

Aplicación de red neuronal artificial al control de un convertidor trifásico de condensadores flotantes

Cristóbal Isaías Cortés Barraza

2024

Requisito parcial para obtener el título de:
Ingeniero Civil Electricista

Profesor Guía:

Dra. Margarita A. Norambuena Valdivia (UTFSM)

Comisión:

Dra. Margarita A. Norambuena Valdivia (UTFSM)

Dr. Pablo A. Lezana Illesca (UTFSM)

Valparaíso, 30-10-2024.



Aplicación de red neuronal artificial al control de un convertidor trifásico de condensadores flotantes

Cristóbal Isaías Cortés Barraza

2024

“La curiosidad es la madre de todas las ciencias.”
— *Michael Faraday*

Agradecimientos

El desarrollo de este trabajo cuenta con el apoyo de muchas personas que, directa o indirectamente, han puesto sus buenas intenciones en la finalización de esta etapa.

Quiero comenzar agradeciendo a mis padres, Don Bernardino Cortés y Doña Mónica Barraza, cuyos valores, consejos y, sobre todo, sacrificios me han llevado a ser la persona que soy ahora. Agradezco la perseverancia que han tenido para, a pesar de todo, no fallar a mis hermanos y a mí. Agradezco el apoyo emocional y económico que siempre me han brindado y que he tratado de retribuirles. Agradezco también a mis hermanos, Mónica e Ignacio, a quienes atesoro con mi vida, por todos los momentos que hemos compartido y por su apoyo genuino en lo que he necesitado.

Agradezco a mis amigos de la vida, quienes han estado pendientes de mis logros y frustraciones para darme una mano. Agradezco también a mis compañeros, y ahora amigos, que comparten las mismas emociones y anécdotas que he experimentado durante mi paso por la universidad. Agradezco a la Escuela Santa María por darme el espacio y la confianza para crecer como persona ayudando a otros. A Tamariet que en un corto tiempo a llenado mi vida de alegrías.

Agradezco también a mi profesora guía, Margarita Norambuena, por otorgarme la oportunidad de viajar al extranjero con motivo de este trabajo de memoria, permitiéndome experimentar nuevos horizontes no solo físicos, sino también de vida, abriendo mi mente a nuevas posibilidades.

Agradezco también al Centro Avanzado de Ingeniería Eléctrica y Electrónica (AC3E) por brindarme la oportunidad de colaborar en un estudio técnico para República Dominicana y por permitirme trabajar y familiarizarme con tecnología clave a nivel nacional, como el RTDS.

Finalmente, reitero mi agradecimiento a todas aquellas personas que estuvieron presentes y/o me apoyaron de cualquier manera durante este periodo.

Índice de Contenidos

Índice de Contenidos	I
Índice de Figuras	III
Índice de Tablas	V
Resumen	1
Abstract	2
1. Introducción	3
2. Convertidor de capacitor flotante de cuatro niveles	5
2.1. Topología del convertidor	5
3. Métodos de control	8
3.1. Modelo de control lineal	8
3.1.1. Modulación <i>phase shifted</i> -PWM	9
3.1.2. Esquema de control lineal aplicado	9
3.2. Control predictivo basado en modelo con conjunto de estados finitos . . .	10
3.2.1. Control predictivo basado en modelo aplicado a un FCC	11
3.3. Redes neuronales artificiales	13
3.3.1. Modelo de una neurona artificial	13
3.3.2. Estructura de una red neuronal artificial	16
3.3.3. Modelos de redes neuronales	17
3.3.3.a. Perceptrón network	17
3.3.3.b. Backpropagation network	18
3.3.4. Paradigmas de aprendizaje	18
3.3.5. Entrenamiento de una red neuronal artificial	19
3.3.5.a. Algoritmos de entrenamiento	19
3.3.5.b. Dimensión de una red neuronal	23
3.3.5.c. Vectores de entrada y salida	25
4. Control en base a redes neuronales artificiales aplicado	27
4.1. Control en base a RNA para estrategia de control lineal	27
4.1.1. Datos de entrenamiento para el caso de control lineal	29
4.1.2. Creación y entrenamiento de la RNA para el caso de control lineal .	31
4.1.3. Implementación de la RNA en entorno de simulación para el caso de control lineal	34

4.1.3.a. Análisis de discrepancia en estado estacionario	36
4.2. Control en base a RNA para estrategia de control predictivo	38
4.2.1. Datos de entrenamiento para el caso de control predictivo	40
4.2.2. Creación y entrenamiento de la RNA para el caso de control predictivo	42
4.2.3. Implementación de la RNA en entorno de simulación para el caso de control predictivo	45
4.3. Resultados experimentales en prototipo de laboratorio	48
4.3.1. Configuración de prototipo de laboratorio	48
4.3.2. Plataforma de control BRAIn	49
4.3.3. Implementación de la RNA para la estrategia de control lineal	49
4.3.4. Implementación de la RNA para la estrategia de control predictivo .	52
4.3.5. Observación del costo computacional de los métodos de control en DSP	57
5. Conclusiones.	59
Apéndices	61
A. Códigos de simulación	61
A.1. Control PI con modulación PS-PWM	61
A.2. Control RNA-PI con modulación PS-PWM	64
A.3. Control FCS-MPC	66
A.4. Control RNA-MPC	70
B. Códigos de la Implementación	73
B.1. Control PI con modulación PS-PWM	73
B.2. Control RNA-PI con modulación PS-PWM	81
B.3. Control FCS-MPC	89
B.4. Control RNA-MPC	96
Bibliografía	105

Índice de Figuras

2.1.	Topología del FCC por fase	5
2.2.	Topología trifásica del FCC	6
3.1.	Esquema básico de control lineal	8
3.2.	Esquema de control lineal aplicado	9
3.3.	Esquema de control FCS-MPC [1]	10
3.4.	Modelo de una neurona artificial [2]	13
3.5.	<i>Threshold</i> [2]	14
3.6.	Lineal pura	15
3.7.	<i>ReLU</i>	15
3.8.	Sigmoidea	15
3.9.	Tangente hiperbólica	16
3.10.	Softmax	16
3.11.	Red neuronal <i>feedforward fully conected</i> con una capa oculta [3]	17
3.12.	Estructura red neuronal tipo perceptrón	18
3.13.	Diagrama de distribución del error, algoritmo <i>backpropagation</i>	18
3.14.	Flujo de señal de la neurona j	20
3.15.	Desempeño de la red en función de la cantidad de nodos en la capa oculta	24
4.1.	Diagrama de RNA aplicada a la estrategia de control lineal	28
4.2.	Esquema de control en base a RNA aplicada a la estrategia de control lineal	29
4.3.	Conjunto de datos de entrada para el entrenamiento de la RNA para caso control PI	30
4.4.	Conjunto de datos de salida para el entrenamiento de la RNA para caso control PI	31
4.5.	Gráficos representativos del entrenamiento y desempeño, para el caso de control lineal	33
4.6.	Corrientes y tensiones medidas para control basado en RNA, caso control lineal	34
4.7.	Comparación de corriente en el eje directo entre el control PI y el control en base a RNA	35
4.8.	Comparación de corriente en el eje directo entre el control PI y el control en base a RNA, $R = 15[\Omega]$	36
4.9.	Diagrama de RNA aplicada a la estrategia de control predictivo	39
4.10.	Esquema de control en base a RNA aplicada a la estrategia de control predictivo	40
4.11.	Conjunto de datos de entrada de la fase a para el entrenamiento de la RNA para caso MPC	41

4.12.	Conjunto de datos de salida de la fase a para el entrenamiento de la RNA para caso MPC	42
4.13.	Gráficos representativos del entrenamiento y desempeño, caso control predictivo	44
4.14.	Corrientes y tensiones medidas para control basado en RNA, caso control predictivo	46
4.15.	Comparación de corriente en el eje directo entre el control FCS-MPC y el control en base a RNA	47
4.16.	Diagrama de comunicación BRAIn y FCC de prototipo de laboratorio . .	48
4.17.	Corrientes y tensiones medidas con control mediante RNA, en prototipo de laboratorio	50
4.18.	Corrientes y tensiones medidas con control mediante RNA, medidas por la BRAIn	51
4.19.	Comparación de corriente en el eje directo entre el control PI y el control en base a RNA, en prototipo de laboratorio	52
4.20.	Corrientes y tensiones medidas con control en base a RNA, en prototipo de laboratorio	53
4.21.	Corrientes y tensiones medidas con control mediante RNA, medidas por la BRAIn	54
4.22.	Comparación de corriente en el eje directo entre el control FCS-MPC y el control en base a RNA, en prototipo de laboratorio	55
4.23.	FFT de corriente i_a y tensión V_{aN} para el control FCS-MPC y el control en base a RNA	56
4.24.	Respuesta a escalón de tensión para control FCS-MPC y control en base a RNA	57

Índice de Tablas

2.1.	Estados de conmutación posibles para un FCC de tres celdas	6
3.1.	Combinaciones posibles de estados de conducción	26
4.1.	Resumen de entrenamiento de RNA, para el caso de control lineal	32
4.2.	Comparación de THD entre el control PI y el control de la RNA en función de la referencia de amplitud de corriente	35
4.3.	Comparación del tiempo de asentamiento entre el control PI y el control de la RNA	36
4.4.	Parametrización de los estados de conducción por fase	38
4.5.	Resumen de entrenamiento RNA, para el caso de control predictivo . . .	43
4.6.	Comparación de THD entre el control PI y el control de la RNA en función de la referencia de amplitud de corriente	46
4.7.	Comparación del tiempo de asentamiento entre el control FCS-MPC y el control de la RNA	47
4.8.	Parámetros generales del sistema de prueba de laboratorio	48
4.9.	Comparación de THD entre el control PI y el control de la RNA en función de la referencia de amplitud de corriente, para prototipo de laboratorio .	51
4.10.	Comparación de THD entre el control FCS-MPC y el control en base a RNA en función de la referencia de amplitud de corriente, para prototipo de laboratorio	55
4.11.	Observación de costo computacional para las distintas estrategias de control	58

Nomenclatura.

x	: Valor temporal o variable de la cantidad x .
x^k	: Valor en el instante k de la cantidad x .
x^*	: Valor de referencia para la cantidad x .
$\mathbf{x}$	: Vector de valores de x .
a, b, c	: Sub-índices utilizados para las fasea a , b y c .
o	: Neutro de la carga.
N	: Neutro del inversor.
R	: Resistencia de la carga.
L	: Inductancia de la carga.
C_{ij}	: Condensador de la celda i de la fase j .
v_{ij}	: Condensador de la celda i de la fase j .
m_i	: Índice de modulación de la fase i .
S_{ij}	: Estado de conmutación de la celda i de la fase j .
SS_{ij}	: Estado de conmutación parametrizado i de la fase j .
V_{dc}	: Tensión de dc-link principal.

Resumen

Algunas de las técnicas de control ampliamente utilizadas en electrónica de potencia son el control lineal Proporcional-Integral (PI), debido a su simplicidad y robustez ante variaciones de parámetros, sin embargo, esta estrategia puede ser considerada lenta y/o incompleta, ya que no es adecuado para sistemas no lineales como los convertidores de potencia. Por otro lado, la técnica de control predictivo basado en modelo (MPC, por sus siglas en inglés Model Predictive Control) se caracteriza por su flexibilidad, rápida respuesta dinámica y buen seguimiento de referencia, a pesar de trabajar con error en estado estacionario [4]. No obstante, requiere la iteración de todos los estados de conmutación posibles, lo que implica un mayor tiempo computacional al aumentar el número de estados de conmutación posibles.

En la actualidad se desarrollan diversas aplicaciones de redes neuronales artificiales, una de sus principales características es que no requiere conocer el modelo de un sistema para poder generalizarlo, sumado a que la complejidad de la red se puede adaptar a las necesidades del problema, lo que resulta en una alternativa atractiva a considerar. En el caso de la electrónica de potencia, la posibilidad de reemplazar el control de un convertidor con una red neuronal artificial relativamente simple es la alternativa a validar en este trabajo de memoria.

El sistema de prueba se conforma de un convertidor de tipo “Capacitor flotante” (FCC, por sus siglas en inglés Flying Capacitor Converter) multinivel, el cual alimenta una carga RL. Con esta topología, en entorno de simulación, se opera a diferentes condiciones de operación para las estrategias de control PI y el control predictivo, obteniendo el conjunto de datos de entrenamiento para cada estrategia, con el cual se entrenan las respectivas redes neuronales artificiales. Se valida en simulación el comportamiento del control basado en redes neuronales artificiales para cada estrategia de control. Las estrategias de control basadas en redes neuronales artificiales se implementan en un prototipo de laboratorio, utilizando la plataforma de control BRAIn y un FCC de cuatro niveles.

En el caso de la red neuronal artificial del control lineal, se logra generalizar el comportamiento proporcional, pero no se consigue incorporar el comportamiento integral en simulación y por ende experimentalmente. En el caso de la red neuronal artificial del control predictivo, se logra obtener un comportamiento similar al control MPC, tanto en régimen transitorio como en régimen estacionario, en corriente y en tensión.

Abstract

Some of the control techniques widely used in power electronics are linear control Proportional-Integral (PI) due to its simplicity and robustness to parameter variations. However, this strategy can be considered slow and/or incomplete, since it is not suitable for nonlinear systems such as power converters. On the other hand, the Model Predictive Control (MPC) technique is characterized by its fast dynamic response and good reference tracking, despite working with steady-state error [4]. However, it requires the iteration of all possible switching states, which implies greater computational time as the number of possible switching states increases.

Currently, various applications of artificial neural networks are being developed. One of their main characteristics is that they do not require knowledge of a system model to generalize it. Additionally, the complexity of the network can be adapted to the needs of the problem, making it an attractive alternative. In the case of power electronics, the possibility of replacing the control of a converter with a relatively simple artificial neural network is explored in this study.

The test system consists of a multilevel Flying Capacitor Converter (FCC), which supplies an RL load. In a simulation environment, different operating conditions are tested for PI control strategies and predictive control, generating training data sets for each strategy. These data sets are used to train the respective artificial neural networks. The control behavior based on artificial neural networks for each control strategy is validated in simulation, and then the neural network-based control strategies are implemented in a laboratory prototype using the BRAIn control platform and a four-level FCC.

For the artificial neural network of linear control case, it is possible to generalize the proportional behavior, but it is not possible to incorporate the integral behavior in both simulation and experimental setups. For the artificial neural network of predictive control case, behavior similar to MPC control is achieved, both in transient and steady states, in terms of current and voltage.

Capítulo 1

Introducción

En electrónica de potencia, la calidad de conversión de energía eléctrica desde un sistema de tensión continua a un sistema de tensión alterna se condiciona principalmente por el tipo de convertidor (2 niveles o multinivel) y la estrategia de control que lo gobierne, tales como control lineal, predictivo, entre otros. En cuanto a las estrategias de control para convertidores de potencia, se ha observado en los últimos años un creciente interés en controles inteligentes y adaptativos, ya que permiten controlar múltiples variables y mejorar la rapidez de la respuesta dinámica.

En los últimos años, se ha ampliado el uso del control predictivo basado en modelo con conjunto de estados finitos (FCS-MPC, por sus siglas en inglés Finite Control Set MPC) en diversos convertidores de potencia, debido a su simplicidad conceptual y a la habilidad de controlar múltiples variables con una dinámica de respuesta rápida. La estrategia de control FCS-MPC toma en cuenta todos los estados de conmutación posibles en la implementación del algoritmo, lo cual conlleva un alto costo computacional si la estrategia de control se implementa en convertidores multinivel de tipo FC, lo que presenta un desafío ante el hardware de las plataformas estándar de control. Si bien existen métodos que disminuyen el costo computacional del control FCS-MPC [5], estos algoritmos se encuentran fuertemente ligados al espacio de conmutaciones posibles cuya dimensión depende directamente de la cantidad de niveles del FCC. Por ello, resulta interesante la implementación de estrategias de control que contemplen las ventajas del FCS-MPC, pero que desliguen, en la medida de lo posible, la dependencia del número de niveles del convertidor.

La popularización del uso de redes neuronales artificiales (RNA) en diversas aplicaciones de la ingeniería se debe a su versatilidad y a la amplia gama de recursos tecnológicos y bibliográficos disponibles. Esta tendencia se ve respaldada, además, por la simplicidad que ofrecen los hiperparámetros que definen la estructura de la red neuronal artificial. En el área de convertidores de potencia y accionamientos eléctricos, el creciente desarrollo de microcontroladores ha permitido implementar soluciones basadas en redes neuronales artificiales en estrategias de control y estimación de variables.

Los controladores en base a redes neuronales artificiales tienen ciertas ventajas en comparación a los métodos clásicos de control, tales como: 1) no requerir modelos matemáticos del sistema para ejecutar el control, dado que las ANN consideran todo el sistema como una caja negra [6], 2) estos controladores tienen parámetros claramente definidos, los cuales son la cantidad de nodos y las funciones de las capas internas y externas, 3) se ha

demostrado [7] que las ANN disminuyen el costo computacional asociado a la evaluación de un gran número de estados posibles de conducción en convertidores multinivel cuando se emplean estrategias de control predictivas.

Las desventajas implicadas en la implementación de una red neuronal artificial son la falta de métodos específicos para la definición de los hiperparámetros, el tratamiento de las entradas y salidas y las técnicas de entrenamiento de la red, sin embargo, se pueden determinar de manera iterativa y utilizando técnicas presentes en la literatura [3, 8]

La estrategia de control en base a redes neuronales artificiales puede emular el comportamiento del control predictivo y a su vez, debido al funcionamiento de las ANN, no condiciona el algoritmo a iterar dentro de un espacio de estados de conmutación, otorgando una alternativa a la implementación del control FCS-MPC.

Las tareas de caracterización del control del convertidor multinivel FC mediante la simulación de diversas condiciones de operación bajo el régimen de control lineal PI y del control FCS-MPC, la determinación de la estructura y de los hiperparámetros de la RNA, el tratamiento de las entradas y salidas del sistema y a los parámetros de entrenamiento, para la posterior implementación del control basado en redes neuronales artificiales en un prototipo de laboratorio conforman el desafío abarcado en este trabajo de memoria.

Capítulo 2

Convertidor de capacitor flotante de cuatro niveles

El convertidor que se utilizará en este trabajo es un convertidor trifásico de capacitor flotante de cuatro niveles. Se presentará su topología y se definirán variables que se utilizarán más adelante.

2.1. Topología del convertidor

El FCC [9] [10] de cuatro niveles está compuesto en cada fase por un arreglo de tres unidades base denominadas celdas, ver figura 2.1. Cada celda consiste en un capacitor y dos conmutadores de potencia que deben ser operados de manera complementaria para evitar cortocircuitos.

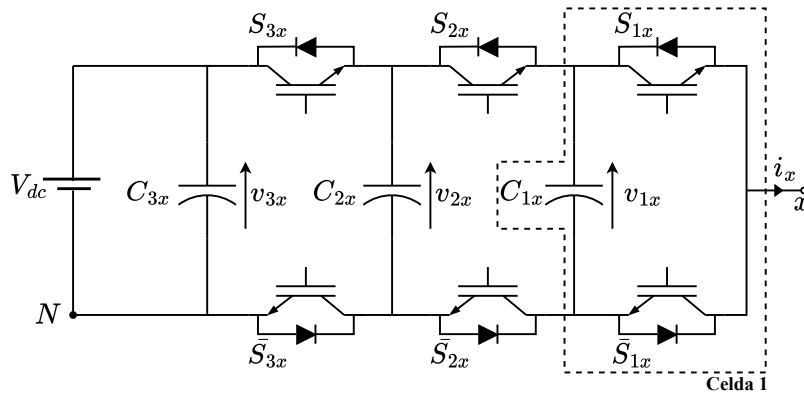


Figura 2.1. Topología del FCC por fase

Donde S_{ix} es el estado de conducción del conmutador de potencia de la celda i de la fase x , i_x la corriente de salida del convertidor de la fase x , C_{ix} es el capacitor de la celda i de la fase x , v_{ix} es la tensión en el capacitor de la celda i de la fase x con $i \in \{1, 2, 3\}$ y $x \in \{a, b, c\}$.

Las tensiones en los condensadores v_{2x} y v_{1x} son medidas entre los terminales de cada condensador y no con respecto al neutro N , es decir, la tensión en los condensadores está

“flotando”.

Cada arreglo de celdas por fase está conectado a una carga RL trifásica con una resistencia de $R = 15[\Omega]$ e inductancia de $L = 10[mH]$, cuyo neutro es el nodo o , como se muestra en la figura 2.2.

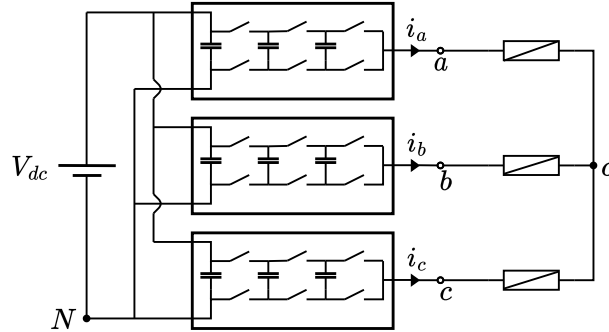


Figura 2.2. Topología trifásica del FCC

Las tres celdas de esta topología permiten conmutar entre cuatro niveles de tensión. Para igualar la tensión de bloqueo de cada semiconductor y conseguir una correcta modulación, la proporción entre la tensión de los condensadores y la tensión V_{dc} es tal que $v_{1x}^* = \frac{V_{dc}}{3}$ y $v_{2x}^* = \frac{2V_{dc}}{3}$, con $x \in \{a, b, c\}$. Esta condición se denomina operación balanceada y se puede conseguir de manera sencilla con técnicas de modulación. Los niveles de tensión posibles se resumen en la tabla 2.1.

Tabla 2.1

ESTADOS DE CONMUTACIÓN POSIBLES PARA UN FCC DE TRES CELDAS

Estados de conmutación			Tensión de salida	
S_{3x}	S_{2x}	S_{1x}	v_{xN}	Nivel de salida*
1	1	1	V_{dc}	V_{dc}
1	1	0	$V_{dc} - v_{1x}$	$\frac{2V_{dc}}{3}$
1	0	1	$V_{dc} - v_{2x} + v_{1x}$	
0	1	1	v_{2x}	
1	0	0	$V_{dc} - v_{2x}$	$\frac{V_{dc}}{3}$
0	1	0	$v_{2x} - v_{1x}$	
0	0	1	v_{1x}	
0	0	0	0	0

*Asumiendo una operación balanceada

Como se observa en la tabla 2.1 los niveles de tensión de salida son combinaciones de las tensiones del *dc-link* y las tensiones en los condensadores de cada celda.

Las principales ventajas de esta topología son:

- Permite el control del flujo de potencia activa y reactiva.
- No utiliza diodos de interconexión o *clamped* en comparación con la topología Neutral Point Clamped (NPC) para la misma cantidad de niveles, lo que reduce las pérdidas del sistema.

