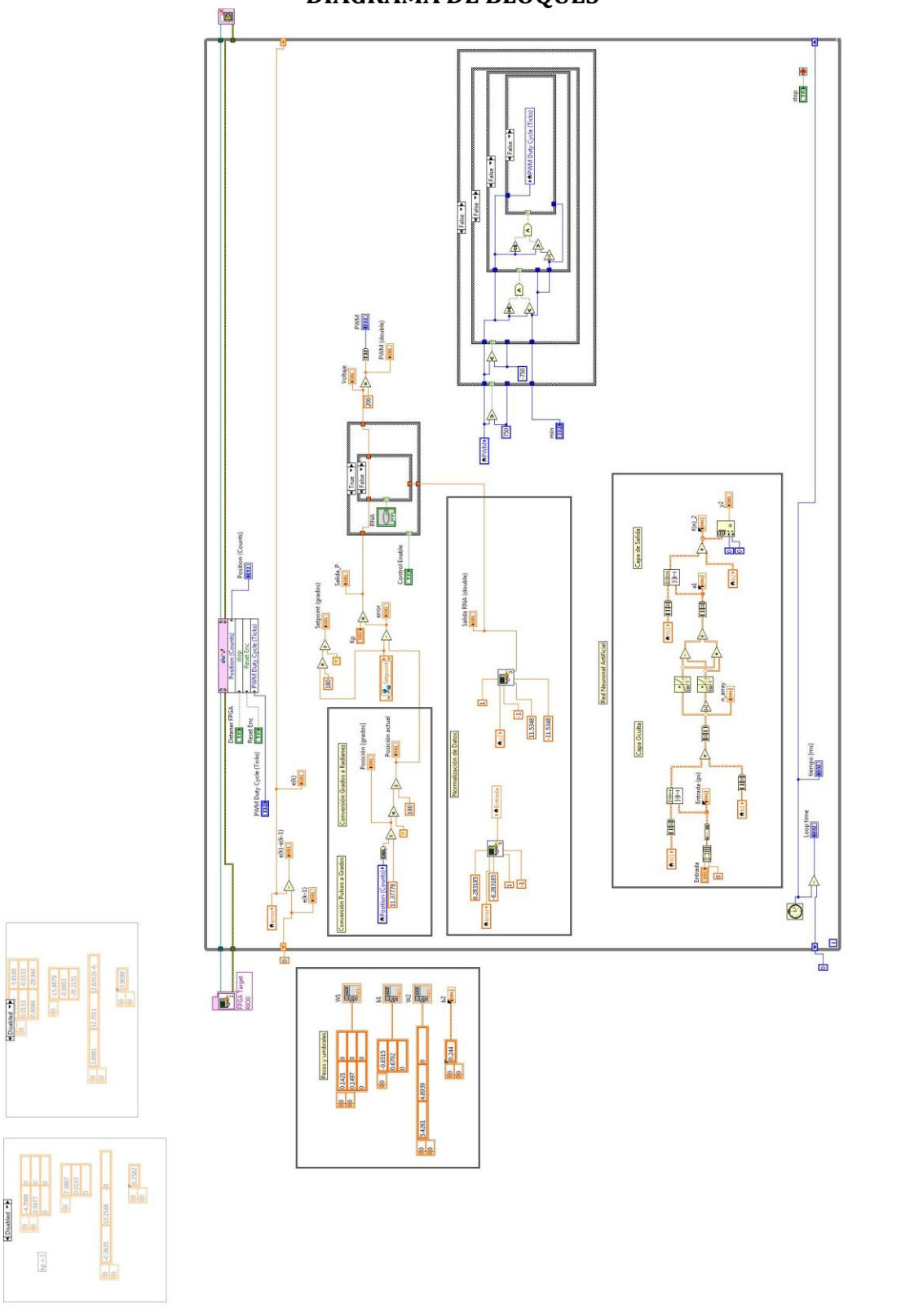