- Al igual que la topología NPC, requiere de un solo *dc-link* para alimentar a todo el convertidor.
- Los condensadores presentes en cada celda permiten almacenar mayor cantidad de energía en el convertidor la cual aumenta la capacidad de solventar fenómenos transitorios.

Las principales desventajas son:

- El número y tamaño de los condensadores aumenta en relación con la cantidad de niveles, añadiendo costo y riesgo de falla.
- Requiere consideraciones especiales en el control o modulación para mantener un balance de tensión correcto en cada uno de los condensadores flotantes.
- El condensador principal debe estar diseñado para soportar la tensión máxima del convertidor, requiriendo para el caso de tensiones muy elevadas un arreglo de condensadores.

Capítulo 3

Métodos de control

3.1. Modelo de control lineal

El esquema de control lineal es ampliamente empleado en la industria. La familia de controladores PID (Proporcional-Integral-Derivativo) han demostrado ser una solución robusta en múltiples aplicaciones. Dependiendo de las necesidades de control y del modelo de planta, esta familia de controladores puede contener combinaciones de las distintas acciones de control: P, PI, PD y PID. En este trabajo se implementará un control proporcional integral.

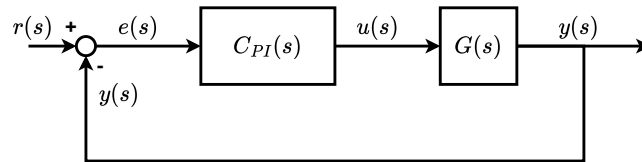


Figura 3.1. *Esquema básico de control lineal*

Considerando el diagrama de control de la figura 3.1, la expresión estándar para el control PI se describe mediante la función de transferencia entre el error relativo $e(s) = r(s) - y(s)$ y la salida del controlador $u(s)$, $\frac{u(s)}{e(s)} = C_{PI}(s)$.

$$C_{PI}(s) = K_p \left(1 + \frac{1}{T_r \cdot s} \right) \quad (3.1)$$

Donde s es el operador de Laplace, K_p es la constante de proporcionalidad y T_r la constante de la acción integral del control o *reset time*.

La acción proporcional del control provee una acción correctiva que depende del valor instantáneo del error relativo. La acción integral, por su parte, adiciona a la acción de control un valor proporcional al error acumulado, lo que implica una reacción lenta dependiente de la duración en el tiempo del error.

3.1.1. Modulación *phase shifted*-PWM

La modulación *Phase Shifted* PWM (PS-PWM) es una técnica que permite mantener el balance natural de tensión en los capacitores del FCC.

En PS-PWM para un FCC, cada celda de una fase es modulada por una señal portadora. Las portadoras triangulares son de igual amplitud, offset y frecuencia, pero con fases distintas, de modo que, el desfase entre dos señales portadoras consecutivas es $2\pi/n$ donde n es la cantidad de celdas del convertidor. Aplicando este desfase a las señales portadoras, y considerando que la referencia varía lentamente con respecto a ellas, se generan ciclos de carga y descarga de igual duración en cada capacitor, provocando de esta manera una característica natural de auto-balance de tensión en los capacitores.

Para el caso del FCC de cuatro niveles que se compone de tres celdas, el ángulo de fase de cada señal portadora para cada celda es $\phi_{1x} = 0$, $\phi_{2x} = 2\pi \cdot \frac{1}{3}$ y $\phi_{3x} = 2\pi \cdot \frac{2}{3}$ con $x \in \{a, b, c\}$.

3.1.2. Esquema de control lineal aplicado

Dadas las características de auto-balance en la tensión en los capacitores del FCC de la modulación PS-PWM, solo se requiere control en la corriente de salida del convertidor, la cual se expresa en el sistema de coordenadas dq .

El convertidor alimenta una carga RL trifásica balanceada, la función de transferencia $G(s) = \frac{v_{sr}}{i_{sr}}$ se modela

$$G(s) = \frac{1}{sL + R} \quad (3.2)$$

El FCC con modulación PS-PWM se modela como un sistema de primer orden con un retardo caracterizado por la constante de tiempo h , que se ajusta considerando el tiempo de cálculo variable de la plataforma de control. Este ajuste permite fijar el tiempo que tarda el conjunto modulador-convertidor en aplicar un escalón de tensión. Para un tiempo de muestreo T_s , las tres señales portadoras triangulares de la modulación, con interrupciones en cada máximo y mínimo, se definen por un período $T_c = 6T_s$. Por lo tanto, la constante de tiempo se define como $h = \frac{3T_c}{2}$.

$$G_{PWM}(s) = e^{-sh} \frac{V_{dc}}{2} \quad (3.3)$$

Realizando la conversión a tiempo discreto del controlador y de la planta, aplicando el respectivo esquema anti-enrollamiento y el limitador de tensión, el esquema de control aplicado al convertidor resulta en el esquema mostrado en la figura 3.2.

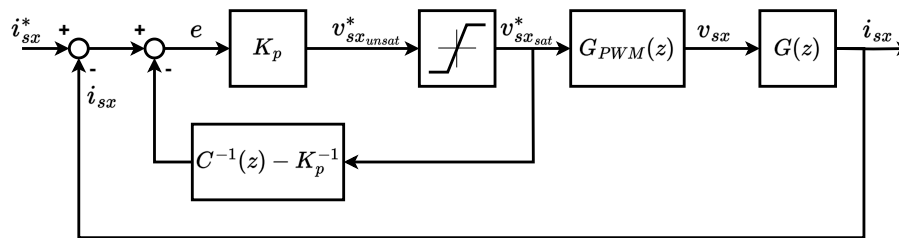


Figura 3.2. Esquema de control lineal aplicado

3.2. Control predictivo basado en modelo con conjunto de estados finitos

Los controles predictivos basados en modelos [11] se conocen desde hace más de tres décadas, resultando ser un control intuitivo y fácil de entender. Además, las restricciones y/o no linealidades del sistema pueden ser fácilmente incluidas, siendo aplicable a una variedad de sistemas [12]. Dado a que el modelo de control predictivo requiere una gran cantidad de cálculos esta metodología se congeló durante décadas. No obstante, debido a los avances tecnológicos en microcontroladores, se han ido estudiando y desarrollando como una alternativa prometedora en control de convertidores de potencia.

En términos generales, los modelos de control predictivo definen la acción de control minimizando una función de costo que compara la referencia con el comportamiento predicho del modelo del sistema. Por tanto, para el caso del control FCS-MPC, el algoritmo itera entre todos los estados de conmutación posibles, seleccionando para la acción de control, el primer vector de estados de conmutación que minimiza la función de costo. El esquema de control FCS-MPC se muestra en la figura 3.3 [11, 13].

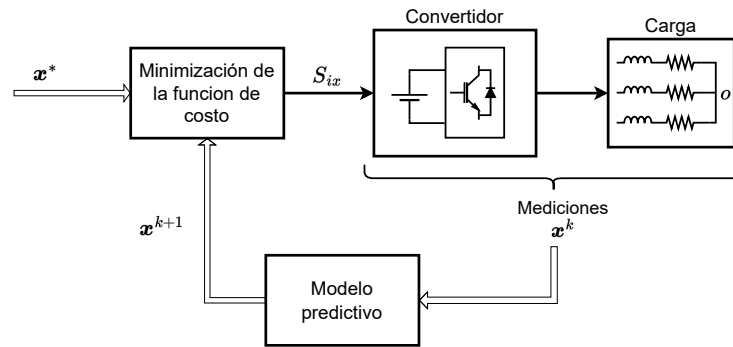


Figura 3.3. Esquema de control FCS-MPC [1]

La estrategia de control FCS-MPC toma como entrada las mediciones de las variables x en un instante k , para estimar el valor de estas en el instante $k + 1$ y con ellas predecir el valor de las variables eléctricas en el instante $k + 2$, mediante la iteración de todos los estados de conmutación posibles y eligiendo el valor que minimiza la función objetivo. En este esquema de control el estado de conmutación se mantiene constante entre $k + 1$ y $k + 2$, es decir durante un período T .

La función de objetivo generalizada [11] es la desviación de cada variable con respecto a su referencia ponderada por un factor γ_i con $i \in \{1, 2, ..n\}$ y n la cantidad de variables consideradas en el modelo.

$$g^k = \sum_{i=1}^n \gamma_i \left| x_i^* - x_i^{k+2} \right|^p \quad (3.4)$$

Donde p es la norma de la desviación, usualmente se escoge $p = 2$ debido al mejor desempeño en aplicaciones de electrónica de potencia [13].

3.2.1. Control predictivo basado en modelo aplicado a un FCC

En el caso del FCC las variables medidas son las corrientes a la salida del convertidor i_a , i_b , i_c y las tensiones de los capacitores de cada fase v_{1a} , v_{2a} , v_{1b} , v_{2b} , v_{1c} y v_{2c} . Con estas mediciones en el instante k se realiza la estimación del valor las variables en el instante $k + 1$.

Para estimar las corrientes y las tensiones en cada condensador en el instante $k + 1$ se utilizan las ecuaciones siguientes

$$v_{xN}^k = S_{3x}^k V_{dc} - (S_{3x}^k - S_{2x}^k) v_{2x}^k - (S_{2x}^k - S_{1x}^k) v_{1x}^k \quad (3.5)$$

$$i_x^{k+1} = (v_{xN}^k - v_{oN}^k) \frac{T}{L} + i_x^k \left(1 - \frac{R \cdot T}{L} \right) \quad (3.6)$$

$$v_{1x}^{k+1} = \frac{T}{2C_{1x}} (i_x^{k+1} + i_x^k) (S_{2x}^k - S_{1x}^k) + v_{1x}^k \quad (3.7)$$

$$v_{2x}^{k+1} = \frac{T}{2C_{2x}} (i_x^{k+1} + i_x^k) (S_{3x}^k - S_{2x}^k) + v_{2x}^k \quad (3.8)$$

Donde T es el tiempo de muestreo, v_{xN}^k es la tensión entre la salida del convertidor de la fase x y el neutro N en el instante k , i_x^k es la corriente medida en el instante k , v_{1x}^k y v_{2x}^k son las tensiones medidas en los capacitores en el instante k y S_{ix}^k con $i \in \{1, 2, 3\}$ y $x \in \{a, b, c\}$ el estado de conmutación seleccionado en el periodo anterior y aplicado en el instante k . La tensión de punto medio puede ser calculada mediante la suma de las tensiones de salida del convertidor $v_{oN}^k = \frac{v_{aN}^k + v_{bN}^k + v_{cN}^k}{3}$. Además, la tensión de punto medio puede aproximarse a $v_{oN}^k = \frac{V_{dc}}{2}$, lo que provoca una reducción de los casos posibles de $8^3 = 512$ a $3 \cdot 8 = 24$ [14]. Las ecuaciones (3.5)-(3.8) se obtienen mediante la discretización del modelo continuo aplicando el método de Euler hacia adelante, lo cual es necesario para considerar la naturaleza discreta del control predictivo en convertidores de potencia [15].

Los valores obtenidos de tensión y corriente en el instante $k + 1$ se utilizan como punto de partida para el paso predictivo.

Mediante las corrientes y tensiones en el instante $k + 1$ y los estados de conmutación S_{ix}^{k+1} se predicen las corrientes i_x^{k+2} y las tensiones v_{xN}^{k+1} , v_{1x}^{k+2} y v_{2x}^{k+2}

$$v_{xN}^{k+1} = S_{3x}^{k+1} V_{dc} - (S_{3x}^{k+1} - S_{2x}^{k+1}) v_{2x}^{k+1} - (S_{2x}^{k+1} - S_{1x}^{k+1}) v_{1x}^{k+1} \quad (3.9)$$

$$i_x^{k+2} = (v_{xN}^{k+1} - V_{dc}/2) \frac{T}{L} + i_x^{k+1} \left(1 - \frac{R \cdot T}{L} \right) \quad (3.10)$$

$$v_{1x}^{k+2} = \frac{T}{2C_{1x}} (i_x^{k+2} + i_x^{k+1}) (S_{2x}^{k+1} - S_{1x}^{k+1}) + v_{1x}^{k+1} \quad (3.11)$$

$$v_{2x}^{k+2} = \frac{T}{2C_{2x}} (i_x^{k+2} + i_x^{k+1}) (S_{3x}^{k+1} - S_{2x}^{k+1}) + v_{2x}^{k+1} \quad (3.12)$$

Donde v_{xN}^{k+1} es la estimación de tensión entre la salida del convertidor de la fase x y el neutro N en el instante $k + 1$, i_x^{k+2} es la corriente de la fase x predicha para el instante de tiempo $k + 2$, v_{1x}^{k+2} y v_{2x}^{k+2} son las tensiones predichas de los capacitores en la celda 1 y 2 en el instante $k + 2$

Utilizando la ecuación (3.13) como función objetivo a minimizar, se iteran todos los estados de conmutación S_{ix}^{k+1} posibles.

$$g^k = \sum_{x=a}^c \left((i_x^* - i_x^{k+2})^2 + \gamma_1 (v_{c1}^* - v_{2x}^{k+2})^2 + \gamma_2 (v_{c2}^* - v_{2x}^{k+2})^2 \right) \quad (3.13)$$

Donde i_x^* es la corriente de referencia, γ_1 y γ_2 son los factores que ponderan la jerarquía entre el control de corriente de salida y la tensión de los capacitores de la celda 1 y 2, v_{ci}^* es la tensión de referencia del capacitor en la celda $i = \{1, 2\}$ con $v_{c1}^* = \mathbf{V}_{dc}/3$ y $v_{c2}^* = 2\mathbf{V}_{dc}/3$, y v_{ix}^{k+2} es la tensión predicha del capacitor en la celda $i = \{1, 2\}$ en el instante $k + 2$.

El ajuste de los factores γ_1 y γ_2 se realiza de forma iterativa, ya que, no existe un método analítico exacto para el ajuste de estos. El ajuste iterativo se realiza equilibrando la importancia del control de voltaje de ambas celdas y el control de la corriente.

3.3. Redes neuronales artificiales

En el campo de la inteligencia artificial una de las técnicas más destacadas son las Redes Neuronales Artificiales (RNA) debido a su capacidad de aprendizaje y generalización [3], las cuales, en la literatura, parecen tener un impacto importante en electrónica de potencia y accionamientos [2].

Las redes neuronales ofrecen propiedades y capacidades útiles. Algunas de ellas son:

- No linealidad: Las RNA pueden ser lineales o no lineales. Sin embargo, la no linealidad es una propiedad fundamental, ya que es el mecanismo subyacente que permite caracterizar fenómenos no lineales.
- Relación entrada – salida: La falta de un modelo explícito para realizar la relación entre entradas y salidas, permite a la red neuronal artificial carecer de parámetros ligados a un modelo y con ello desprenderse de los problemas ligados a la estimación de parámetros [16].
- Adaptabilidad: Debido a la estructura de la red neuronal artificial, ésta puede ajustar sus pesos para adaptarse a los cambios en el entorno de operación mediante la implementación de algoritmos que permiten el entrenamiento online, reentrenando la red.
- Respuesta evidencial: En el contexto de clasificación de patrones, la red neuronal artificial se puede diseñar para proveer información no solo del patrón a seleccionar, sino también de la certeza que tiene la RNA ante dicha decisión.
- Información contextual: La estructura del flujo de información está representada por la estructura de la RNA misma y los estados de activación de la red, de este modo, el efecto de cada neurona está influenciada por la actividad global de las otras neuronas en la red y por consecuencia la contextualidad en la información es una característica natural en las redes neuronales.

3.3.1. Modelo de una neurona artificial

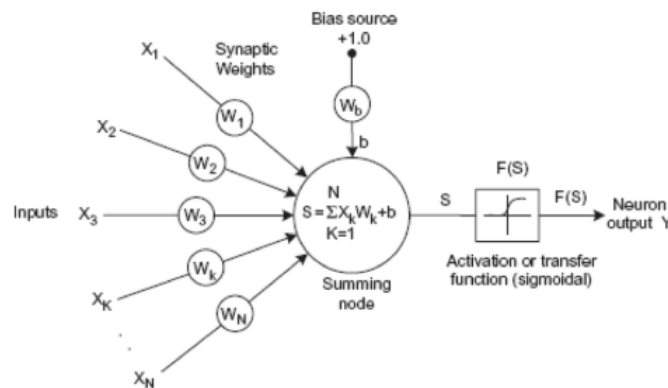


Figura 3.4. Modelo de una neurona artificial [2]

Una neurona artificial (o simplemente neurona) también llamada unidad de procesamiento, toma las señales de entrada y las multiplica por una ganancia o peso correspondiente para luego sumarlas junto con el parámetro de sesgo. El resultado de este cálculo pasa a través de una función de activación obteniéndose la salida de la neurona, como se muestra en la figura 3.4. Algebraicamente este proceso se resume de la forma siguiente

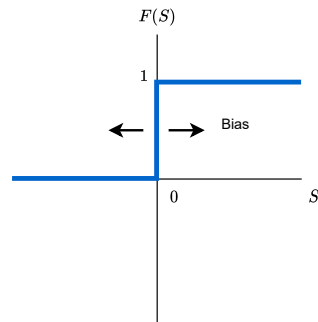
$$y = F \left(\sum_{i=1}^n x_i \cdot w_i + b \right) \quad (3.14)$$

Donde y es la salida de la neurona, x_i es el elemento i de la entrada de la neurona, n es la cantidad de elementos de la entrada, w_i es el parámetro de peso para cada x_i , b es el parámetro de sesgo de la neurona, y $F(\cdot)$ es la función de activación.

La función de activación es una función que puede ser lineal o no lineal, sin embargo, el potencial de la neurona radica en la no linealidad que aporta la función de activación. Esto evita que la red neuronal artificial sea un conjunto linealmente dependiente, posibilitando la reducción de varias neuronas en una sola, lo cual no es deseable, ya que se pierde la capacidad de que cada neurona pueda especializarse en una característica en particular. Además, proporciona la capacidad de caracterizar relaciones no lineales entre entradas y salida de la neurona.

Algunas de las funciones de activación comunes en distintas aplicaciones son:

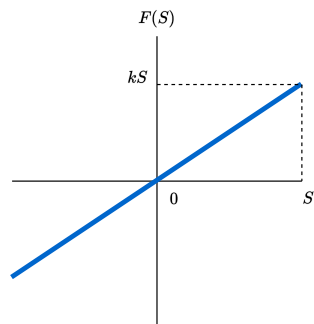
- Función de Heaviside o *Threshold*, figura 3.5, utilizada en un modelo de neurona primitiva denominada perceptrón.



$$\begin{aligned} F(S) &= 1 & \text{si } S \geq 0 \\ F(S) &= 0 & \text{si } S < 0 \end{aligned} \quad (3.15)$$

Figura 3.5. *Threshold* [2]

- Función lineal pura, figura 3.6, esta función cumple con el acondicionamiento de la salida de la neurona, lo cual puede ser una ventaja para aplicaciones de regresión, pero debido a su linealidad no permite independizar el aporte de cada neurona.

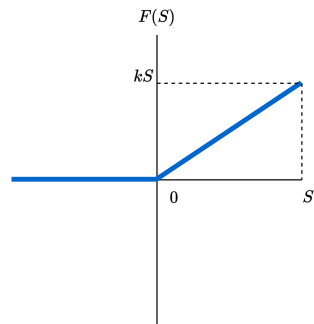


$$F(S) = k \cdot S \quad (3.16)$$

Con $k \in \mathbb{R}$

Figura 3.6. *Lineal pura*

- Función unidad lineal rectificada o ReLU, figura 3.7, es ampliamente utilizada en redes neuronales masivas dada su simplicidad y no linealidad.



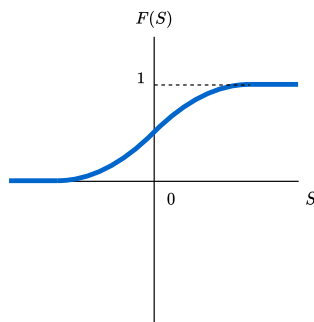
$$F(S) = k \cdot S \quad \text{si } S \geq 0$$

$$F(S) = 0 \quad \text{si } S < 0 \quad (3.17)$$

Con $k \in \mathbb{R}$

Figura 3.7. *ReLU*

- La función sigmoideal o *log-sigmoid*, figura 3.8 es una función derivable no lineal, requerida para los métodos derivativos de entrenamiento.

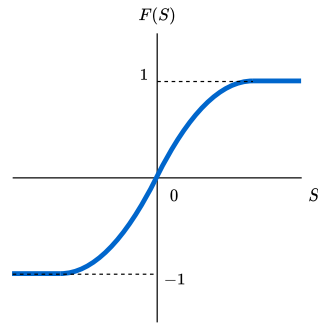


$$F(S) = \frac{1}{1 + e^{-\alpha S}} \quad (3.18)$$

Usualmente con $\alpha = 1$.

Figura 3.8. *Sigmoidea*

- Función tangente hiperbólica o *tansig*, figura 3.9 al igual que la función sigmoidea se destaca dada su derivabilidad, con la diferencia que los valores de salida varían entre $[-1,1]$.

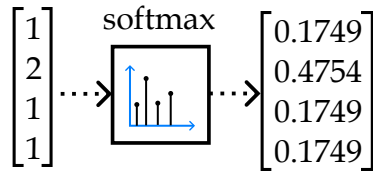


$$F(S) = \frac{e^{\alpha S} - e^{-\alpha S}}{e^{\alpha S} + e^{-\alpha S}} \Big|_{\alpha=1} = \frac{2}{1 + e^{-2S}} - 1 \quad (3.19)$$

Usualmente con $\alpha = 1$.

Figura 3.9. *Tangente hiperbólica*

- Función *softmax*, figura 3.10. A diferencia de las funciones de activación anteriores la función softmax opera considerando todos los valores del vector de entrada, devolviendo la ponderación exponencial de la suma exponencial de todos los valores, esta función se aplica generalmente en la capa de salida de las RNA de clasificación ya que permite cuantificar la clase con mayor probabilidad para un determinado vector de entrada.



$$F(S) = \frac{e^{S_i}}{\sum_{j=1}^n e^{S_j}} \quad (3.20)$$

Donde n es la cantidad de datos del vector de entrada S

Figura 3.10. *Softmax*

3.3.2. Estructura de una red neuronal artificial

Las redes neuronales artificiales se componen de arreglos de neuronas o nodos denominados capas. La capa de entrada consiste en un arreglo de variables de entrada de la RNA, donde no se implementan neuronas, pero sí nodos que siguen la estructura de la red neuronal artificial. La capa de salida contiene neuronas, y la cantidad de neuronas en esta capa corresponde a la cantidad de salidas de la red neuronal artificial. Las capas ubicadas entre la capa de entrada y la capa de salida se conocen como capas ocultas.

Es común que la interconexión entre capas se realice de manera totalmente conectada o *fully connected*, es decir que cada nodo está conectado a cada nodo de la capa siguiente, sin embargo, se pueden omitir algunas de estas conexiones, para tal caso se denominará que la RNA está parcialmente conectada o *partially connected*.

En general, se pueden identificar tres clases fundamentales de estructuras de redes neuronales artificiales [3]:

- Red neuronal *feedforward* de una capa: La red cuenta con una capa de entrada y de salida, donde *feedforward* se refiere a la dirección del flujo de información, el cual va desde la capa de entrada hacia la capa de salida.

- Red neuronal *feedforward* multi-capa: Esta red se caracteriza por la inclusión de una o más capas ocultas entre las capas de entrada y salida.
- Red neuronal recurrente: Esta red se distingue de las redes *feedforward* ya que contienen al menos un lazo de realimentación ya sea entre las capas de entrada y salida, entre dos de las capas ocultas o entre la capa de salida y alguna de las capas ocultas.

En la figura 3.11 se muestra una red neuronal *feedforward fully connected* con una capa oculta, en la capa de entrada tiene 10 nodos, cuatro neuronas en la capa oculta y dos neuronas en la capa de salida.

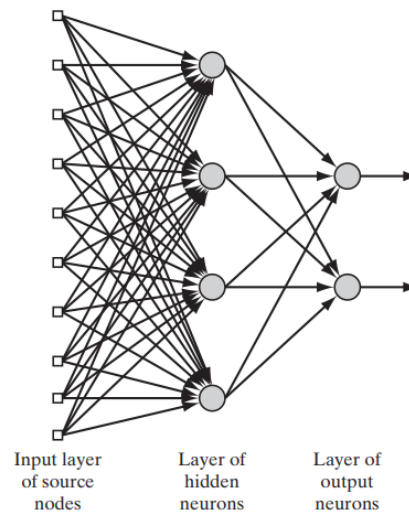


Figura 3.11. Red neuronal *feedforward fully connected* con una capa oculta [3]

3.3.3. Modelos de redes neuronales

Diferentes estructuras de redes neuronales se han propuesto a lo largo de los años, las cuales se destacan por su desempeño en ciertas aplicaciones.

3.3.3.a. Perceptrón network

Una red neuronal perceptrón es una red simple *feedforward* conformada por una única capa, la cual es usada en aplicaciones de clasificación de los patrones presentes en las señales de entrada. Esta red fue inventada por Frank Rosenblatt [3] a finales de la década de 1950. La estructura general de esta red, como se muestra en la figura 3.12, se compone de una capa cuyas neuronas tienen la función de activación *Threshold*. A este tipo de neuronas se les conoce como perceptron [17]. Una de las aplicaciones de esta red neuronal es el mapear funciones booleanas complejas, permitiendo relacionar las señales lógicas de entrada y las respuestas características de un sistema de accionamiento eléctrico.

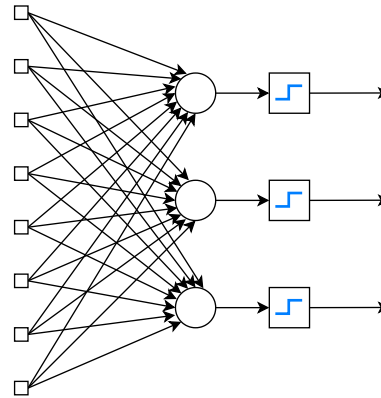


Figura 3.12. Estructura red neuronal tipo perceptrón

3.3.3.b. Backpropagation network

La red neuronal *backpropagation* es una red *feedforward* multicapa comúnmente utilizada en aplicaciones de electrónica de potencia y accionamientos [18]. El nombre *backpropagation* proviene del método de entrenamiento supervisado del mismo nombre utilizado para entrenar esta red [19]. A diferencia de la red de perceptrones, las funciones de activación empleadas en las neuronas son la función sigmoidea y la tangente hiperbólica.

El método de entrenamiento supervisado *backpropagation* consiste en el ajuste de los pesos de cada neurona mediante la ponderación del error a través de una retroalimentación inversa, es decir, desde la capa de salida los pesos de cada neurona se actualizan en proporción a su contribución al error, esta contribución disminuye a medida que se actualizan los pesos en dirección de las capas ocultas hacia la capa de entrada, el error se *propaga* en esta dirección. Ver figura 3.13.

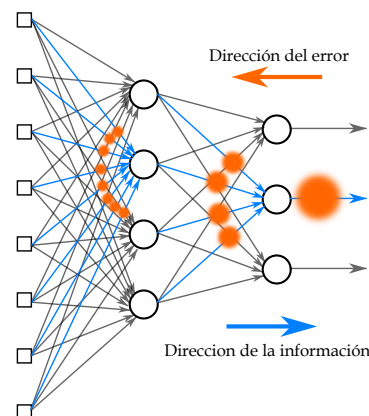


Figura 3.13. Diagrama de distribución del error, algoritmo backpropagation

3.3.4. Paradigmas de aprendizaje

En *Machine Learning* existen tres paradigmas de aprendizaje estos son: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje reforzado.

1. El aprendizaje supervisado consiste en entrenar la red proporcionando datos de entrada $\mathbf{x}$ y datos de salida $\mathbf{t}$ que se compone de etiquetas u objetivos de manera que el algoritmo aprende a generar una salida $\mathbf{y}$, que es similar a $\mathbf{t}$, dada una entrada $\mathbf{x}$.
2. El aprendizaje no supervisado consiste en entrenar la red proporcionando datos de entrada $\mathbf{x}$, definiendo solo la cantidad de características de salida, de esta manera se busca que la red pueda sintetizar estas características a partir los datos $\mathbf{x}$, tales como; patrones, secuencias y relaciones.
3. El aprendizaje reforzado consiste en una red subordinada cuya salida es evaluada por otra red la cual proporciona una señal correctiva (de recompensa o castigo) en función de mejorar el desempeño de la red subordinada.

En este trabajo se emplea el aprendizaje supervisado, proporcionando datos de entrada y salida que la red relacionará.

3.3.5. Entrenamiento de una red neuronal artificial

No existe una manera determinada de entrenar una RNA, pero existen técnicas que optimizan el desempeño de una red.

3.3.5.a. Algoritmos de entrenamiento

Un algoritmo de entrenamiento es el método por el cual una red ajusta los pesos y sesgos de manera automática. El avance en estos algoritmos es la raíz por la cual las redes neuronales han podido desarrollar tareas más complejas, como es el caso de los algoritmos basados en *backpropagation*.

Para entender el proceso de entrenamiento de una RNA, debemos definir algunos conceptos clave que serán recurrentes durante el desarrollo del trabajo. Para actualizar los pesos y sesgos, es necesario calcular un error en la salida de la RNA. Los datos de entrada y salida utilizados para calcular el error se denominan *instancias*, y un conjunto de entrenamiento generalmente se compone de miles de instancias. Al terminar la rutina de actualización de pesos y sesgos, obtendremos una versión actualizada de la RNA, la cual se espera que tenga un mejor desempeño que la anterior. A estas versiones les llamaremos *épocas*. El proceso de entrenamiento, por ende, se puede describir como la actualización de pesos y sesgos mediante la retropropagación del error de cada instancia, repetido durante un número determinado de épocas.

El algoritmo de *backpropagation* [3] ajusta los pesos y sesgos de cada neurona a partir del error de cada una de las instancias del conjunto de datos de entrenamiento, de esta manera el error de la instancia n a la salida de la neurona j , de la figura 3.14, está dado por

$$e_j(n) = d_j(n) - y_j(n) \quad (3.21)$$

Donde $d_j(n)$ es una componente del vector de salida deseado $\mathbf{d}(n)$. Se define además un *error de energía instantánea*

$$\mathcal{E}_j(n) = \frac{1}{2}e_j^2(n) \quad (3.22)$$

Sea $v_j(n)$ la sumatoria de las entradas de la neurona j

$$v_j(n) = \sum_{i=0}^m w_{ji}(n)x_i(n) \quad (3.23)$$

La salida de la neurona j es

$$y_j(n) = F_j(v_j(n)) \quad (3.24)$$

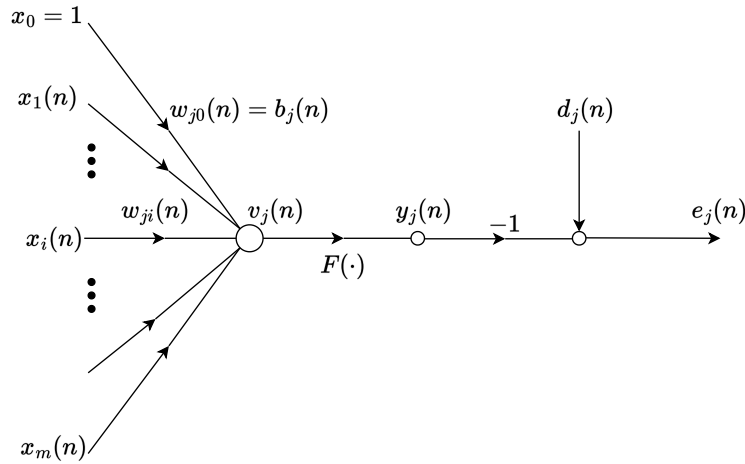


Figura 3.14. Flujo de señal de la neurona j

El algoritmo de *backpropagation* aplica una corrección $\Delta w_{ji}(n)$ a cada uno de los pesos $w_{ji}(n)$ que es proporcional a la derivada parcial $\partial \mathcal{E}(n)/\partial w_{ji}(n)$, la cual, siguiendo la regla de la cadena, la derivada parcial se puede expresar como

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \quad (3.25)$$

Derivando la ecuación (3.22) con respecto a $e_j(n)$ se tiene $\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n)$, derivando la ecuación (3.21) con respecto a $y_j(n)$ se obtiene $\frac{\partial e_j(n)}{\partial y_j(n)} = -1$, derivando la ecuación (3.24) con respecto a $v_j(n)$ se tiene $\frac{\partial y_j(n)}{\partial v_j(n)} = F'_j(v_j(n))$, finalmente derivando la ecuación (3.23) con respecto a $w_{ji}(n)$ se tiene $\frac{\partial v_j(n)}{\partial w_{ji}(n)} = x_i(n)$.

Por otra parte, el gradiente local $\delta_j(n)$ se define

$$\delta_j(n) = \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \quad (3.26)$$

Reemplazando las derivadas parciales previamente calculadas en (3.26)

$$\delta_j(n) = \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = -e_j(n)F'_j(v_j(n)) \quad (3.27)$$

Por último el factor de corrección se define

$$\Delta w_{ji}(n) = \eta \delta_j(n) x_i(n) \quad (3.28)$$

Donde η es el ratio de entrenamiento o *learning-rate*. De esta manera el algoritmo puede corregir los parámetros de cada neurona de manera proporcional al error que generan, ya sea de manera directa con respecto la respuesta deseada para las neuronas de la capa de salida o mediante el flujo del error distribuido para el caso de las neuronas de las capas ocultas.

Si la neurona j es un nodo de salida, la ecuación (3.28) se desarrolla

$$\Delta w_{ji}(n) = \eta \delta_j(n) x_i(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = -\eta (-e_j(n) F'_j(v_j(n)) x_i(n)) \quad (3.29)$$

Por otra parte, si la neurona j está en la capa oculta, el gradiente local es

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} F'_j(v_j(n)) = -\sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} F'_j(v_j(n)) \quad (3.30)$$

Donde $\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}$ y la neurona k es un nodo de salida, luego por regla de la cadena $\frac{\partial e_k(n)}{\partial y_j(n)} = \frac{\partial e_k(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial y_j(n)}$, $\frac{\partial e_k(n)}{\partial v_j(n)} = -F'_k(v_k(n))$ y $\frac{\partial v_j(n)}{\partial y_j(n)} = w_{kj}(n)$. De este modo

$$\begin{aligned} \frac{\partial \mathcal{E}(n)}{\partial y_j(n)} &= -\sum_k e_k F'_k(v_k(n)) w_{kj}(n) \\ &= -\sum_k \delta_k(n) w_{kj}(n) \end{aligned} \quad (3.31)$$

Se obtiene entonces la fórmula de *back-propagation* para el gradiente local $\delta_j(n)$, descrita por

$$\delta_j(n) = F'_k(v_k(n)) \sum_k \delta_k(n) w_{kj}(n) \quad (3.32)$$

Resumiendo, podemos calcular el factor de corrección $\Delta w_{ji}(n)$ de (3.28), usando $\delta_j(n)$ de (3.27) si la neurona j es de salida. Por otra parte, podemos usar $\delta_j(n)$ de (3.32) si la neurona j es una neurona de la capa oculta.

Para clarificar las ecuaciones anteriores, en el código MATLAB siguiente se realiza el entrenamiento de una RNA compuesta por dos neuronas en la capa oculta y una neurona en la capa de salida.

```

%% Training Set
X1 = 0:0.0001:10*pi;
X2 = 0.5*X1;
X(1,:) = X1; % input data x1
X(2,:) = X2; % input data x2
D = 0.5*( sin(X1)+(sin(X2)).^3 ); %target output data
%*****%
%% Initialization
in = 2; % dimension de la entrada
hidden = 2; % dimension de la capa oculta
out = 1; % dimension de la capa de salida
fac1 = 0.01;
fac2 = 0.01;
W1 = 2*fac1*rand(hidden, in) - fac1; %inicializacion de los pesos W1
W2 = 2*fac2*rand(out, hidden) - fac2; %inicializacion de los pesos W1
num = size(X,2);
lr = 0.5; %learning rate
%*****%
%% Training
for epoch = 1:num
    x = X(:,epoch);
    d = D(:,epoch);
    step(epoch) = epoch;

    w1x = W1*x;
    y1 = tansig(w1x); %salida de la capa oculta
    w2d1 = W2*y1;
    y2 = tansig(w2d1); %salida de la RNA

    e2 = d - y2; %error entre la salida esperada d y la salida de la RNA y2
    delta2 = -e2*(1 - y2.^2);
    deltaW2 = -lr*delta2*(y1)';

    delta1 = delta2*W2*(1 - y1.^2);
    deltaW1 = -lr*delta1*(x)';

    W2 = W2 + deltaW2;
    W1 = W1 + deltaW1;
end

```

Para una red neuronal artificial de mayor dimensión el código presentado puede ser tedioso, es por ello que existen funciones que permiten realizar algoritmos de entrenamiento de forma generalizada. La función *train* de MATLAB permite entrenar una RNA creada con las herramientas del paquete *Deep learning toolbox*, la RNA contiene la estructura, dimensión y algoritmo de entrenamiento como parte de sus propiedades. La función *train* recibe como argumento la RNA, los vectores de entrada y los vectores de salida.

```

%Training Function
trainFcn = 'traingda';
% Create a Pattern Recognition Network
hiddenLayerSize = 6;
net = fitnet(hiddenLayerSize, trainFcn);
% Train Params
net.trainParam.epochs=1000;
net.trainParam.goal=0.01;
net.trainParam.min_grad = 1e-4;
% Train the Network
[net,tr] = train(net,x,t);

```

Como hemos visto, el algoritmo de entrenamiento *backpropagation* nos permite entrenar la RNA actualizando los pesos y sesgos en base a la retropropagación del error de cada instancia. Si bien el desarrollo de estas ecuaciones para el entrenamiento de una RNA personalizada puede resultar extenuante, en la actualidad hay herramientas que nos facilitan la aplicación de estos algoritmos de entrenamiento en RNA personalizadas. Estas herramientas permiten la utilización de algoritmos más sofisticados, como es el caso del algoritmo *Gradient descent with adaptive learning rate backpropagation*, el cual utiliza los gradientes provenientes del algoritmo *backpropagation* para direccionar la corrección de los pesos con el descenso del gradiente y con ello minimizar el error, ajustando además el *learning rate* para aumentar o disminuir la magnitud de la corrección. En MATLAB la función de entrenamiento *Gradient descent with adaptive learning rate backpropagation* se llama **traingda**.

En el código de ejemplo la RNA denominada **net** se define mediante la función **fitnet**, que es una estructura prediseñada de RNA *single layer feed foward* con una función de activación en la capa oculta de tangente hiperbólica y una función de activación lineal pura en la capa de salida, que recibe como argumento la cantidad de neuronas en la capa oculta y la función de entrenamiento.

Los parámetros de entrenamiento, conforma parte de las propiedades de la RNA, siendo **trainParam.epochs** la cantidad de épocas de entrenamiento, **trainParam.goal** el error objetivo del entrenamiento y **trainParam.min_grad** el gradiente de descenso mínimo del error que se desea establecer. El cumplimiento de cualquiera de estas condiciones detendrá el entrenamiento.

3.3.5.b. Dimensión de una red neuronal

Es intuitivo pensar que una red con mayor número de neuronas en la capa oculta o con un mayor número de capas [20], [21], se tendrá un mejor desempeño en cuanto a la regresión o clasificación que se requiera, sin embargo, esta afirmación no es siempre correcta. Además, una red con una gran cantidad de neuronas requerirá mayor capacidad de procesamiento en el cálculo online y en entrenamiento, eliminando una de las ventajas en comparación con los métodos tradicionales de control.

Por ende, es recomendable encontrar la cantidad de neuronas en la capa oculta que optimice la relación entre el desempeño y el tamaño de la red neuronal artificial.

Con el objetivo de ejemplificar lo mencionado anteriormente se realizará una exploración de la relación entre la dimensión de una red neuronal artificial y el error *Mean square error* (MSE) entre la salida de la RNA y la salida esperada. La red neuronal artificial de

prueba es una versión previa de la RNA aplicada al caso del control PI presentado en este trabajo de memoria.

La red tiene seis nodos de entrada correspondiente a la corriente de cada fase y el error con respecto a la corriente de referencia, la capa de salida tiene tres nodos correspondientes a los índices de modulación de cada fase.

Para observar la influencia de la cantidad de datos del conjunto de entrenamiento en el valor del error, se realiza la exploración con dos conjuntos de datos. El conjunto de entrenamiento uno (RNA 1) contiene cinco veces más datos que el conjunto dos (RNA 2).

En la figura 3.15 se muestra el comportamiento del desempeño medido en el error cuadrado medio en función de la cantidad de neuronas de la capa oculta, para la misma estructura de red neuronal entrenada con dos conjuntos de entrenamiento diferentes.

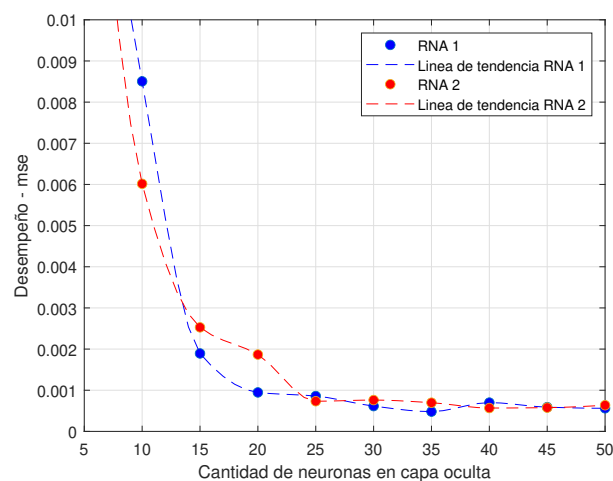


Figura 3.15. Desempeño de la red en función de la cantidad de nodos en la capa oculta

De la figura 3.15 se observa que el error MSE no siempre disminuye al aumentar la cantidad de neuronas en la capa oculta, además se observa que para la tercera cifra significativa del error MSE, se mantiene constante para una cierta cantidad de neuronas, 20 neuronas para el caso uno y 25 neuronas para el caso dos.

Del análisis exploratorio anterior podemos desprender que la cantidad de neuronas en la capa oculta no es proporcional con la disminución del error, por ende, es recomendable explorar y definir la cantidad de neuronas de la capa oculta que optimice la relación entre menor error y tamaño de la RNA.

Es necesario destacar que el error no es el único parámetro para evaluar el desempeño de una RNA, dado que la generalización y caracterización de un conjunto de entrenamiento por parte de la RNA superan lo cuantificable por un indicador de error como el MSE. El ajuste ideal de pesos y sesgos que minimiza el error puede significar un correcto desempeño de la RNA, o puede resultar en una combinación de pesos y sesgos que minimizan el error pero no refleja el comportamiento deseado. La matriz de confusión, el histograma y el índice de regresión, entre otros, son herramientas útiles para evaluar el comportamiento deseado de la RNA antes de su implementación.

3.3.5.c. Vectores de entrada y salida

Si nuestro objetivo es que la red neuronal artificial pueda relacionar correctamente los datos de entrada con los datos de salida, debemos procurar que estamos efectivamente proporcionando la información suficiente para que la RNA pueda realizar esta tarea.

Si entregamos a la red un vector de entrada que contiene información relevante e irrelevante a la RNA, durante el proceso de entrenamiento, la RNA aislará las entradas irrelevantes solo si hay entradas claramente relevantes, en caso contrario, el algoritmo de entrenamiento intentará relacionar esta información sin éxito, resultando en un desempeño deficiente. En problemas de clasificación cuando la RNA clasifica erróneamente o la probabilidad de acierto es marginal con respecto a varias clases, se denomina *confusión*.

Para simplificar el entrenamiento de red, se puede preprocesar la información, restando complejidad a las relaciones que tendrá que encontrar el algoritmo de entrenamiento.

Por ejemplo, el control PI realiza una acción proporcional al error entre el valor de referencia y la medición. Si proporcionamos a la red la medición y el valor de referencia, durante el entrenamiento la RNA intentará relacionar la medición, el valor de referencia y una combinación no lineal de ambos con la acción proporcional del control, lo cual resulta innecesariamente complejo para una simple sustracción.

$$\begin{array}{lcl}
 \text{Vector de entrada } \mathbf{x} = \{i_a^*, i_a\} & & \text{Vector de entrada } \mathbf{x} = \{i_a^* - i_a\} \\
 \left. \begin{array}{l} F(w_{11} \cdot i_a + b) \approx k_p(i_a^* - i_a) \\ F(w_{22} \cdot i_a^* + b) \approx k_p(i_a^* - i_a) \\ F(w_{11} * i_a + w_{12} * i_a^* + b) \approx k_p(i_a^* - i_a) \\ F(w_{21} * i_a + w_{22} * i_a^* + b) \approx k_p(i_a^* - i_a) \end{array} \right\} & & F(w_1 \cdot (i_a^* - i_a) + b) \approx k_p(i_a^* - i_a)
 \end{array}$$

Por ello es recomendable preprocesar la información, realizando estos acondicionamientos simples y proporcionar a la RNA vectores de entrada simples de relacionar con los vectores de salida.

El análisis de las características de los datos y la manera de preprocesar la información entra en el campo del *feature engineering* y depende de las características particulares de cada problema, en el caso de control de convertidores se puede hacer uso del conocimiento de la manera en que realizan el control las distintas estrategias, para elegir el preprocesado de las entradas.

Luego de elegir cuáles serán las entradas de la RNA, es necesario normalizar la entrada, esto es definir un rango de operación para cada componente del vector de entrada. Con rangos de entrada definidos podremos comparar el desempeño de redes de igual estructura, pero entrenadas con datos diferentes.