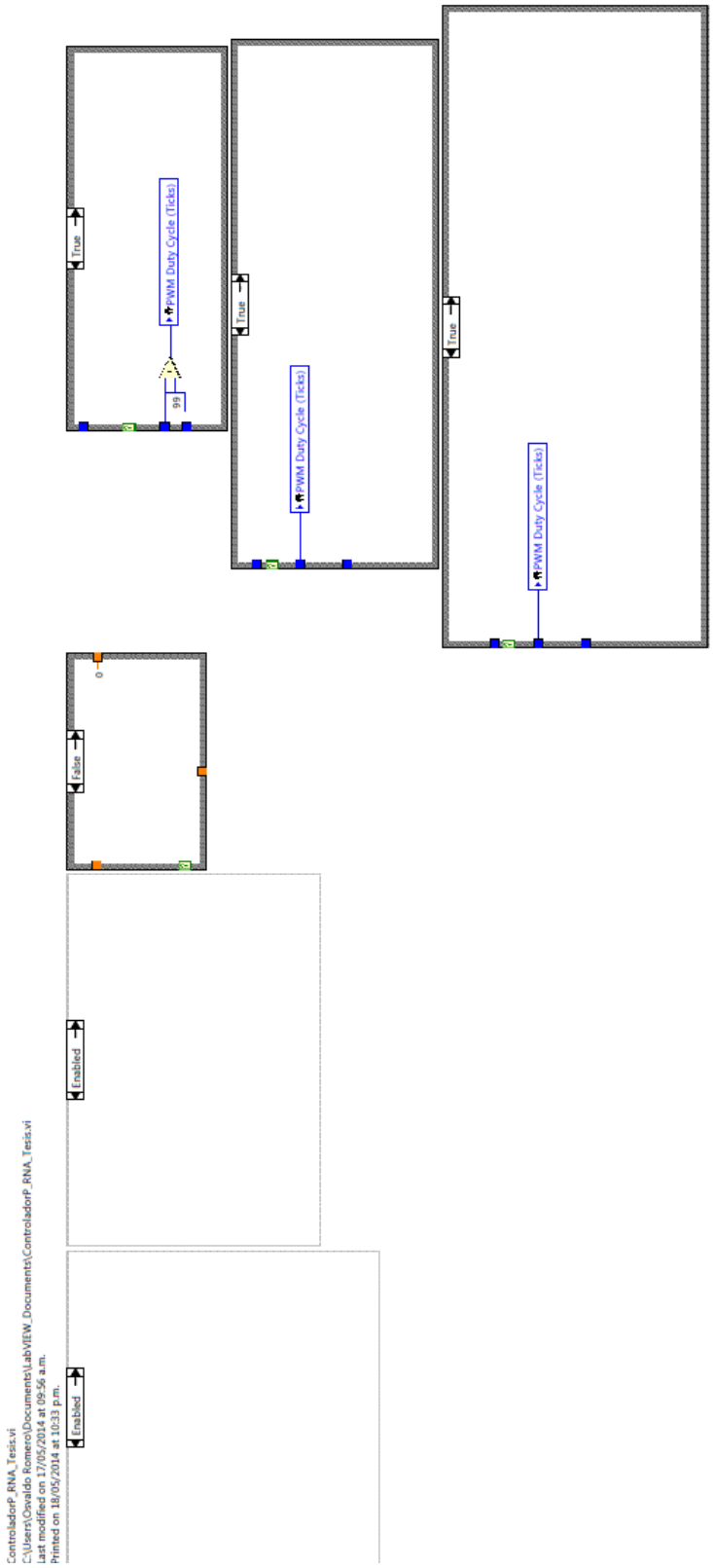