La elección de los vectores de salida también requiere consideraciones especiales. Un vector de salida con mayor complejidad presentará un mayor desafío al entrenar la red. Por ejemplo, para un control lineal tradicional con PWM, un vector de salida con estados de conmutación será más complejo que un vector de salida con los índices de modulación, dado que el vector de estados de conmutación añade información de la señal portadora triangular y del comparador. Además, el uso de herramientas computacionales para construir redes neuronales artificiales prediseñadas condiciona una estructura predeterminada de salidas. Esto ocurre, por ejemplo, en el caso de redes prediseñadas de clasificación, las cuales están

diseñadas para clasificar una clase a la vez y, por ende, requieren una reparametrización de la salida para clasificar varios estados de conducción simultáneos.

Considerando la topología del FCC por fase visto en la sección 2.1 cuyos estados de conducción son los descritos en la tabla 3.1, si se desea usar una RNA que clasifique los estados de conducción activos para una cierta entrada y se define el vector de salida como $\mathbf{y} = \{S_{1x}, S_{2x}, S_{3x}\}$, existirán casos donde se activen más de una clase a la vez, provocando problemas de confusión en la RNA. Para evitar esta situación y que solo se active una clase a la vez, podemos parametrizar la salida $\mathbf{y}$ donde cada combinación es una clase por si sola, de la forma $\mathbf{y}' = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$ con C_1 corresponde a la combinación 1. Si la combinación activa es la combinación 6 entonces $C_6 = 1$ y el vector de salida para este caso es $\mathbf{y}' = \{0, 0, 0, 0, 0, 1, 0, 0\}$

Tabla 3.1
COMBINACIONES POSIBLES DE ESTADOS DE CONDUCCIÓN

S_{1x}	S_{2x}	S_{3x}	
0	0	0	Combinación 1
0	0	1	Combinación 2
0	1	0	Combinación 3
0	1	1	Combinación 4
1	0	0	Combinación 5
1	0	1	Combinación 6
1	1	0	Combinación 7
1	1	1	Combinación 8

El *feature engineering* es un campo que no cuenta con métodos exactos que se puedan estudiar y aplicar, pero si da ciertas directrices que se deben tener en consideración al momento de crear una RNA. El desempeño de una RNA depende en gran medida de la representación elegida de los datos y de qué forma es aplicada. Por esa razón, mucho del esfuerzo real en la implementación de algoritmos de aprendizaje automático se destina al diseño de vías de preprocesamiento y transformaciones de datos que dan como resultado una representación de los datos que puede respaldar un aprendizaje automático eficaz. En la actualidad se buscan algoritmos de aprendizaje que no dependan del *feature engineering*, pero estas alternativas se orientan a RNA más complejas [22] [23].

Capítulo 4

Control en base a redes neuronales artificiales aplicado

En este trabajo se realizará el control en base a redes neuronales considerando dos estrategias de control por separado, la primera estrategia es el control lineal PI con modulación PS-PWM y el segundo es el control predictivo FCS-MPC, por ende, se tendrá una red neuronal artificial para cada estrategia.

4.1. Control en base a RNA para estrategia de control lineal

La estructura de la red neuronal artificial para la estrategia de control lineal es una red de tipo *backpropagation network* la cual estará compuesta de una capa oculta y una capa de salida.

De la subsección 3.1.2, se observa que el espectro de información del controlador PI comprende la corriente medida i_{sx} , la corriente de referencia i_{sx}^* y el error entre ambas. Por medio de la prueba y error se añade al vector de entrada la información del índice de modulación del paso de tiempo anterior. Caracterizar el controlador PI utilizando las corrientes en coordenadas dq y la salida del bloque de control PI con una RNA, no añade una ventaja al utilizar una RNA por lo que se decide realizar el control en base a RNA directamente en coordenadas abc , añadiendo complejidad a la RNA.

En el control en base a RNA para la estrategia de control lineal y al igual que el control PI, el balance de tensión de los condensadores flotantes es realizado por la modulación PS-PWM. El balance de tensión es sensible a cambios abruptos de índice de modulación. Sin embargo, nos permite obtener una estrategia de control tradicional relativamente simple para comparar con la estrategia en base a redes neuronales.

La información necesaria para el control PI aplicado a un convertidor tipo FC, es la corriente de referencia y la medida de cada fase y el índice de modulación del paso anterior, por ello se define como entrada las corrientes de referencia y medidas de cada fase, agregando además el error simple de corriente de cada fase y el índice de modulación del paso de tiempo anterior, del siendo el vector de entrada $\mathbf{x} = \{i_a^*, i_a, i_a^* - i_a, i_b^*, i_b, i_b^* - i_b, i_c^*, i_c, i_c^* - i_c, m_a^{k-1}, m_b^{k-1}, m_c^{k-1}\}$.

El vector de salida se define como el índice de modulación de corriente de cada fase

$\mathbf{y} = \{m_a, m_b, m_c\}$. La red neuronal artificial requiere encontrar una relación entre las corrientes y el respectivo índice de modulación, por ende, la red neuronal artificial se desempeñará como una RNA de regresión.

En la sección 3.3.5.b se menciona que existe un coeficiente ideal entre la cantidad de neuronas y el error de entrenamiento de la red, este coeficiente se encuentra mediante la iteración de entrenamientos. Dado el gran volumen de datos, el análisis exploratorio se realiza con 10, 15 y 20 neuronas encontrándose que la mejor dimensión es 15. Posteriormente, tras realizar iteraciones de prueba y error con diferentes entradas, se encontró que, para una dimensión de 12 neuronas, el error no aumenta significativamente en comparación con la RNA de 15 neuronas. Por lo tanto, se determina que la capa oculta estará compuesta por 12 neuronas.

La estructura de la RNA se compone de una capa oculta con 12 neuronas y una función de activación tangente hiperbólica, la capa de salida tiene tres neuronas y una función de activación lineal, como se muestra en la figura 4.1.

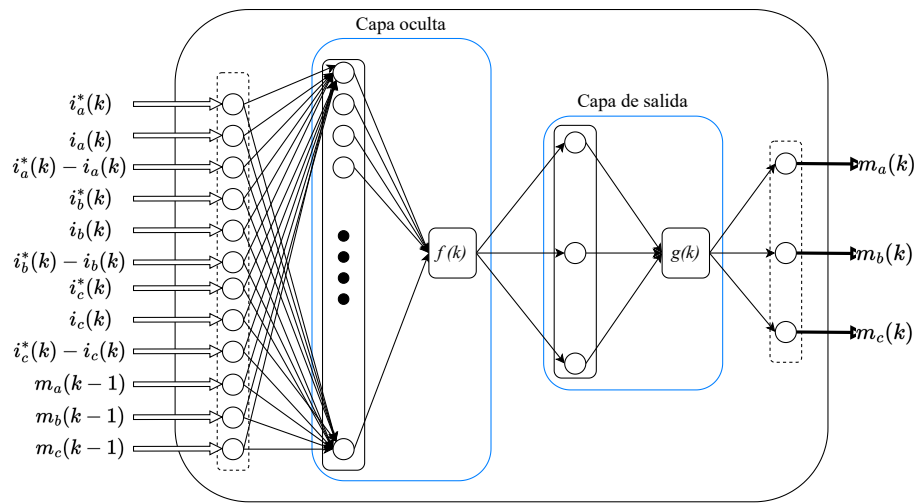


Figura 4.1. Diagrama de RNA aplicada a la estrategia de control lineal

Dado que las salidas son índices de modulación, las salidas no requieren un procesamiento adicional y son directamente enviadas a un modulador PS-PWM.

El esquema de control en base a redes neuronales aplicada a la estrategia de control lineal se muestra en la figura 4.2

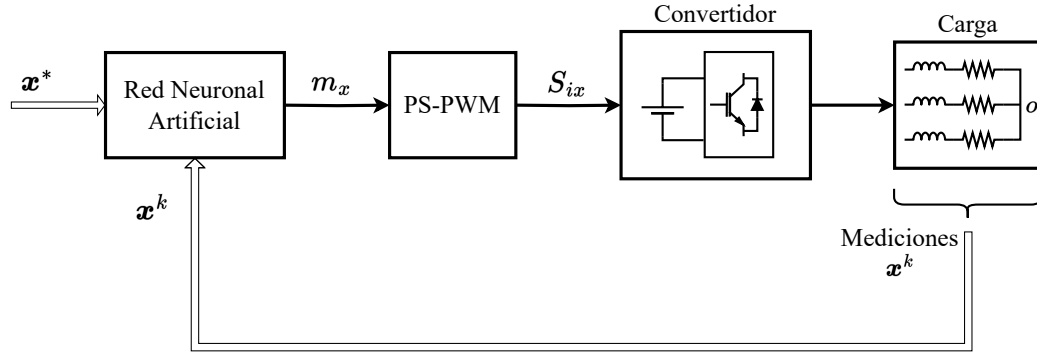


Figura 4.2. Esquema de control en base a RNA aplicada a la estrategia de control lineal

4.1.1. Datos de entrenamiento para el caso de control lineal

En la figura 4.3 se muestran los datos utilizados para el entrenamiento de la red neuronal artificial, la cual consiste en las corrientes medidas i_a , i_b y i_c y las corrientes de referencia i_a^* , i_b^* y i_c^* y los índices de modulación del instante $k - 1$, m_a^{k-1} , m_b^{k-1} y m_c^{k-1} de la operación de un FCC con control PI y modulación PS-PWM, alimentando una carga RL con un inductor de $10[mH]$ y distintos casos de valor de resistencia $R = 15[\Omega]$, $R = 16[\Omega]$ y $R = 18[\Omega]$. La frecuencia de muestreo del controlador PI es de $10[kHz]$ y la frecuencia del modulador PWM es de $1.666[kHz]$.

Para obtener los datos se da como referencia una corriente $i_q^* = 0$ y se varia la magnitud de la corriente i_d^* en $\{0,1,2,3,4,5,6,7,8,9,10\}$ y $\{-1,-2,-3,-4,-5,-6,-7,-8,-9,-10\}$ permaneciendo tres periodos en cada magnitud, luego se repite el procedimiento con la referencia $i_d^* = 0$ y se varia la magnitud de corriente i_q^* . Lo anterior se repite para una carga con $R = 15[\Omega]$, $R = 16[\Omega]$ y $R = 18[\Omega]$.

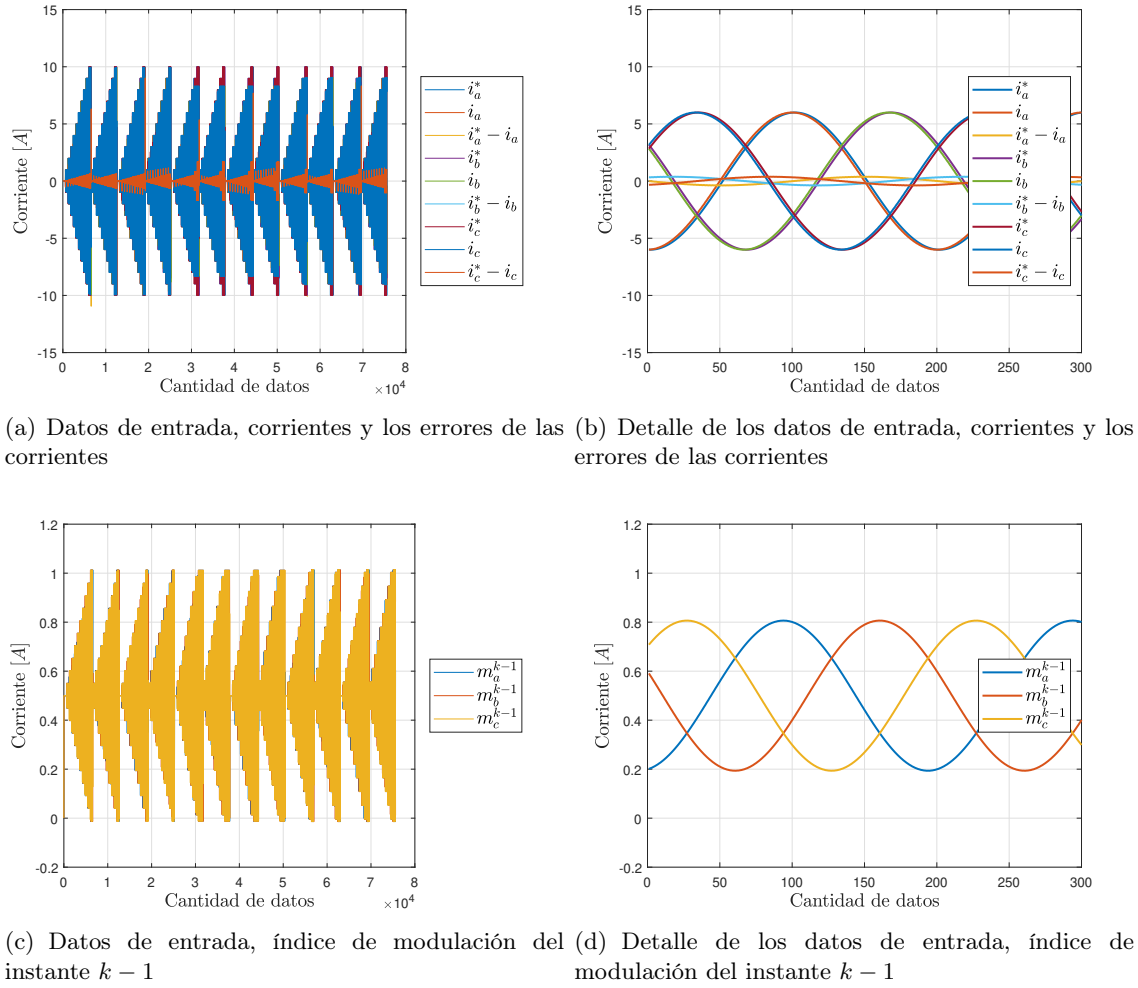


Figura 4.3. Conjunto de datos de entrada para el entrenamiento de la RNA para caso control PI

Los datos de salida corresponden a los índices de modulación m_a , m_b y m_c , presentados en la figura 4.4.

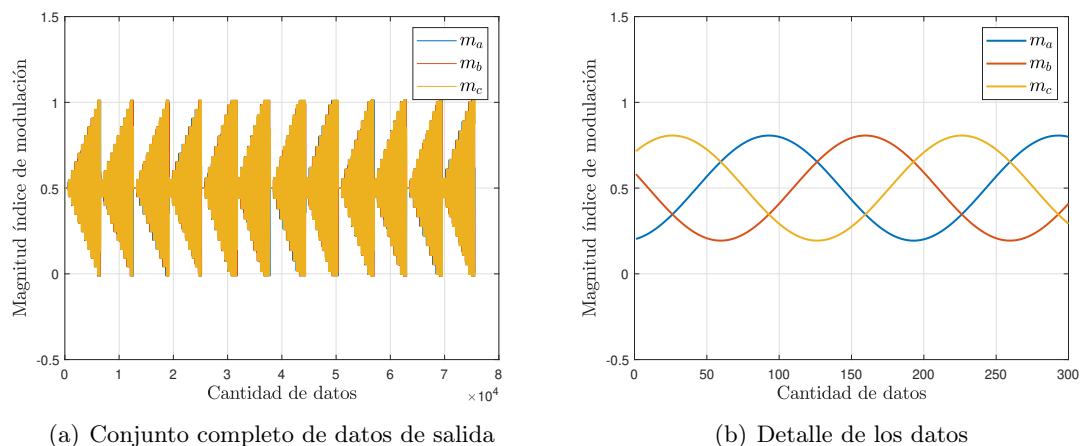


Figura 4.4. Conjunto de datos de salida para el entrenamiento de la RNA para caso control PI

El conjunto de datos de entrenamiento se compone de 75.600 vectores individuales de entrada y salida. Para el entrenamiento, se selecciona aleatoriamente el 95 % de los datos, dejando el 5 % restante para las pruebas. Dado que la RNA que se desea implementar tiene como objetivo imitar el control PI operando dentro de los rangos proporcionados en el conjunto de entrenamiento, el sesgo de aprendizaje de la RNA por este conjunto no representa necesariamente una característica negativa.

4.1.2. Creación y entrenamiento de la RNA para el caso de control lineal

De la sección 4.1, se obtiene la estructura de la RNA, la dimensión y componentes del vector de entrada y de salida, además de la dimensión de la capa oculta.

La definición de los rangos de entrada y salida, según lo mencionado en la sección 3.3.5.c, se determina de acuerdo con el rango de operación de las distintas variables. En particular, para las corrientes medidas y de referencia, estas operan en rangos de ± 10 [A], por lo que se define un rango de -15 [A] a 15 [A]. El error de corriente oscila mayoritariamente en $\pm 0,5$ [A], y en los cambios de referencia alcanza ± 1 [A]. Para no influenciar el entrenamiento ante cambios de referencia mayores, se define un rango de -2 [A] a 2 [A]. Por último, el rango del índice de modulación del paso anterior se define entre 0 y 1.

Los valores iniciales de los pesos y sesgos de las distintas capas de la RNA se definen como valores aleatorios entre $[-1,1]$ para que todas las neuronas tengan un punto de partida diferente en el entrenamiento.

En la sección 3.3.5.a se revisó el algoritmo de entrenamiento *backpropagation* el cual es el algoritmo base de las funciones de entrenamiento que se utilizan en la actualidad. En este trabajo de memoria se utilizará el algoritmo *Gradient descent with adaptive learning rate backpropagation* correspondiente a la función `traingda` de MATLAB para entrenar la red neuronal artificial.

Finalmente, se definen como parámetros de entrenamiento la cantidad de épocas y el error objetivo. La cantidad de épocas del parámetro `trainParam.epochs` define la cantidad máxima de evoluciones de la RNA, una vez alcanzado este límite, el entrenamiento se

detendrá. El parámetro `trainParam.goal` establece un error objetivo, si durante el entrenamiento se alcanza un error menor a este objetivo, se detendrá el entrenamiento. Estos límites son impuestos durante el proceso de entrenamiento, sin embargo, en MATLAB es posible finalizar el entrenamiento manualmente si así se desea.

Utilizando el paquete de herramientas de MATLAB, *Deep Learning Toolbox*, se genera una red neuronal artificial de base mediante la función *fitnet*, definiendo 12 neuronas en la capa oculta. Se ajustan los parámetros de inicialización de la red, se definen los rangos de entrada y salida, y se elige el algoritmo de entrenamiento junto con sus parámetros asociados. Estos detalles se presentan en el siguiente resumen de código:

```
trainFcn = 'traingda'; %Funcion de entrenamiento

% Creacion una red Fitting Neural Network
hiddenLayerSize = 12;
net = fitnet(hiddenLayerSize, trainFcn);

% Inicializacion de la red neuronal artificial
net.layers{:}.initFcn = 'initwb';
net.inputWeights{1}.initFcn = 'rands';
net.layerWeights{2}.initFcn = 'rands';
net.biases{:}.initFcn = 'rands';
net = init(net);

% Rangos de entrada y salida
net.inputs.range = {[-15, 15; -15, 15; -2,2;-15,15; -15,15; -2,2;-15,15;
-15,15; -2,2; 0,1; 0,1; 0,1]};
net.outputs.range = {[], [0,1; 0,1; 0,1]};

% Definir division de los datos para Entrenamiento, Validacion y Prueba
net.divideParam.trainRatio = 95/100;
net.divideParam.valRatio = 0/100;
net.divideParam.testRatio = 5/100;

% Parametros de entrenamiento
net.trainParam.epochs=200000;
net.trainParam.goal=0.00001;
```

La red neuronal artificial *fitnet* es una red que tiene una capa oculta con una función de activación tangencial y una capa de salida con una función de activación lineal, lo cual coincide con la estructura de la red definida en la sección 4.1.

Para entrenar la red se hace uso de la función *train* de MATLAB. En la tabla 4.1 se resumen los resultados del entrenamiento.

Tabla 4.1
RESUMEN DE ENTRENAMIENTO DE RNA, PARA EL CASO DE CONTROL LINEAL

Parámetro	Valor
Función de entrenamiento	<i>Gradient descent with adaptive learning rate backpropagation</i>
Número de épocas	150018
Tiempo	83 [min]
Función de desempeño	Mean Squared Error
Mejor error de desempeño	3.307e-06

El proceso de entrenamiento se interrumpe en la época 150018 dado que la variación del descenso del error es mínima, evitando así el sobre ajuste de la RNA con los datos de entrenamiento.

En la figura 4.5 se muestran los gráficos representativos del entrenamiento de la red neuronal artificial.

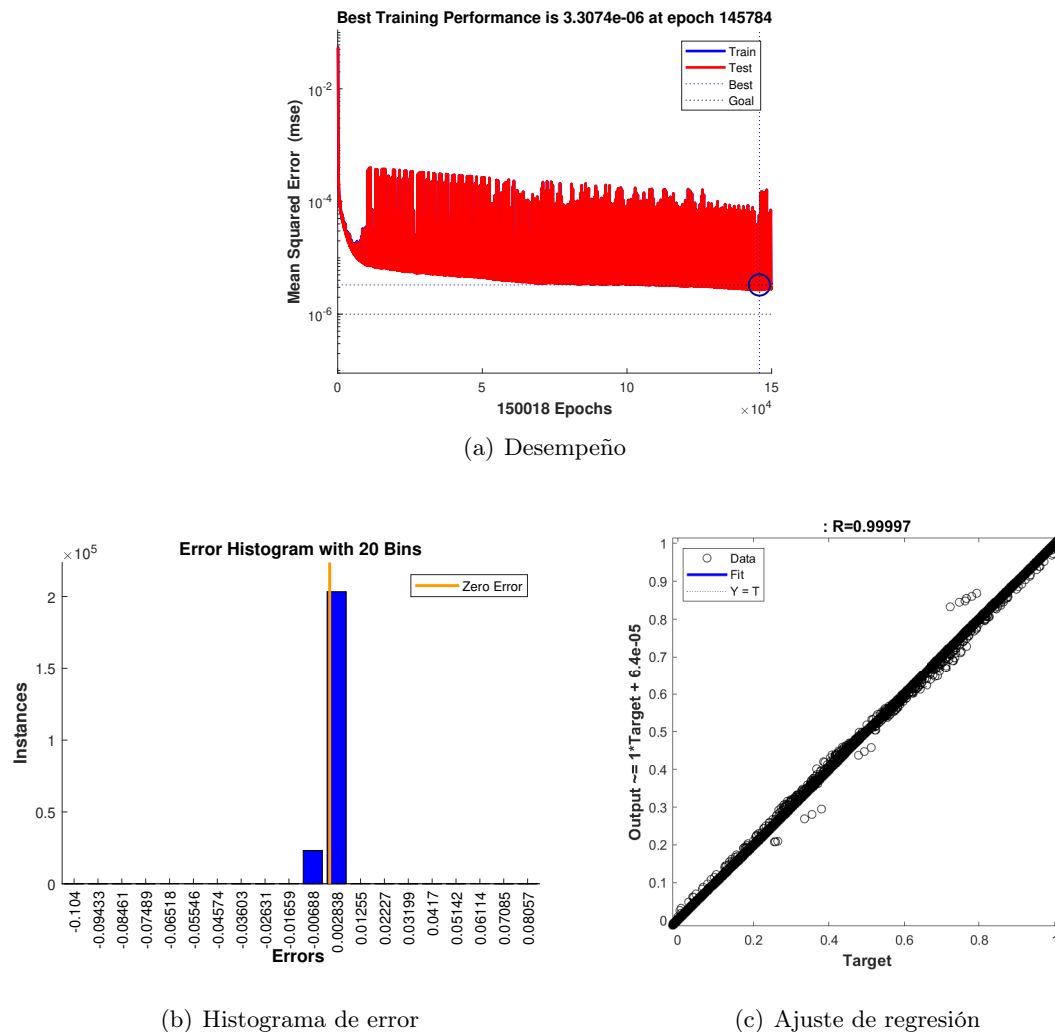


Figura 4.5. Gráficos representativos del entrenamiento y desempeño, para el caso de control lineal

La figura 4.5(a) muestra el progreso de error MSE en cada versión de la RNA o época, durante el proceso de entrenamiento se busca conseguir el menor error MSE. La figura 4.5(b) muestra el histograma de error de la última época, se busca que todos los errores de cada dato o instancia tiendan a cero, tanto en distribución como en magnitud. En la figura 4.5(c) se muestra el ajuste de regresión de la salida de la RNA y , con los datos de salida de entrenamiento t , se busca que todos los datos estén confinados en la diagonal $y = t$.

De la figura 4.5 se observa que durante el entrenamiento se alcanza un error MSE mínimo en la época número 121531 y que la regresión es adecuada, demostrada por un índice de correlación para todos los datos de $R = 0,99997$.

4.1.3. Implementación de la RNA en entorno de simulación para el caso de control lineal

Mediante el entorno de simulación PLECS se implementa el control en base a redes neuronales, donde los pesos y sesgos de la RNA ya entrenada se almacenan en un archivo de extensión .h como matrices, para luego ser llamadas en el código del control.

La frecuencia de muestreo del control se realiza a 10 [kHz] con un modulador PS-PWM de tres triangulares, cada una de ellas a 1.666[kHz], el convertidor FC alimenta una carga RL con resistencia de $R = 15[\Omega]$ e inductancia de $L = 10[mH]$, siendo así consecuente con las condiciones de los datos de entrenamiento.

En la figura 4.6 se muestra la forma de onda de corriente de la fase a para el FCC controlado con la red neuronal artificial aplicada a la estrategia de control lineal, las corrientes medidas y de referencia para cada una de las fases, las corrientes en coordenadas dq y las tensiones en los condensadores del FCC.

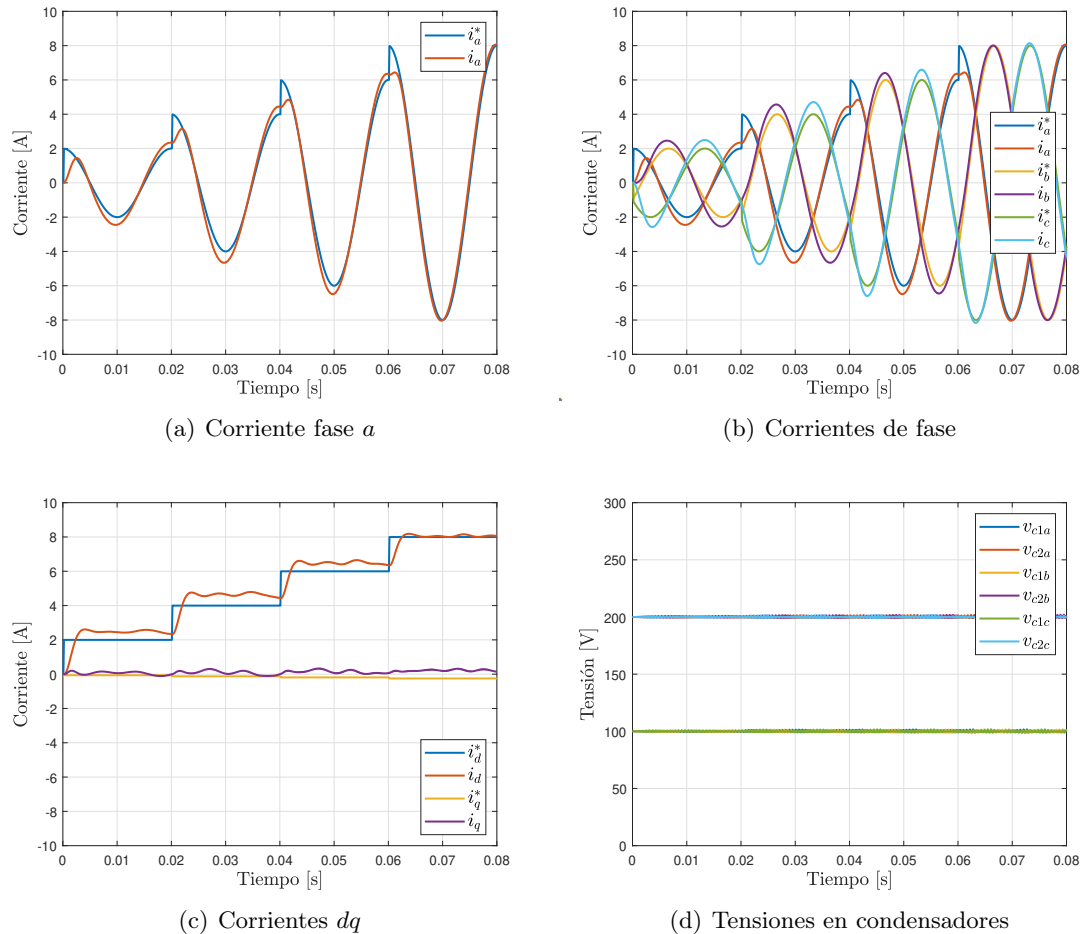


Figura 4.6. Corrientes y tensiones medidas para control basado en RNA, caso control lineal

En la figura 4.6 Se observa un comportamiento proporcional entre el cambio en la referencia de corriente y las corrientes controladas por la red neuronal artificial en todas

las fases y se puede observar que las tensiones en los condensadores se mantienen debido a la modulación PS-PWM. Sin embargo, de las figuras 4.6(a), 4.6(b) y 4.6(c) se observa que existe un error en estado estacionario de magnitud 0.53 [A] en promedio con respecto a la referencia. El criterio que se utilizará para evaluar la calidad del control es el THD en estado estacionario y para evaluar el comportamiento dinámico se medirán los tiempos de asentamiento de la componente de corriente en el eje d y se compararán con el control lineal tradicional.

Tabla 4.2

COMPARACIÓN DE THD ENTRE EL CONTROL PI Y EL CONTROL DE LA RNA EN FUNCIÓN DE LA REFERENCIA DE AMPLITUD DE CORRIENTE

$A_{ref}[A]$	2	4	6	8	10
THD PI [%]	2,7	2,5	2,45	1,8	1,2
THD RNA [%]	3.0	3.1	2,6	1,9	1,7

En la tabla 4.2 se observa que el THD disminuye al aumentar el valor de la referencia de corriente, esto es debido a que los escalones de tensión entre los que se conmuta producen un *ripple* de corriente que es relativamente menor a medida que la magnitud de corriente aumenta, por lo cual este comportamiento del THD se ve reflejado en ambos casos. Además, se observa que el valor de THD para los casos considerados son muy similares para ambos métodos de control siendo el caso de la red neuronal artificial el método con un THD marginalmente mayor.

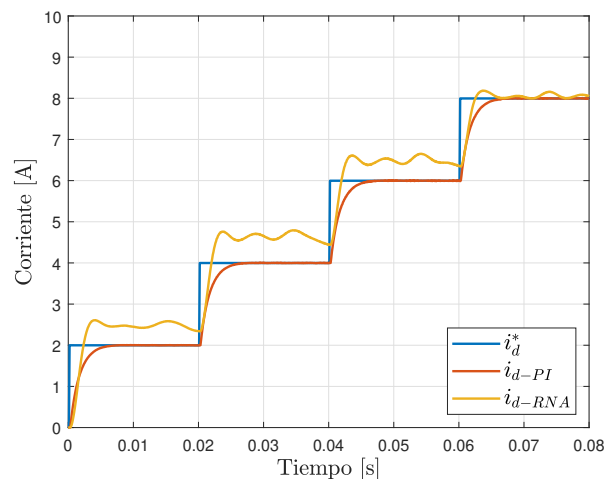


Figura 4.7. Comparación de corriente en el eje directo entre el control PI y el control en base a RNA

Se observa en la figura 4.7 que en comportamiento dinámico la RNA controla la corriente i_d con tiempos similares al control PI, sin embargo, en estado estacionario el error no es cero, debido a que al entrenar la RNA con distintos valores de resistencias $R = 15[\Omega]$, $R = 16[\Omega]$ y $R = 18[\Omega]$, la RNA promedia la magnitud de los índices de modulación de cada caso proporcionado en los datos de entrenamiento, en función de reducir el error total.

Tabla 4.3

COMPARACIÓN DEL TIEMPO DE ASENTAMIENTO ENTRE EL CONTROL PI Y EL CONTROL DE LA RNA

Control	Tiempo de asentamiento t_s
PI	4,5 [ms]
RNA	5 [ms]

De la tabla 4.3 se observa que los tiempos de asentamiento son similares para ambos casos de control.

4.1.3.a. Análisis de discrepancia en estado estacionario

De los resultados anteriores se puede observar que la red neurona artificial consigue capturar la acción proporcional del control PI, sin embargo, la acción integral no es adquirida por la RNA, evidenciado por la permanencia de un error en estado estacionario.

En una primera etapa de este trabajo de memoria se realizó el entrenamiento de la RNA utilizando datos de la operación del convertidor en entorno de simulación en un sistema con una resistencia de 15 [Ω]. El comportamiento del control en base a redes neuronales artificiales para este caso se muestra en la figura 4.8.

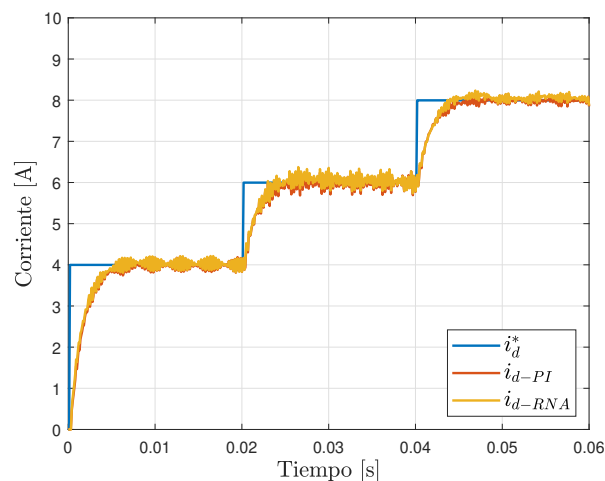


Figura 4.8. Comparación de corriente en el eje directo entre el control PI y el control en base a RNA, $R = 15[\Omega]$

En la figura 4.8 se observa que el comportamiento de la corriente i_d del control en base a redes neuronales es similar a la corriente del eje directo del control PI. Con esta RNA se prosiguió a la implementación en un prototipo de laboratorio en donde se observó que el control en base a RNA era deficiente y la falta de acción integral era evidente. La salida de la RNA era un índice de modulación proporcional a la corriente de referencia de la fase correspondiente. Dado que, en el entorno de simulación, los parámetros del sistema eran los mismos, la falta de acción integral no era apreciable. Sin embargo, la variabilidad de los parámetros del prototipo de laboratorio rápidamente evidenció la incorrecta relación de la RNA.

Para corregir este problema se decide entrenar la RNA con distintos valores de resistencia del sistema, reduciendo el sesgo de la RNA por los parámetros del sistema y permitiendo a la RNA generalizar la respuesta para un rango de valores de resistencia.

Como se observó en la sección previa, a pesar de los cambios realizados al conjunto de entrenamiento, la RNA no capturó la acción integral del controlador PI. Debido a que el controlador proporcional-integral es un controlador sencillo, pierde sentido el complejizar la RNA en la búsqueda de capturar el efecto integral y como se expuso en la sección 4.1 reducir la complejidad de la RNA realizando el control en coordenadas dq no significa una ventaja ante el uso de un control PI.

Con la experiencia adquirida durante el desarrollo de este trabajo de memoria se puede dar una explicación para la causa del problema y una recomendación de solución para futuros trabajos. El conjunto de entrenamiento contiene datos en estado transitorio y en estado estacionario que proporcionan la acción proporcional e integral del control PI, sin embargo, los datos en estado estacionario tienen una mayor proporción que los datos en estado transitorio. Lo anterior, no provoca que la RNA capture la acción de control que conlleva a error cero en estado estacionario, lo cual es característico de la parte integral del control PI, si no que, durante el entrenamiento el algoritmo intenta disminuir el error de entrenamiento encontrando un valor de salida (o índice de modulación, en nuestro caso) que promedie un menor error para todas las instancias del conjunto de entrenamiento. Por ello, para la RNA entrenada con resistencias de 15, 16 y 18 $[\Omega]$, en un sistema con una resistencia de 15 $[\Omega]$, se obtiene un índice de modulación y, por tanto, una corriente mayor que la de referencia. Esto se debe a que, para las resistencias de 16 y 18 $[\Omega]$, se requiere de un índice de modulación de mayor magnitud para obtener la misma corriente que en el caso de una resistencia de 15 $[\Omega]$, como se observa en la figura 4.7.

Por otra parte se sugiere como alternativa elegir para los datos de entrenamiento aquellos vectores que contengan un error entre el valor de referencia y la medición, distinto de cero o cercanos a cero. Es decir, priorizar aquellos datos transitorios entre el cambio de la referencia y la respuesta del sistema ante la acción del control PI. De esta manera, se espera que durante el entrenamiento la RNA ajuste los pesos y sesgos de manera efectiva en función de la parte proporcional e integral de la acción de control.

4.2. Control en base a RNA para estrategia de control predictivo

La estructura de la red neuronal artificial para la estrategia de control predictivo es una red de tipo *backpropagation network*, compuesta de una capa de oculta y una capa de salida.

El modelo de control predictivo con conjunto de estados finitos aplicado a un convertidor de tipo capacitor flotante, recibe como entradas las corrientes de referencia y medidas de cada fase, además debido a que este método controla activamente las tensiones en los condensadores de cada celda, se requiere de las tensiones de referencia y medidas en los terminales de estos condensadores. La red neuronal artificial utiliza como entrada estas mismas variables, sin embargo, se agregan los errores simples de corriente y tensión, además de una aproximación de la tensión de salida del convertidor por fase con respecto al neutro del enlace de tensión continua v'_{xN} , figura 2.2.

$$v'_{xN} = S_{3x}^{k-1} \mathbf{V}_{dc} - (S_{3x}^{k-1} - S_{2x}^{k-1})v_{2x}^k - (S_{2x}^{k-1} - S_{1x}^{k-1})v_{1x}^k \quad (4.1)$$

Donde S_{ix}^{k-1} con $i \in \{1, 2, 3\}$, $x \in \{a, b, c\}$ son los estados de conducción de cada celda en el instante $k-1$ de esta manera se obtiene un valor de tensión a la salida del convertidor sin mediciones extras. Si bien v'_{xN} es una tensión ficticia ya que combina valores en distintos instantes de tiempo, es una variable que permite proporcionarle a la RNA información sobre el estado de conmutación en el instante $k-1$ sin agregar mayor complejidad a la entrada, como sería el caso al considerar las tensiones v_{ix}^{k-1} o al aumentar el número de entradas al agregar directamente los tres estados de conducción por fase en el instante $k-1$.

El vector de entrada se define

$$\mathbf{x} = \{i_a^*, i_a, v_{1a}, v_{2a}, v_{c1}^* - v_{1a}, v_{c2}^* - v_{2a}, i_a^* - i_a, v'_{aN}, \\ i_b^*, i_b, v_{1b}, v_{2b}, v_{c1}^* - v_{1b}, v_{c2}^* - v_{2b}, i_b^* - i_b, v'_{bN}, \\ i_c^*, i_c, v_{1c}, v_{2c}, v_{c1}^* - v_{1c}, v_{c2}^* - v_{2c}, i_c^* - i_c, v'_{cN}\}.$$

Para el vector de salida se define una parametrización de los nueve estados de conducción S_{1a} , S_{2a} , S_{3a} , S_{1b} , S_{2b} , S_{3b} , S_{1c} , S_{2c} y S_{3c} a 24 estados de conducción SS_{ix} los cuales corresponden a las ocho combinaciones posibles de conducción por fase de S_{1x} , S_{2x} , S_{3x} , dados en la tabla

Tabla 4.4
PARAMETRIZACIÓN DE LOS ESTADOS DE CONDUCCIÓN POR FASE

S_{1x}	S_{2x}	S_{3x}	
0	0	0	SS_{1x}
0	0	1	SS_{2x}
0	1	0	SS_{3x}
0	1	1	SS_{4x}
1	0	0	SS_{5x}
1	0	1	SS_{6x}
1	1	0	SS_{7x}
1	1	1	SS_{8x}

Los estados de conducción SS_{ix} son variables que pueden tomar dos valores correspondientes a la activación $SS_{ix} = 1$ o no activación $SS_{ix} = 0$ de la combinación de estados i de la fase x con $i \in \{1, 2, \dots, 8\}$ y $x \in \{a, b, c\}$. Es decir, si el estado de conducción es $SS_{1a} = 1$ entonces se aplica la combinación de estados $\{S_{1a}, S_{2a}, S_{3a}\} = \{0, 0, 0\}$ en la fase a .

La mayoría de los esquemas de entrenamiento prediseñados de clasificación consideran la activación de una clase a la vez, por ende, el diseñar una red cuya salida sean los estados de conducción de cada celda y de cada fase del FCC, genera un vector de salida donde pueden estar activas nueve clases a la vez, provocando problemas en la discriminación de clases al entrenar estas redes prediseñadas. Es por ello que, al realizar esta parametrización, el vector de salida tendrá solo tres estados de conducción activos a la vez.

El vector de salida se define

$$\mathbf{y} = \{SS_{1a}, SS_{2a}, SS_{3a}, SS_{4a}, SS_{5a}, SS_{6a}, SS_{7a}, SS_{8a}, \\ SS_{1b}, SS_{2b}, SS_{3b}, SS_{4b}, SS_{5b}, SS_{6b}, SS_{7b}, SS_{8b}, \\ SS_{1c}, SS_{2c}, SS_{3c}, SS_{4c}, SS_{5c}, SS_{6c}, SS_{7c}, SS_{8c}\}$$

La red neuronal artificial requiere encontrar una relación entre las corrientes y tensiones de referencia y medidas, con el respectivo estado de conducción parametrizado, por ende, la red se desempeñará como una RNA de clasificación multi-clase multi-etiqueta, es decir clasificará entre ocho clases y etiquetará las tres fases a la vez.

Al igual que para el caso de la RNA entrenada con un control lineal tradicional en la sección 4.1 y la búsqueda de la cantidad de neuronas cercana al óptimo entre la magnitud de la RNA y el error de entrenamiento según lo mencionado en la sección 3.3.5.b. Se realiza un análisis exploratorio con 30, 34 y 38 neuronas en la capa oculta. Se estima que la dimensión óptima se encuentra entre 34 y 38 neuronas, se determina que la dimensión de la capa oculta sea de 36 neuronas.

La estructura de la red se compone de 24 entradas individuales, una capa oculta con 36 neuronas y una función de activación tangente hiperbólica, y la capa de salida tiene 24 neuronas con una función de activación sigmoidea, como se muestra en la figura 4.9.

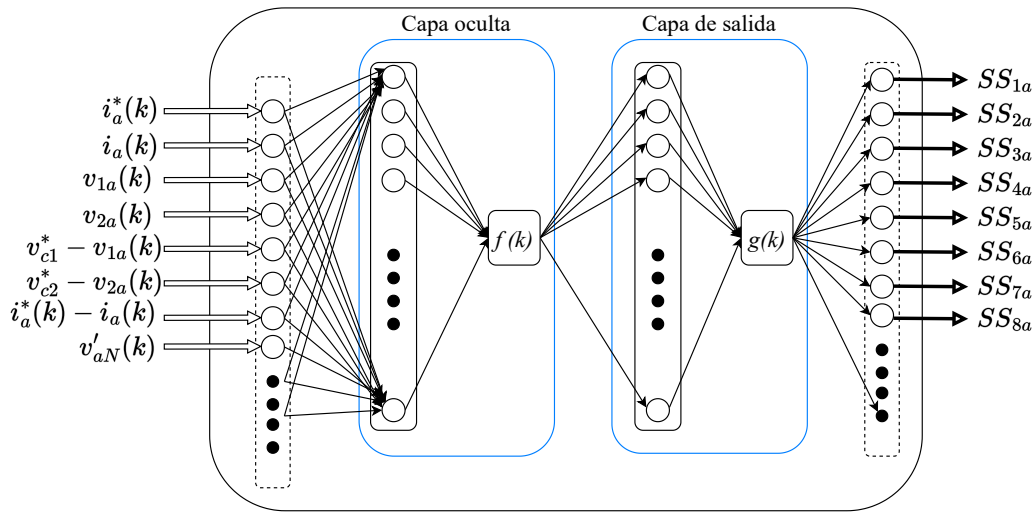


Figura 4.9. Diagrama de RNA aplicada a la estrategia de control predictivo

Dado que la salida de la red son estados de conducción parametrizados SS_{ix} , esta salida

tendrá que ser procesada para volver a los estados de conducción de cada celda S_{ix} que irán al convertidor. El procesamiento de la salida de la RNA consiste en transformar las variables de salida, que pertenecen al rango $[0, 1] \in \mathfrak{R}$, a un estado de conducción SS_{ix} válido. Esto se realiza activando el estado de conducción con el valor máximo entre los ocho posibles estados de la misma fase, eligiendo así un único estado con el mayor nivel de certeza por parte de la RNA. Este proceso se replica para cada fase, obteniéndose al final un estado de conducción válido por fase. Por ejemplo si la salida para la fase a es $\{0,1,0,11,0,4,0,9,0,2,0,5,0,01,0,05\}$ el estado de conducción activo es $SS_{4a} = 1$ y los demás se mantendrán en cero.

El esquema de control en base a redes neuronales aplicada a la estrategia de control predictivo se muestra en la figura 4.10.