DIAGRAMA DE BLOQUES (CONTINUACIÓN)



A.12: Interfaz de LabVIEW para la implementación del controlador PD/RNA

PANEL FRONTAL

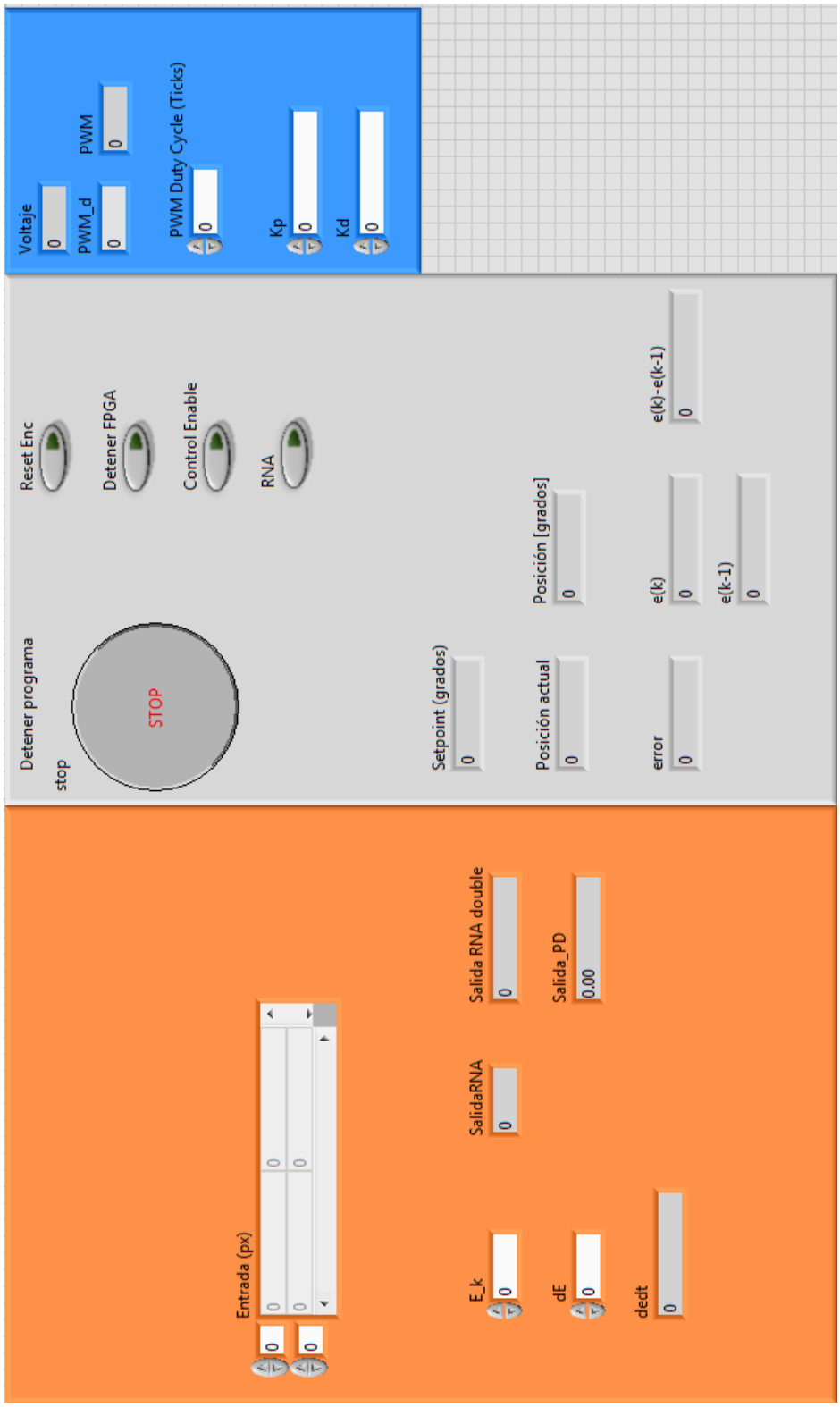
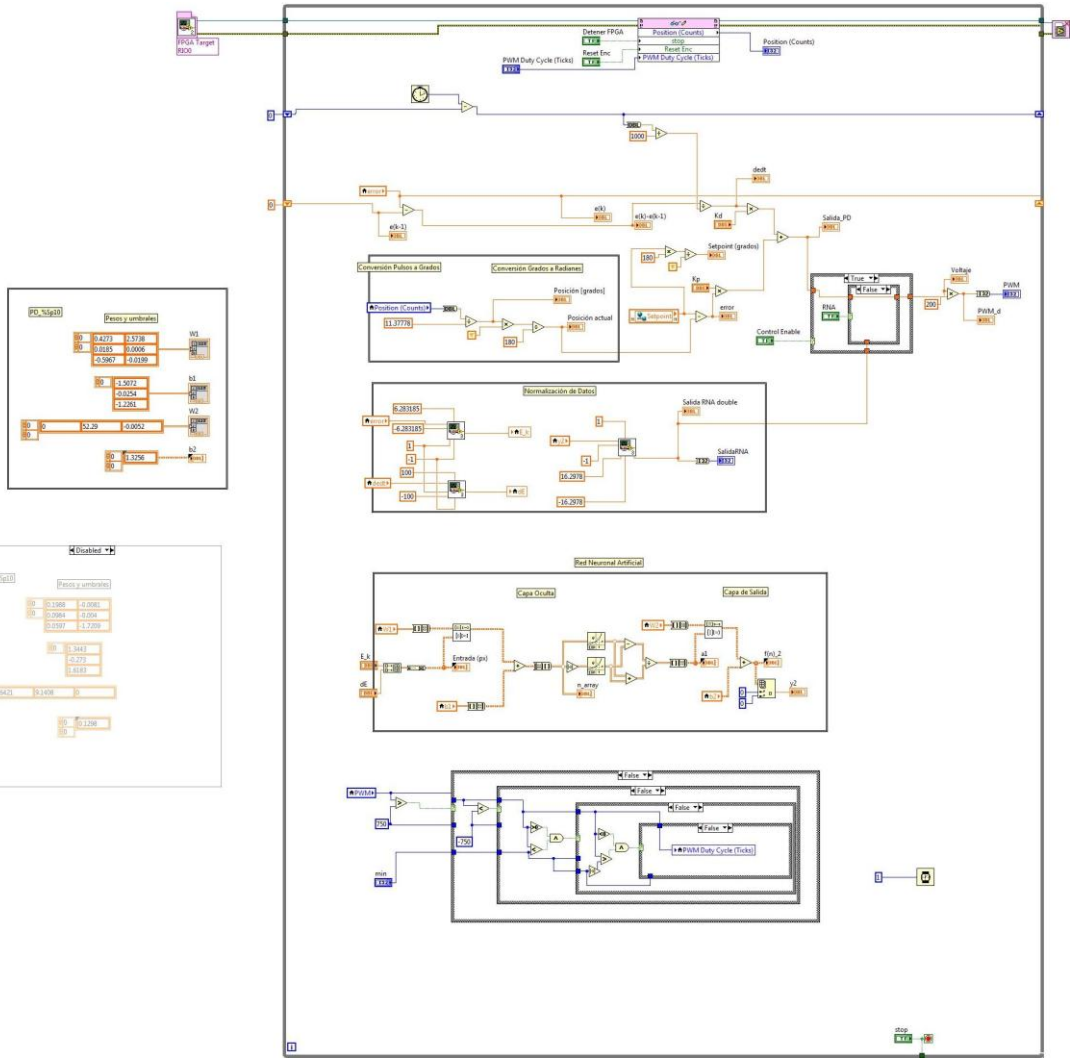
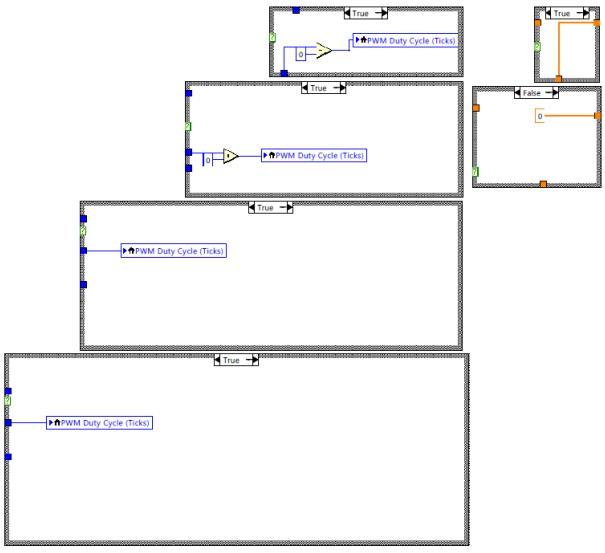


DIAGRAMA DE BLOQUES





A.13: Lista de Figuras

REFERENCIAS

[1.] Antsaklis, P. J. (1990). Neural Networks in Control Systems. *IEEE Control Systems Magazine*, 3.

[2.] Müller, B., & Reinhardt, J. (1991). *Neural Networks. An Introduction.* Berlín, Heidelberg: Springer-Verlag.

[3.] Nguyen, D. H., & Widrow, B. (1990). Neural Networks for Self-Learning Control Systems. *IEEE Control Systems Magazine*, 6.

[4.] Redes Neuronales, Universidad de Antioquia, http://ingenieria.udea.edu.co/investigacion/mecatronica/mectronics/redes.htm

[5.] Pérez Cano, M., & García Martín, J. (2004). *La audición humana*. Recuperado el Mayo de 2014, de Sitio web del Laboratorio de Procesado de Imagen: http://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_04_05/io1/public_html/oidohumano.htm

[6.] Ponce Cruz, P. (2011). *Inteligencia Artificial con Aplicaciones a la Ingeniería.* México, D.F.: Alfaomega.

[7.] Sánchez Camperos, E. N., & Alanís García, A. Y. (2006). *Redes Neuronales. Conceptos fundamentales y aplicaciones a control automático.* Madrid, España: Pearson - Prentice Hall.

[8.] Martín del Brío, B., & Sanz Molina, A. (2006). Redes Neuronales y Sistemas Difusos. Colombia: Alfaomega.

[9.] Haykin, Simon. *Neural Networks. A Comprehensive Foundation.* USA: Prentice Hall, 1999.

[10.] Hagan, Martin T., Howard B. Demuth, y Mark Beale. *Neural Network Design.* China: Citic Publishing House, 2002.

[11.] Ruíz, C. A., & Basualdo, M. S. (2001). *Redes Neuronales: Conceptos Básicos y Aplicaciones.* Cátedra: Informática Aplicada a la Ingeniería de Procesos, Universidad Tecnológica Nacional, Departamento de Ingeniería Química, Rosario.

[12.] Karris, S. T. (s.f.). *Digital Circuit Analysis and Design with Simulink Modeling and Introduction to CPLDs and FPGAs* (Segunda ed.). Fremont, California: Orchad Publications.

[13.] De Barros Ruano, A. E. (1992). *Applications of Neural Networks to Control Systems.* Tesis doctoral, University of Wales, School of Electronic Engineering Science, Bangor.

[14.] Vásquez Choquehuanca, F. (2007). *Diseño e Implementación de Sistema de Control de Tensado y Alineamiento para una Máquina Procesadora de Papel Basado en Redes Neuronales Artificiales y Control PD.* Universidad Nacional Ingeniería, Facultad

Ingeniería Eléctrica y Electrónica, Lima, Perú.

[15.] Pezeshi, S., Badalkhani, S., & Javadi, A. (2012). Performance Analysis of a Neuro-PID Controller Applied to a Robot Manipulator. *International Journal of Advanced Robotic Systems*, 10

[16.] Cozma, A., & Pitica, D. (2008). *Artificial Neural Network And PID Based Control System For DC Motor Drives.* Technical University of Cluj Napoca, Applied Electronics Department, Cluj-Napoca, Rumania

[17.] Galván Rivera, F., Ramírez López, R., & Pérez Paredes, S. (2007). Control NeuroPID de un motor de CD de 180 V. *6to. Congreso Nacional de Mecatrónica*.

[18.] Báez Golubowski, G. I. (2009). *Control de Par de un Motor de Corriente Directa Mediante Redes Neuronales Artificiales, a Través de una PC.* Tesis de licenciatura, Instituto Tecnológico de León, León, Guanajuato.

[19.] Woolf, P. (2013, Febrero). *University of Michigan Chemical Engineering Process Dynamics and Controls Open Textbook*. Consultado en Diciembre de 2013, de MIMO Using Neural Networks: https://controls.engin.umich.edu/wiki/index.php/NN

[20.] *MathWorks Documentation Center*. (2014). Consultado en Enero de 2014, de Neural Network Design Steps: http://www.mathworks.com/help/nnet/gs/neural-network-design-steps.html

[21.] *Corchado, J. M., Díaz, F., Borrajo, L., & Fernández, F. (2000). Redes Neuronales Artificiales. Un enfoque práctico. Vigo, España: Servicio de Publicacións da Universidade de Vigo.*