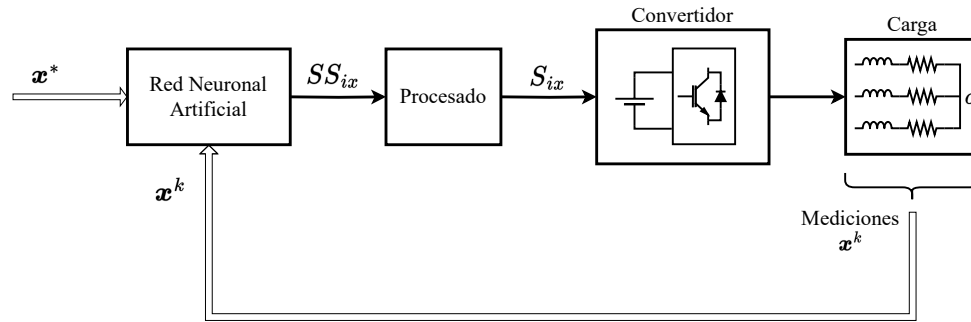


Figura 4.10. Esquema de control en base a RNA aplicada a la estrategia de control predictivo

4.2.1. Datos de entrenamiento para el caso de control predictivo

En la figura 4.11 se muestran los datos de la fase a utilizados para el entrenamiento de la red neuronal artificial. Estos datos consisten en las corrientes de referencia i_a^* , i_b^* y i_c^* , las tensiones de referencia v_{c1a}^* , v_{c2a}^* , v_{c1b}^* , v_{c2b}^* , v_{c1c}^* y v_{c2c}^* , las corrientes medidas i_a , i_b y i_c , las tensiones medidas de los capacitores v_{c1a} , v_{c2a} , v_{c1b} , v_{c2b} , v_{c1c} y v_{c2c} en la operación de un FCC con control FCS-MPC en entorno de simulación, alimentando una carga RL con resistencia de $R = 15[\Omega]$ y un inductor de $10[mH]$, la frecuencia de muestreo del control es de $10[kHz]$.

Para obtener los datos se varia la amplitud A de las corrientes de referencia $i_x^* = A \cos(\omega t + \phi_x)$ con $x \in \{a, b, c\}$ en $A \in \{0,1,2,3,4,5,6,7,8,9,10,-1,-2,-3,-4,-5,-6,-7,-8,-9,-10\}$. Si bien, se da mayor proporción a los datos dentro el rango $[5,10]$ para obtener un mejor ajuste en dicho rango. También se añade al conjunto de datos, la respuesta del FCS-MPC ante escalones de voltaje los capacitores de cada celda de cada fase del convertidor.

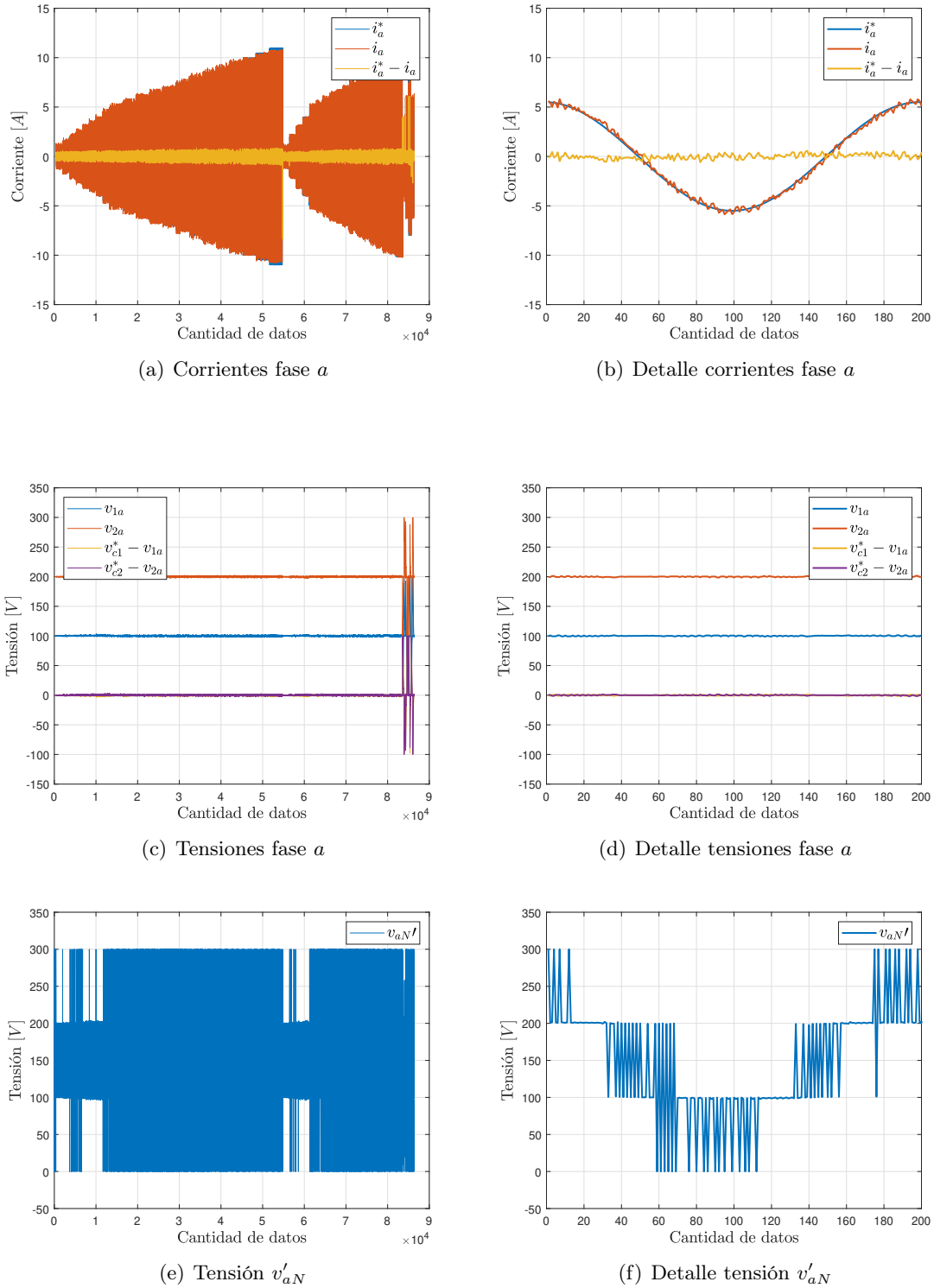


Figura 4.11. Conjunto de datos de entrada de la fase a para el entrenamiento de la RNA para caso MPC

En la figura 4.11 se muestran los datos de entrenamiento para la fase a por simplicidad, pero el conjunto de datos de entrenamiento completo contiene la información para las demás

fases.

Los datos de salida corresponden a los 24 estados de conducción SS_{ix} respectivos a los datos de entrada. En la figura 4.12 se muestra los ocho estados de conducción parametrizados correspondientes a la fase a .

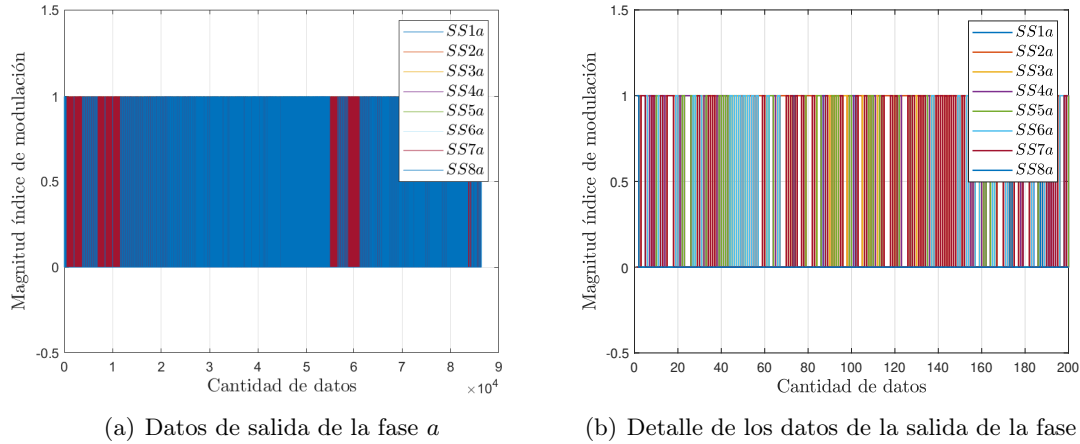


Figura 4.12. *Conjunto de datos de salida de la fase a para el entrenamiento de la RNA para caso MPC*

El conjunto de datos de entrenamiento se compone de 86.399 vectores individuales de entrada y salida, de los cuales se escoge para el entrenamiento el 95 % de los datos y el 5 % se escoge para las pruebas durante el entrenamiento, la división de los datos es de manera aleatoria. Al igual que para el caso del control PI, el objetivo de la RNA es imitar el control FCS-MPC el sesgo de ajuste a los datos de entrenamiento de la RNA no representa necesariamente una característica negativa.

4.2.2. Creación y entrenamiento de la RNA para el caso de control predictivo

En la sección 4.2 se determinó la estructura de la RNA, la dimensión y componentes del vector de entrada y de salida, además de la dimensión de la capa oculta.

Los rangos de entrada y salida se determinan según el rango de operación de las variables de entrada, según lo mencionado en la sección 3.3.5.c. Se define el rango de las corrientes medidas y de referencia en un rango de -15 [A] a 15 [A], El error de corriente oscila mayoritariamente entre $\pm 0,5$ [A] definiéndose un rango de -2 [A] a 2 [A]. Las tensiones medidas se sitúan entre 0 y 300 [V], y los errores de tensión entre -100 y 100 [V].

Al igual que el entrenamiento de la RNA para el caso de control lineal, el algoritmo de entrenamiento seleccionado es el *Gradient descent with adaptive learning rate backpropagation*.

Se genera una red neuronal artificial utilizando a modo de plantilla la red tipo *patternet*, definiendo 36 neuronas en la capa oculta, se modifican los parámetros de inicialización de la red, se establecen los rangos de entrada y salida, además de establecer el algoritmo de entrenamiento de la red y parámetros asociados. El código resumen se muestra a

continuación.

```
trainFcn = 'traingda'; %Funcion de entrenamiento

% Creacion una red Pattern Recognition Network
hiddenLayerSize = 36;
net = fitnet(hiddenLayerSize, trainFcn);
net.layers{1}.transferFcn = 'tansig';
net.layers{2}.transferFcn = 'logsig';

% Inicializacion de la red neuronal artificial
net.layers{:}.initFcn = 'initwb';
net.inputWeights{1}.initFcn = 'rands';
net.layerWeights{2}.initFcn = 'rands';
net.biases{:}.initFcn = 'rands';
net = init(net);

% Rangos de entrada y salida
net.inputs.range = {[ -15,15; -15,15; 0,300; 0,300; -100,100; -100,100; -30,30;
    0,300; -15,15; -15,15; 0,300; 0,300; -100,100; -100,100; -30,30; 0,300;
    -15,15; -15,15; 0,300; 0,300; -100,100; -100,100; -30,30; 0,300]};

% Definir division de los datos para Entrenamiento, Validacion y Prueba
net.divideParam.trainRatio = 90/100;
net.divideParam.valRatio = 0/100;
net.divideParam.testRatio = 10/100;
```

La red neuronal artificial *patternet* es una red que tiene una capa oculta con una función de activación tangencial hiperbólica y una capa de salida con una función de activación *Softmax*. Dado que la función *Softmax* realiza una ponderación entre todas las salidas de la capa de salida, eligiendo la que tenga el valor mayor, esta función no se adecua a nuestro caso dado que debido a las tres fases es necesario seleccionar tres clases a la vez. Es por ello que se modifica la función de activación de la capa de salida por una función Sigmoidea, lo que permite mediante código y según lo descrito en sección 4.1, seleccionar la clase con mayor porcentaje de certeza por fase.

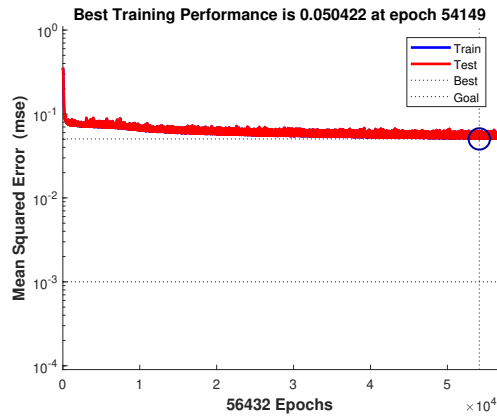
Para entrenar la red se hace uso de la función *train* de MATLAB. En la tabla 4.5 se resumen los resultados del entrenamiento.

Tabla 4.5
RESUMEN DE ENTRENAMIENTO RNA, PARA EL CASO DE CONTROL PREDICTIVO

Parámetro	Valor
Función de entrenamiento	<i>Gradient descent with adaptive learning rate backpropagation</i>
Número de épocas	56432
Tiempo	129.1 [min]
Función de desempeño	Mean Squared Error
Mejor error de desempeño	0.0504

El proceso de entrenamiento se interrumpe en la época 56432 dado que la variación del descenso del error es mínima, evitando así el sobre ajuste de la RNA con los datos de entrenamiento

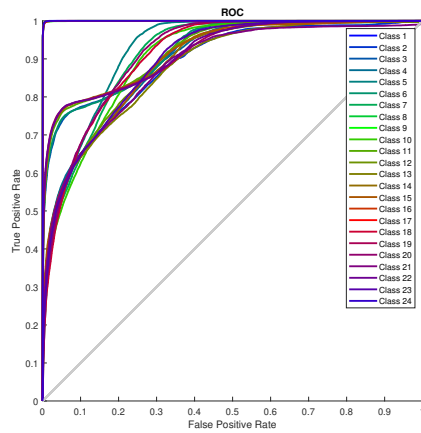
En la figura 4.13 se muestran los gráficos representativos del entrenamiento de la red



(a) Desempeño



(b) Histograma de error



(c) ROC

Confusion Matrix

	1	2	3	4	5	6	7	8	
1	11412 13.2%	127 0.1%	199 0.2%	0 0.0%	218 0.3%	0 0.0%	7 0.0%	0 0.0%	95.4% 4.6%
2	24 0.0%	7413 8.6%	2012 2.3%	240 0.3%	3360 3.9%	212 0.2%	23 0.0%	0 0.0%	55.8% 44.2%
3	57 0.1%	361 0.4%	6396 7.4%	29 0.0%	776 0.9%	0 0.0%	37 0.0%	0 0.0%	83.5% 16.5%
4	0 0.0%	275 0.3%	110 0.1%	5859 6.8%	16 0.0%	551 0.6%	906 1.0%	27 0.0%	75.7% 24.3%
5	17 0.0%	2514 2.9%	954 1.1%	53 0.1%	6047 7.0%	67 0.1%	344 0.4%	0 0.0%	60.5% 39.5%
6	0 0.0%	28 0.0%	0 0.0%	471 0.5%	24 0.0%	6348 7.3%	424 0.5%	10 0.0%	86.9% 13.1%
7	0 0.0%	208 0.2%	271 0.3%	3954 4.6%	455 0.5%	2494 2.9%	8986 10.4%	17 0.0%	54.8% 45.2%
8	0 0.0%	43 0.0%	0 0.0%	225 0.3%	0 0.0%	211 0.2%	165 0.2%	11422 13.2%	94.7% 5.3%
	99.1% 0.9%	67.6% 32.4%	64.3% 35.7%	54.1% 45.9%	55.5% 44.5%	64.2% 35.8%	82.5% 17.5%	99.5% 0.5%	73.9% 26.1%

Output Class

Target Class

(d) Matriz de confusión fase a

Confusion Matrix

	1	2	3	4	5	6	7	8	
1	11260 13.0%	289 0.3%	161 0.2%	0 0.0%	180 0.2%	0 0.0%	0 0.0%	0 0.0%	94.7% 5.3%
2	11 0.0%	6241 7.2%	1380 1.6%	673 0.8%	2906 3.4%	106 0.1%	140 0.2%	0 0.0%	54.5% 45.5%
3	37 0.0%	653 0.8%	6589 7.6%	34 0.0%	276 0.3%	0 0.0%	43 0.0%	0 0.0%	86.3% 13.7%
4	0 0.0%	486 0.6%	315 0.4%	8041 9.3%	102 0.1%	2135 2.5%	4186 4.8%	36 0.0%	52.6% 47.4%
5	46 0.1%	3303 3.8%	1427 1.7%	47 0.1%	7230 8.4%	192 0.2%	576 0.7%	0 0.0%	56.4% 43.6%
6	0 0.0%	30 0.0%	0 0.0%	635 0.7%	60 0.1%	7154 8.3%	824 1.0%	78 0.1%	81.5% 18.5%
7	0 0.0%	0 0.0%	59 0.1%	1372 1.6%	236 0.3%	326 0.4%	4922 5.7%	24 0.0%	70.9% 29.1%
8	0 0.0%	0 0.0%	0 0.0%	102 0.1%	0 0.0%	111 0.1%	161 0.2%	11204 13.0%	96.8% 3.2%
	99.2% 0.8%	56.7% 43.3%	66.3% 33.7%	73.7% 26.3%	65.8% 34.2%	71.4% 28.6%	45.4% 54.6%	98.8% 1.2%	72.5% 27.5%

Output Class

Target Class

(e) Matriz de confusión fase b

Confusion Matrix

	1	2	3	4	5	6	7	8	
1	11257 13.0%	148 0.2%	162 0.2%	0 0.0%	159 0.2%	1 0.0%	0 0.0%	0 0.0%	96.0% 4.0%
2	33 0.0%	6782 10.2%	2289 2.6%	374 0.4%	3939 4.6%	198 0.2%	90 0.1%	0 0.0%	55.9% 44.1%
3	51 0.1%	472 0.5%	6817 7.9%	34 0.0%	614 0.7%	0 0.0%	34 0.0%	0 0.0%	85.0% 15.0%
4	0 0.0%	444 0.5%	88 0.1%	5663 6.6%	77 0.1%	836 1.0%	2339 2.7%	17 0.0%	59.8% 40.2%
5	20 0.0%	906 1.0%	283 0.3%	4 0.0%	5475 6.3%	54 0.1%	267 0.3%	0 0.0%	78.1% 21.9%
6	0 0.0%	32 0.0%	0 0.0%	634 0.7%	17 0.0%	7053 8.2%	438 0.5%	51 0.1%	85.8% 14.2%
7	0 0.0%	110 0.1%	243 0.3%	3940 4.6%	651 0.8%	1767 2.0%	7762 9.0%	21 0.0%	53.6% 46.4%
8	0 0.0%	0 0.0%	0 0.0%	174 0.2%	5 0.0%	142 0.2%	110 0.1%	11322 13.1%	96.3% 3.7%
	99.1% 0.9%	80.6% 19.4%	69.0% 31.0%	52.3% 47.7%	50.1% 49.9%	70.2% 29.8%	70.3% 29.7%	99.2% 0.8%	74.2% 25.8%

Output Class

Target Class

(f) Matriz de confusión fase c

Figura 4.13. Gráficos representativos del entrenamiento y desempeño, caso control predictivo

La figura 4.13(a) muestra el progreso de error MSE en cada época, durante el proceso de entrenamiento se busca conseguir el menor error MSE. La figura 4.13(b) muestra el histograma de error de la última época, se busca que todos los errores de cada instancia tiendan a cero, tanto en distribución como en magnitud.

En la figura 4.13(c) se muestra curva de característica operativa del receptor o ROC (por sus siglas en inglés *Receiver operating characteristic*), para las 24 clases de salida de la RNA. La curva ROC es un gráfico que muestra la relación entre la clase de salida de la RNA y la clase real de la instancia, siendo una representación gráfica del desempeño de una RNA de clasificación. En particular, la curva ROC presenta la tasa de verdaderos positivos, que es la razón entre las instancias correctamente clasificadas y la suma de las clasificaciones correctas e incorrectas. De igual manera, la tasa de falsos positivos es la razón entre las instancias erróneamente clasificadas y la suma de las clasificaciones correctas e incorrectas. En el eje de las ordenadas se grafica la tasa de verdaderos positivos y en el eje de las abscisas la tasa de falsos positivos. Un buen desempeño de la RNA tiende a concentrar los datos en la parte superior izquierda de la curva ROC.

En la figura 4.13(d), 4.13(e) y 4.13(f) se muestra la matriz de confusión para las clases correspondientes a la fases *a*, *b* y *c*, respectivamente. La matriz de confusión también es una representación gráfica del desempeño de una RNA de clasificación, con la diferencia de que nos permite visualizar la distribución de los errores de clasificación. Cada fila de la matriz de confusión muestra las instancias clasificadas por la RNA en cierta clase, y cada columna corresponde a la clase real de las instancias. Es decir, en la fila uno están todas las instancias clasificadas por la RNA como clase uno. En la celda que intersecta con la columna uno están todas las instancias que coinciden con esta clasificación, mientras que en la celda que intersecta con la columna dos están aquellas instancias que fueron clasificadas por la RNA como clase uno, pero que en realidad pertenecían a la clase dos. De esta manera ocurre con las demás clases. Cada vez que la RNA clasifica una instancia en una clase incorrecta, se denomina *confusión*, lo que da nombre a esta matriz. El objetivo de una RNA de clasificación es asignar correctamente cada instancia; por lo tanto, se espera que todas las instancias clasificadas se concentren en la diagonal de la matriz de confusión.

De la figura 4.13 se observa que existe una buena correlación entre los datos de entrada y la correcta clasificación de la RNA. En particular, en la curva ROC se visualiza que todas las clases tienen una mayor tasa de instancias clasificadas correctamente que incorrectamente. Además, en las matrices de confusión de cada fase se observa que todas las instancias clasificadas se concentran en las diagonales, lo que indica un buen desempeño de la RNA.

4.2.3. Implementación de la RNA en entorno de simulación para el caso de control predictivo

Mediante el entorno de simulación PLECS se implementa el control basado en redes neuronales, donde los pesos y sesgos de la red neuronal artificial ya entrenada se almacenan en un archivo con extensión *.h* como matrices, para luego ser llamados en el código del control.

La frecuencia de muestreo del control es de 10 [kHz]. El convertidor FC alimenta una carga RL con resistencia de $R = 15[\Omega]$ y una inductancia de $L = 10[mH]$, siendo así consecuente con las condiciones de los datos de entrenamiento.

En la figura 4.14 se muestra la forma de onda de corriente de la fase *a* para el FCC

controlado con la red neuronal artificial aplicada a la estrategia de control predictivo, la corriente de referencia para dicha fase, las corrientes en coordenadas dq y las tensiones en los condensadores del FCC.

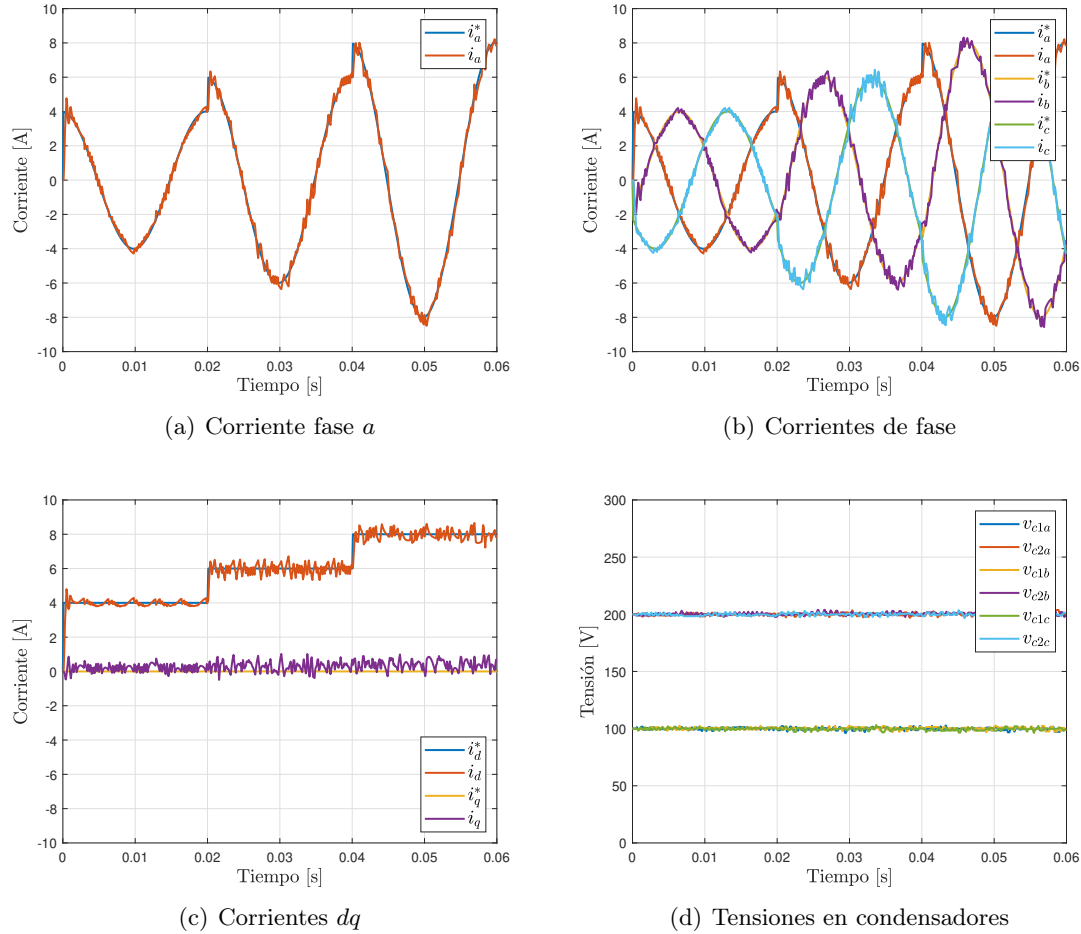


Figura 4.14. Corrientes y tensiones medidas para control basado en RNA, caso control predictivo

En la figura 4.14 se observa que hay control de las corrientes y de las tensiones en todas las fases. Existe seguimiento de referencia y buen comportamiento en estado estacionario. Para evaluar la calidad del control basado en RNA en estado estacionario, se utilizará el índice THD en comparación con el control predictivo, como se presenta en la tabla 4.6. Para evaluar el desempeño dinámico del control de la RNA, se medirá el tiempo de asentamiento de la componente de corriente en el eje d y se comparará con el control predictivo.

Tabla 4.6

COMPARACIÓN DE THD ENTRE EL CONTROL PI Y EL CONTROL DE LA RNA EN FUNCIÓN DE LA REFERENCIA DE AMPLITUD DE CORRIENTE

$A_{ref}[A]$	2	4	6	8	10
THD MPC [%]	12,7	4,9	4,2	3,2	2,6
THD ANN [%]	18,7	4,7	6,3	4,5	3,5

En la tabla 4.6 se observa que, para una magnitud de referencia de corriente de valor 2[A], el THD es mayor en ambos métodos. Esto se debe a que, para una corriente pequeña, los armónicos de corriente de bajo orden, producto de las conmutaciones no periódicas del FCS-MPC y de las diferencias en la velocidad de carga y descarga de los elementos reactivos, provocan un mayor impacto, lo que se refleja en el THD. Por lo general se observa que el THD para el control en base a RNA es mayor que en el caso del control predictivo. Debido a que no todas las conmutaciones dadas por la RNA entrenada son adecuadas, provocando componentes armónicas en la tensión de salida del convertidor, adicionales a las presentes en un control FCS-MPC.

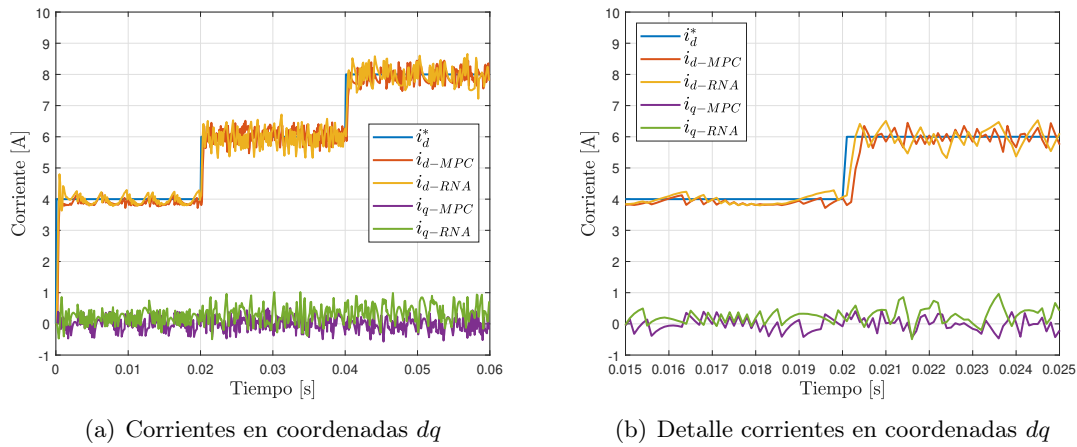


Figura 4.15. Comparación de corriente en el eje directo entre el control FCS-MPC y el control en base a RNA

Se observa en la figura 4.15 que el comportamiento dinámico del control en base a RNA es muy similar al comportamiento del control predictivo. En la tabla 4.7 se presentan los tiempos de asentamiento entre el control FCS-MPC y el control en base a redes neuronales artificiales.

Tabla 4.7
COMPARACIÓN DEL TIEMPO DE ASENTAMIENTO
ENTRE EL CONTROL FCS-MPC Y EL CONTROL DE LA RNA

Control	Tiempo de asentamiento t_s
FCS-MPC	0,6 [ms]
RNA	0,5 [ms]

De la tabla 4.7 se observa que los tiempos de asentamiento son similares en ambos métodos de control. Esto implica que la RNA también tiene la respuesta rápida que caracteriza al control predictivo.

4.3. Resultados experimentales en prototipo de laboratorio

4.3.1. Configuración de prototipo de laboratorio

El convertidor trifásico de capacitor flotante de cuatro niveles está compuesto por tres placas, una para cada fase, que constituyen el arreglo de celdas por fase mostrado en la figura 2.1. Estas placas integran el circuito de disparo y el circuito de medición de corriente y tensión de los condensadores. La comunicación entre la plataforma de control y el convertidor se realiza mediante un enlace de fibra óptica para los pulsos de disparo, utilizando la tarjeta de expansión PowerEloBase de la plataforma BRAIn, que permite enviar la señal de fibra óptica a una tarjeta receptora acoplada a la placa del convertidor. Las mediciones provenientes de la placa del convertidor se reciben mediante la misma tarjeta de expansión PowerEloBase hacia la plataforma BRAIn, como se muestra en el esquema de la figura 4.16.

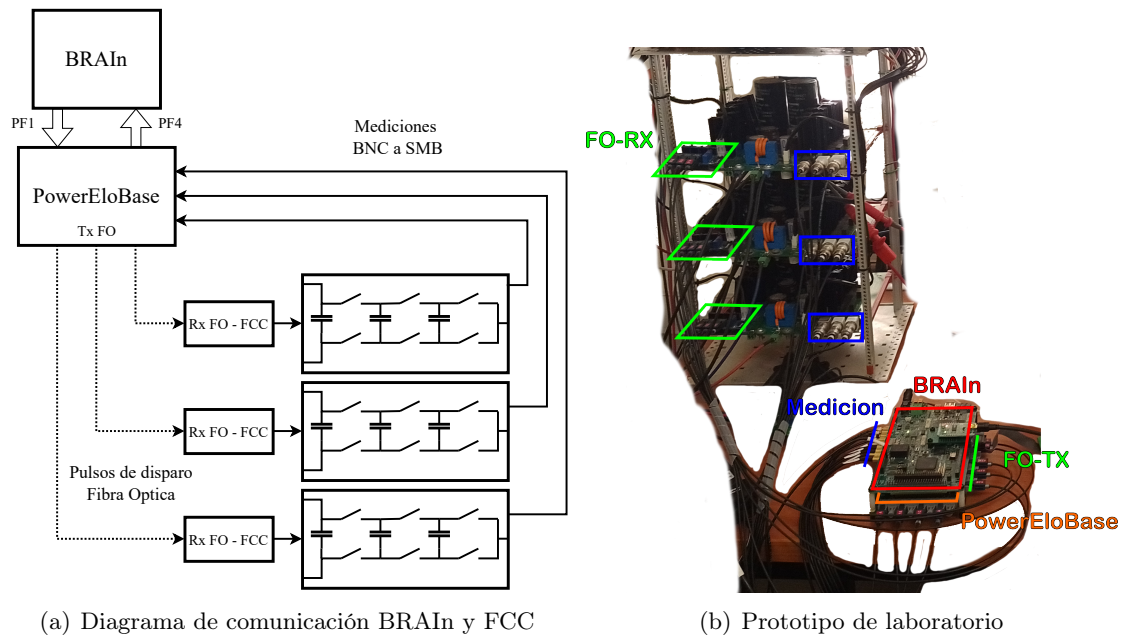


Figura 4.16. Diagrama de comunicación BRAIn y FCC de prototipo de laboratorio

Los parámetros del sistema de prueba se describen en la tabla 4.8.

Tabla 4.8

PARÁMETROS GENERALES DEL SISTEMA DE PRUEBA DE LABORATORIO

Elemento	Parámetro	Valor
Placa capacitor flotante	C_1	330 [μ F]
	C_2	330 [μ F]
	C_3	900 [μ F]
	Switch	FGA40N60UFD
	Banda de medición de corriente	0 - 3000 [Hz]
Carga	R_1	17 [Ω]
	L_1	10.1 [mH]
	R_2	18 [Ω]
	L_2	10.3 [mH]
	R_3	15.7 [Ω]
	L_3	9.9 [mH]

4.3.2. Plataforma de control BRAIn

La plataforma de control digital BRAIn, del inglés *Board for Research, Academic and Industrial applications*, la cual combina un procesador digital de señales (DSP del inglés Digital Signal Processor) y una FPGA (del inglés Field Programmable Gate Array). La plataforma nos permite programar las distintas estrategias de control en procesador DSP y mediante la FGPA utilizar los módulos y periféricos disponibles en la BRAIn, en particular moduladores PWM y ADCs [24].

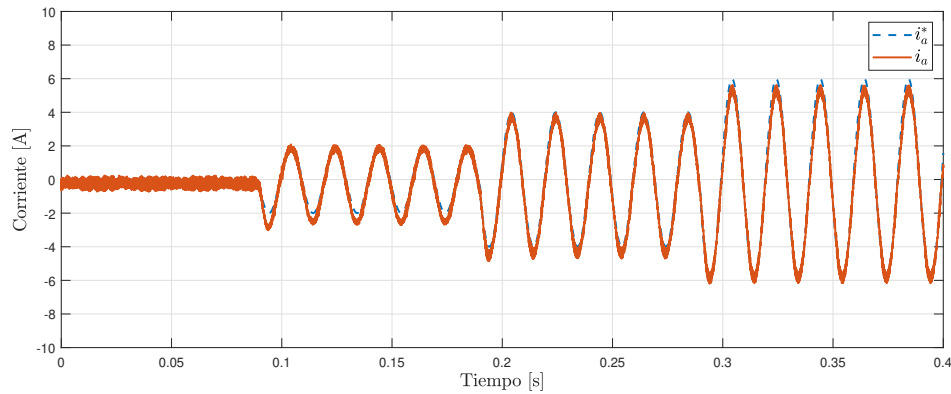
El procesador DSP de la plataforma de control BRAIn se puede programar mediante el entorno integrado de desarrollo *Code Composer Studio*, utilizando el lenguaje de programación C. La plataforma BRAIn dispone de librerías que facilitan el uso de los módulos y periféricos, y al mismo tiempo permite un alto grado de personalización del programa. Si se desea, también es posible modificar el firmware de la FPGA de la plataforma BRAIn para lograr un mayor grado de configuración.

La plataforma BRAIn cuenta con un DSP TMS320C6748 de *Texas Instruments*, una FPGA XCA35T, 120 puertos GPIO y 12 canales analógicos, entre otras cualidades que la hacen adecuada para usos académicos e industriales.

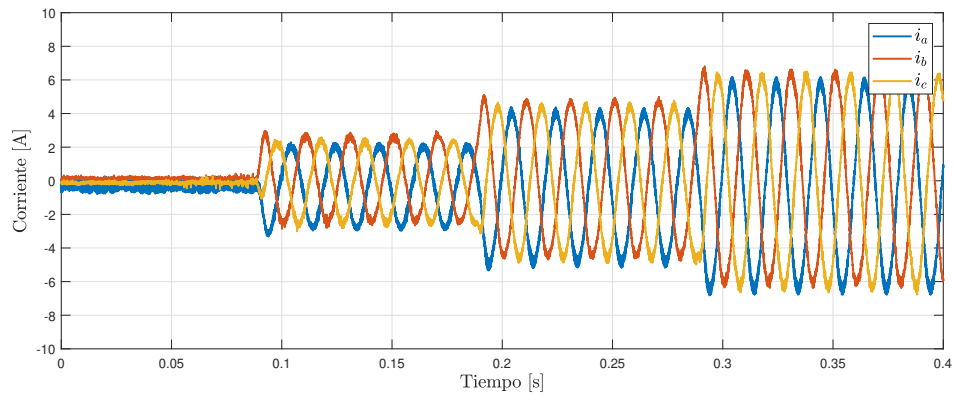
4.3.3. Implementación de la RNA para la estrategia de control lineal

Replicando las condiciones indicadas en la sección 4.1.1, se establece una frecuencia de muestreo de la DSP de 10 [kHz] y la frecuencia de la señal portadora de 1.666 [kHz].

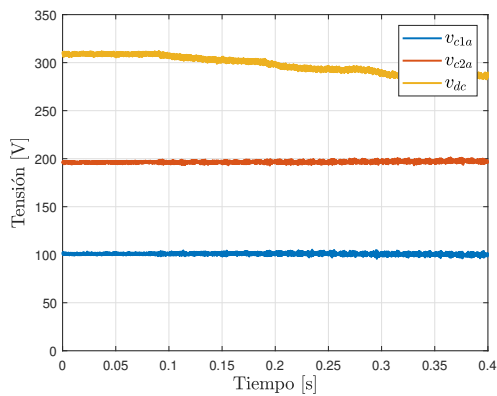
En la figura 4.17 se muestra las mediciones realizadas con el osciloscopio para el FCC controlado con la red neuronal artificial aplicada a la estrategia de control lineal en el prototipo de laboratorio: la corriente de la fase *a*, las corrientes de cada fase, las tensiones en los condensadores de la fase *a* y la tensión de salida del FCC de la fase *a*.



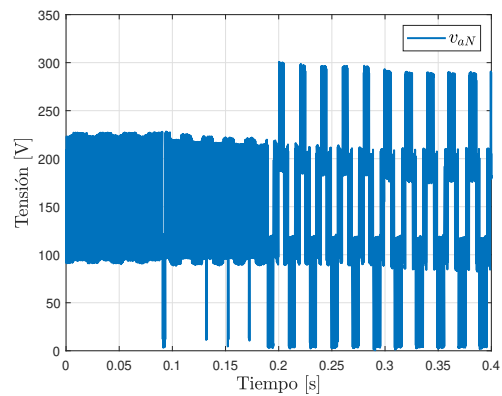
(a) Corriente fase *a*



(b) Corrientes de fase



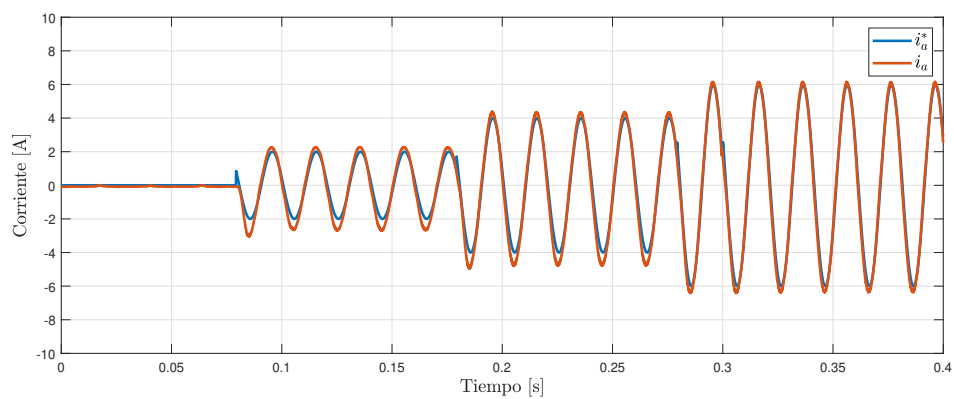
(c) Tensiones en condensadores



(d) Tensión de salida de la fase a con respecto a su neutro N

Figura 4.17. Corrientes y tensiones medidas con control mediante RNA, en prototipo de laboratorio

En la figura 4.18 se muestra las mediciones obtenidas por la BRAIn.



(a) Corriente fase a

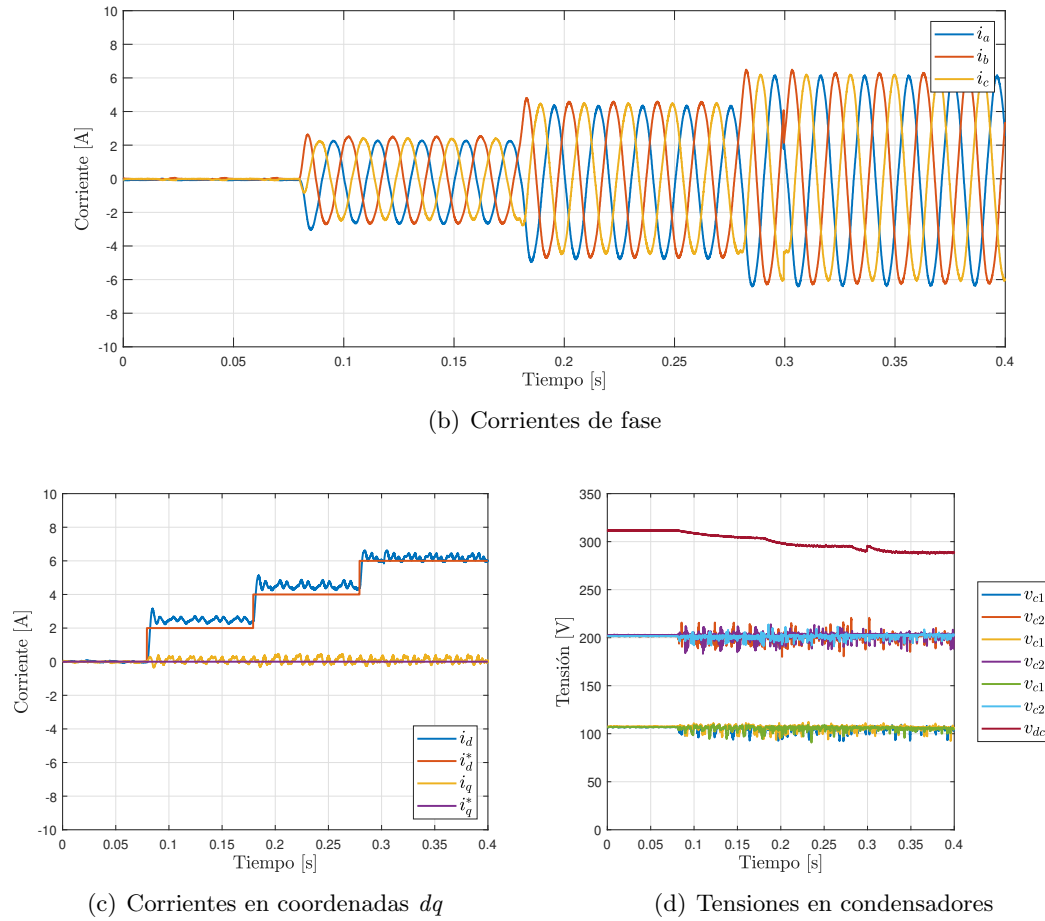


Figura 4.18. Corrientes y tensiones medidas con control mediante RNA, medidas por la BRAIn

En las figuras 4.17 y 4.18 Se observa un comportamiento proporcional entre el cambio en la referencia de corriente y las corrientes controladas por la red neuronal artificial en todas las fases y al igual que lo observado en entorno de simulación hay un error en estado estacionario de magnitud 0.49 [A] en promedio. Se observa que la tensión en el *dc-link* disminuye a medida que aumenta la corriente, pero las tensiones en los condensadores se mantienen, ya que, debido a la corta duración de la maniobra, el balance de tensión de la modulación PS-PWM no es apreciable.

En la tabla 4.9 se muestra la comparación de THD para distintas magnitudes de referencia de corriente entre el control PI y el control de la RNA implementado en el prototipo de laboratorio.

Tabla 4.9

COMPARACIÓN DE THD ENTRE EL CONTROL PI Y EL CONTROL DE LA RNA EN FUNCIÓN DE LA REFERENCIA DE AMPLITUD DE CORRIENTE, PARA PROTOTIPO DE LABORATORIO

$A_{ref}[A]$	2	4	6	8	10
THD PI [%]	4.88	3.23	2.65	1.99	2.31
THD RNA [%]	5.76	3.9	2.91	2.56	3.0

En la tabla 4.9 se observa una disminución del THD al aumentar la amplitud de la referencia de corriente, para el caso del control PI y el control basado en RNA, repitiendo el comportamiento visto en la tabla 4.2. Para las distintas amplitudes de corriente de referencia, el THD del control de la red neuronal artificial es mayor en comparación al control PI, sin embargo, las diferencias son menores al 1 %.

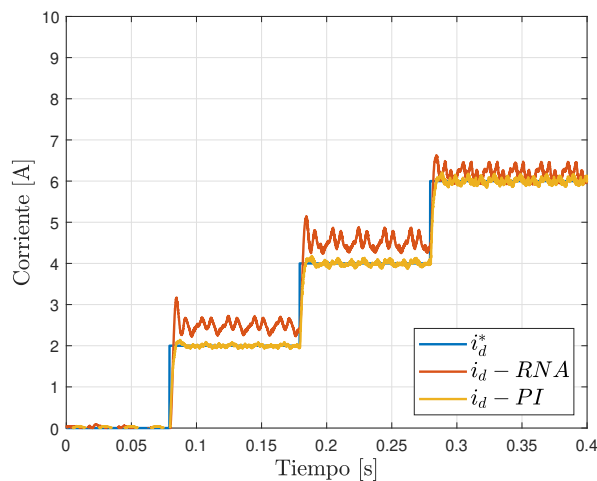
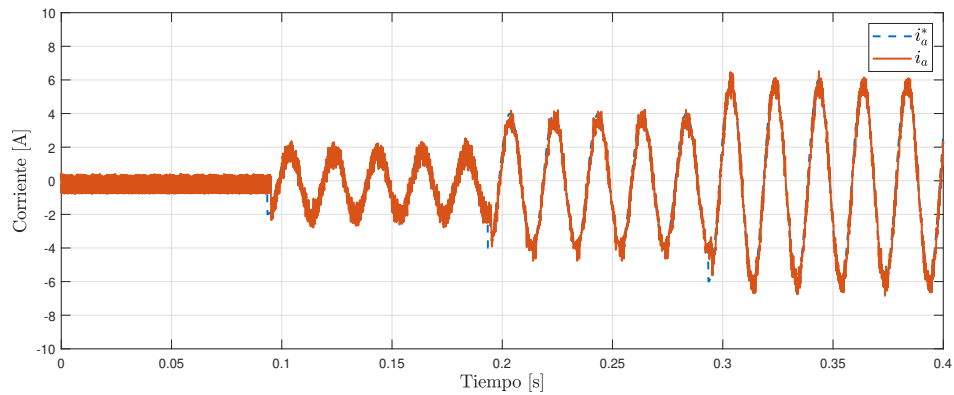
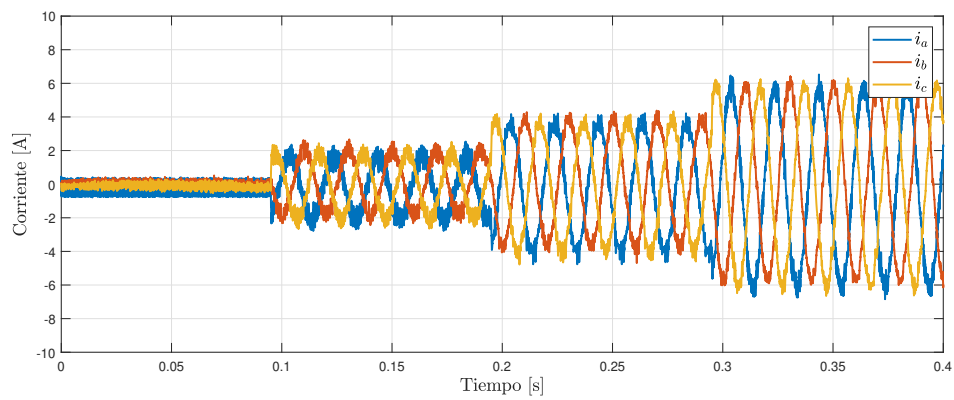


Figura 4.19. Comparación de corriente en el eje directo entre el control PI y el control en base a RNA, en prototipo de laboratorio

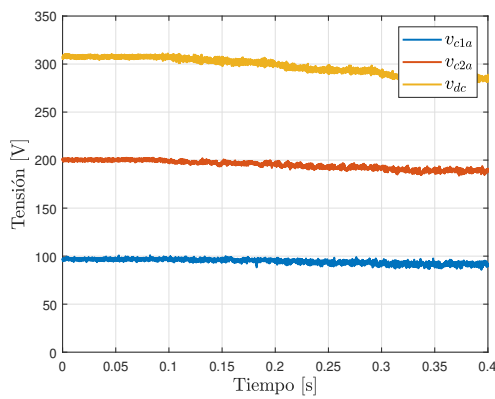
En la figura 4.19 se muestra la corriente en el eje directo para ambas estrategias de control en el prototipo de laboratorio. En esta figura se observa que se repite el comportamiento observado en el entorno de simulación, según lo mencionado en la sección 4.1.3.a. Esto es la mayor magnitud de corriente i_d en control basado en RNA en comparación a la referencia en estado estacionario y el mayor ruido presente en esta.

4.3.4. Implementación de la RNA para la estrategia de control predictivo

Se replican las condiciones indicadas en la sección 4.2.1 en el prototipo de laboratorio. En la figura 4.20 se muestran las mediciones realizadas con el osciloscopio para el FCC controlado con la red neuronal artificial aplicada a la estrategia de control predictivo en el prototipo de laboratorio: la corriente de la fase a , las corrientes de cada fase, las tensiones en los condensadores de la fase a y la tensión de salida del FCC de la fase a

(a) Corriente fase a 

(b) Corrientes de fase



(c) Tensiones en condensadores

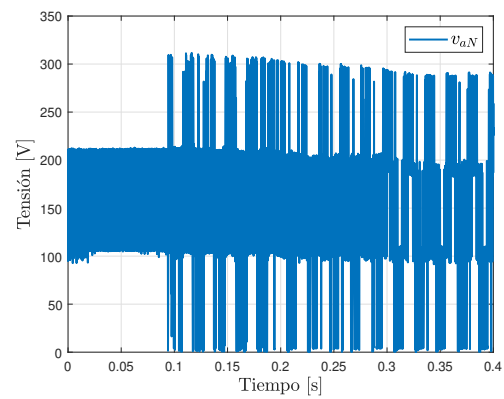
(d) Tensión de salida de la fase a con respecto a su neutro N

Figura 4.20. Corrientes y tensiones medidas con control en base a RNA, en prototipo de laboratorio

En la figura 4.21 se muestran las mediciones obtenidas por la BRAIn.

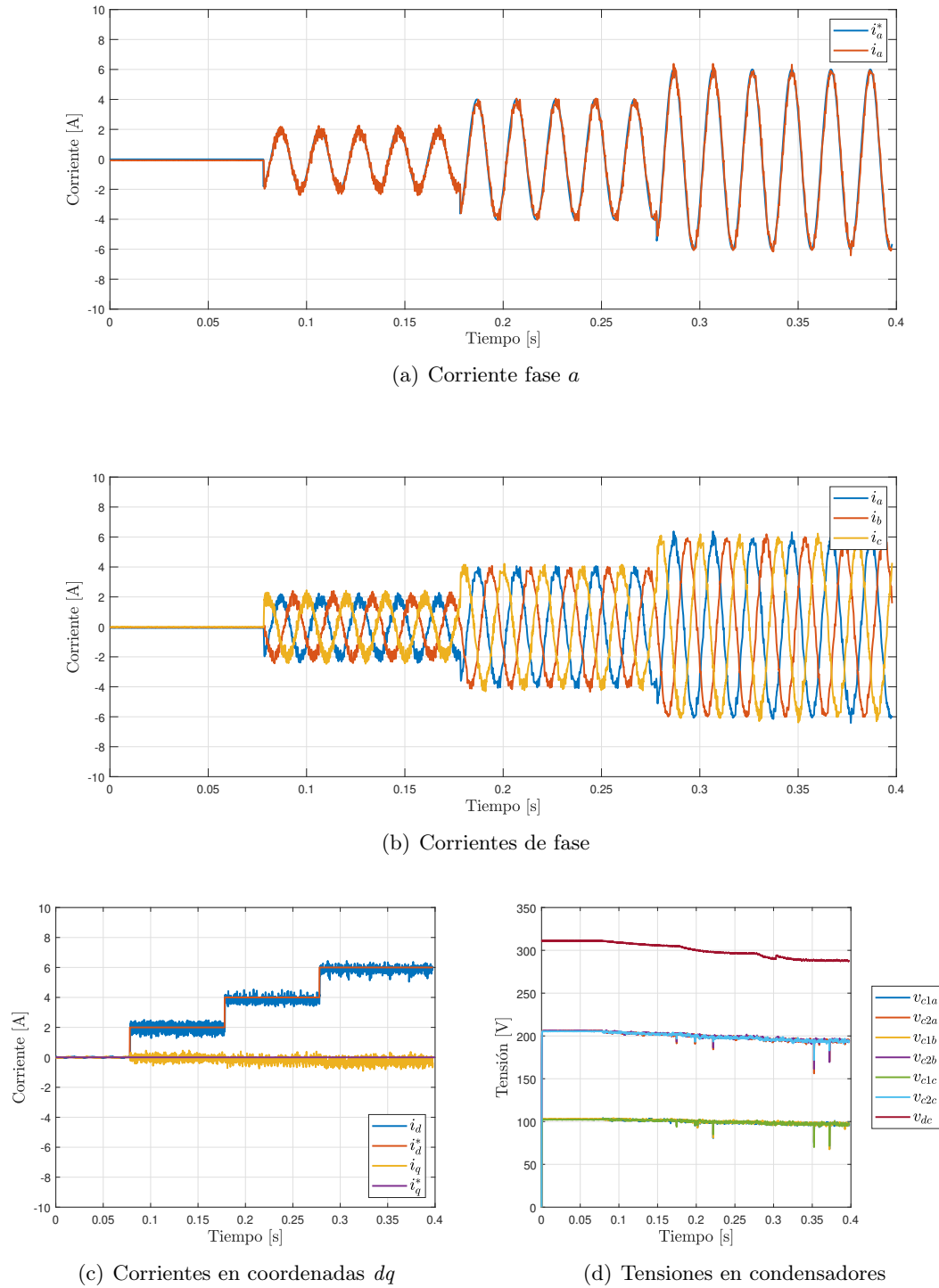


Figura 4.21. Corrientes y tensiones medidas con control mediante RNA, medidas por la BRAIn

En las figuras 4.20 y 4.21 se observa que hay seguimiento de la referencia para las corrientes controladas por la red neuronal artificial para todas las fases, las tensiones en los condensadores también se mantienen en balance con respecto a la tensión medida en el dc -link debido al control de la RNA.

En la tabla 4.10 se muestra la comparación de THD para distintas magnitudes de referencia de corriente entre el control FCS-MPC y el control de la RNA implementado en el prototipo de laboratorio.

Tabla 4.10

COMPARACIÓN DE THD ENTRE EL CONTROL FCS-MPC Y EL CONTROL EN BASE A RNA EN FUNCIÓN DE LA REFERENCIA DE AMPLITUD DE CORRIENTE, PARA PROTOTIPO DE LABORATORIO

$A_{ref}[A]$	2	4	6	8	10
THD FCS-MPC [%]	9.0	5.54	3.9	2.84	3.31
THD RNA [%]	9.16	4.94	3.32	2.61	2.92

En la tabla 4.10 se observa una disminución del THD al aumentar amplitud de la referencia de corriente, para el caso del control PI y el control en base a RNA, repitiendo el comportamiento visto en la tabla 4.6. Además, se observa que el THD es marginalmente menor para el caso del control en basado en RNA.

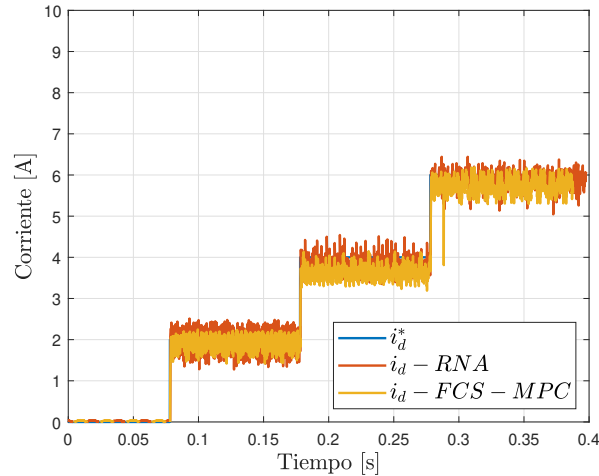


Figura 4.22. Comparación de corriente en el eje directo entre el control FCS-MPC y el control en base a RNA, en prototipo de laboratorio

En la figura 4.22 se muestra la corriente en el eje directo para ambas estrategias de control en el prototipo de laboratorio. En esta figura se observa que se repite el comportamiento observado en el entorno de simulación, donde la magnitud y el comportamiento de la corriente i_d son muy similares.

En la figura 4.23 se muestra el espectro en frecuencia de la corriente medida i_a y la tensión de salida del convertidor en la fase a con respecto al neutro, para las estrategias de control FCS-MPC y el control basado en RNA. Se observa que ambas estrategias presentan un comportamiento en frecuencia similar; el decaimiento y el orden de las armónicas coinciden para ambas variables, lo que respalda la capacidad de la RNA para caracterizar el control FCS-MPC.

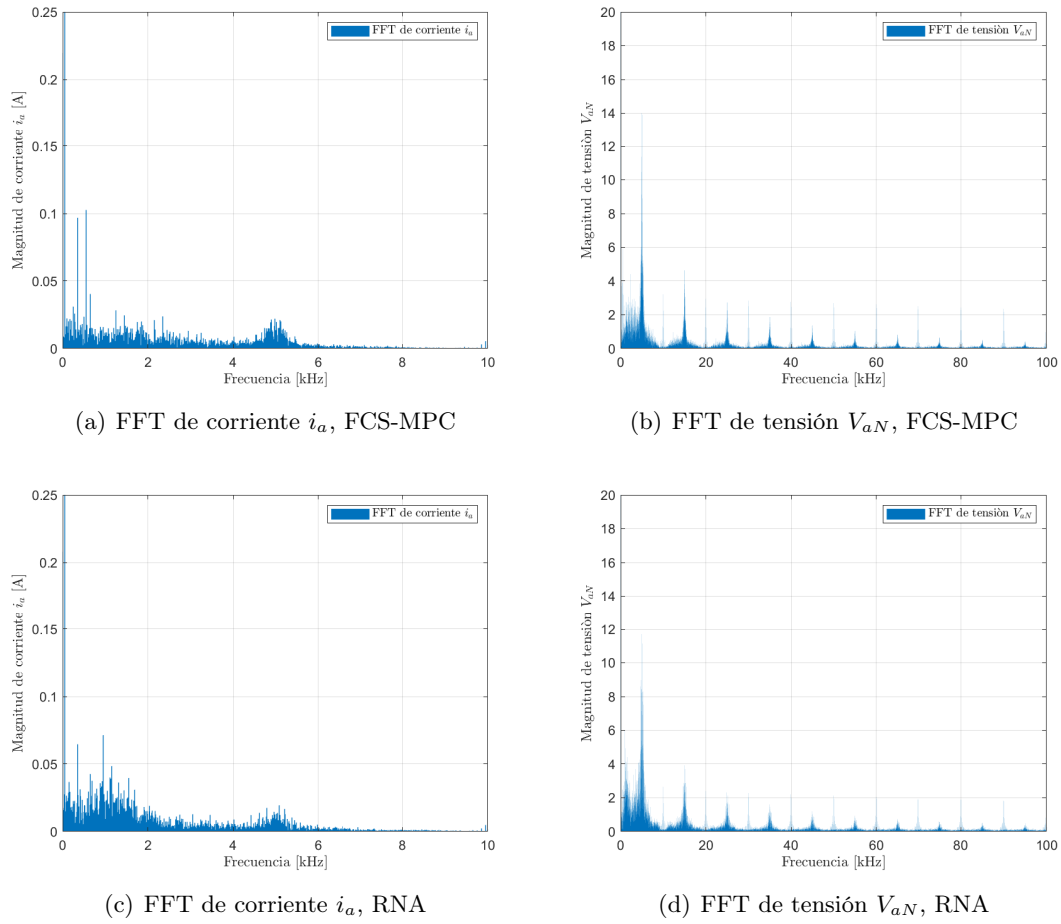
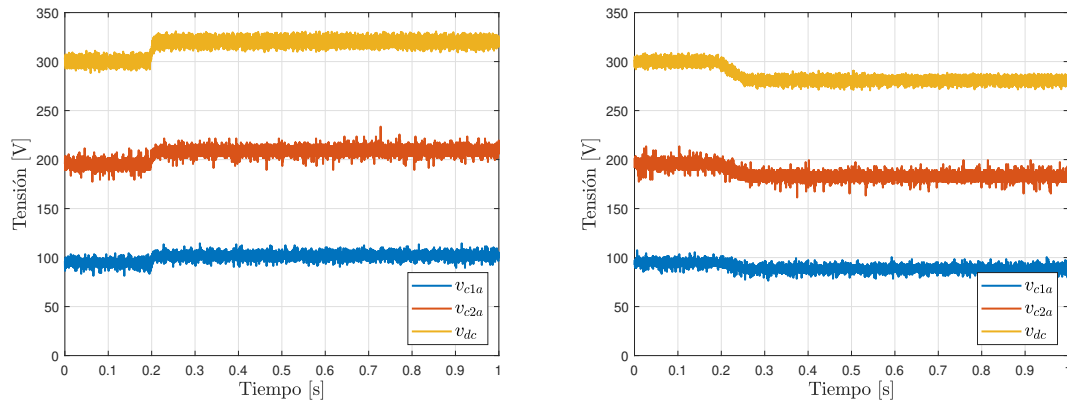
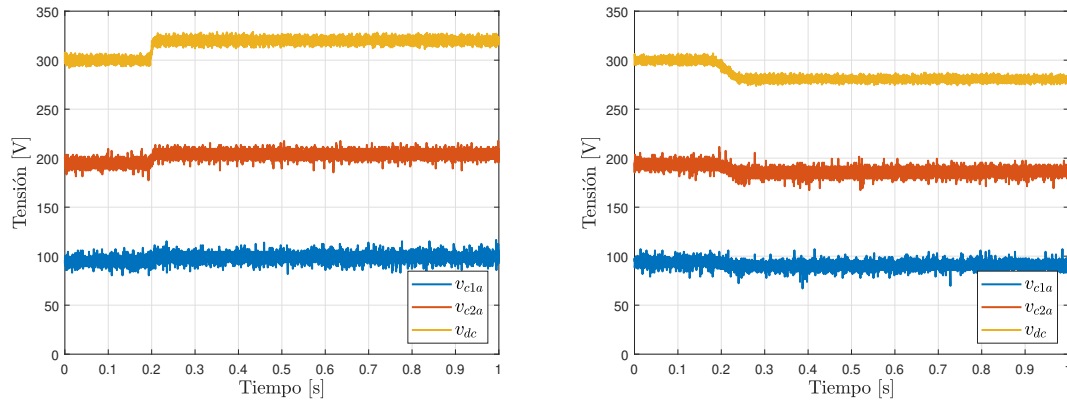


Figura 4.23. FFT de corriente i_a y tensión V_{aN} para el control FCS-MPC y el control en base a RNA

Dado que el control FCS-MPC y el control basado en RNA realizan balance activo de tensión en los condensadores se decide realizar un ensayo a escalón para observar la respuesta. Mediante una fuente controlada de tensión, se configura un escalón positivo $\Delta V = 20[V]$, el cual se aplica al *dc-link* del convertidor. Del mismo modo, se realiza un escalón negativo de magnitud $-\Delta V = -20[V]$. En la figura 4.24 se muestra la respuesta a escalón de tensión del *dc-link* para ambas estrategias de control



(a) Tensiones fase a , escalón de tensión $+\Delta V$, para FCS-MPC (b) Tensiones fase a , escalón de tensión $-\Delta V$, para FCS-MPC



(c) Tensiones fase a , escalón de tensión $+\Delta V$, para RNA (d) Tensiones fase a , escalón de tensión $-\Delta V$, para RNA

Figura 4.24. Respuesta a escalón de tensión para control FCS-MPC y control en base a RNA

Se observa en la figura 4.24 que para ambas estrategias la respuesta es similar, por lo que se puede observar que el control en base a RNA realiza control de tensión con tiempos de respuesta similares al control FCS-MPC.

4.3.5. Observación del costo computacional de los métodos de control en DSP

Realizar una comparativa rigurosa del costo computacional de los métodos de control no es una tarea fácil, ya que, a pesar de utilizar el mismo procesador digital de señales, la velocidad de ejecución de una rutina de control depende de múltiples factores como el compilador, la configuración de las memorias y la estructura del código.

En esta evaluación se procuró tener el código y sus variables en la misma memoria cercana a la DSP, así como una estructura de código relativamente similar. En la tabla 4.11 se muestra el número de ciclos de reloj que le toma a la DSP realizar la rutina de control para cada estrategia.

Tabla 4.11

OBSERVACIÓN DE COSTO COMPUTACIONAL PARA LAS DISTINTAS ESTRATEGIAS DE CONTROL

Estrategia de control	Cantidad de ciclos de reloj	Tiempo [μs]
PI con PS-PWM	5677	18.9
RNA - PI con PS-PWM	10726	35.8
FCS - MPC	21046	70.2
RNA - MPC	23467	78.2

En la tabla 4.11 se observa que la estrategia RNA-PI es mucho más costosa computacionalmente que el control PI. Esto se debe a que el sistema tiene que buscar y multiplicar los pesos y sesgos de cada capa para obtener la salida, en comparación con el control PI que solo requiere un par de multiplicaciones. Recordando que la red neuronal artificial RNA-PI tiene 12 neuronas y la red RNA-MPC tiene 36 neuronas en la capa oculta, se puede notar que el tiempo de cálculo aumenta al incrementar el número de neuronas de la capa oculta, pero no necesariamente de forma proporcional.

Para el caso del costo computacional del control FCS-MPC y RNA-MPC, se observa que el costo del control en base a RNA es mayor que el control predictivo, sin embargo, esta diferencia es relativamente menor. Dado que la adición de una neurona más a la RNA-MPC implica calcular adicionalmente 24 multiplicaciones, una suma y una transformación no lineal, reducir en una neurona la cantidad de neuronas en la capa oculta de la RNA disminuiría la cantidad de cálculos, logrando así un costo computacional menor al del control FCS-MPC. Además, dado que no se realizó un trabajo de optimización exhaustivo del código, es posible concluir que eventualmente se podría obtener una RNA que sea más eficiente computacionalmente que el método de control FCS-MPC.

Capítulo 5

Conclusiones.

En este trabajo de memoria se desarrolló e implementó el control basado en redes neuronales para dos estrategias de control conocidas: el control PI con modulación PS-PWM y el control predictivo FCS-MPC para un convertidor de capacitor flotante de cuatro niveles alimentando una carga RL.

Para el control PI con modulación PS-PWM se utilizó una red neuronal *feed-forward* con 12 neuronas en la capa oculta y los índices de modulación de cada fase como salida. Para el conjunto de datos de entrenamiento se recolectaron las mediciones y variables en simulación de la operación del sistema con diversas consignas de corriente y a distintos valores de resistencia, caracterizando así el control lineal. Una vez entrenada la red, se observó en el entorno de simulación y en el prototipo de laboratorio que la red es capaz de ajustar su actuación acorde a la variación de la consigna de corriente e imitar el comportamiento transitorio del control lineal, pero no es capaz de corregir el error en estado estacionario. En cuanto al desempeño de ambas estrategias, se evaluó mediante el THD, observando que las variaciones de THD para distintas amplitudes de corriente no superan el 1 % de diferencia entre ellas, tanto en simulación como en el prototipo de laboratorio.

Para el caso del control predictivo FCS-MPC se utilizó una red neuronal *feed-forward* con 36 neuronas en la capa oculta y una parametrización de los estados de conmutación de cada celda de cada fase como salida. El conjunto de datos está conformado por las mediciones y variables en simulación de la operación del sistema con diversas consignas de corriente y niveles de tensión en los condensadores, caracterizando así el control de corriente y de tensión realizado por la estrategia. Al implementar el control basado en redes neuronales en el entorno de simulación y en el prototipo de laboratorio, se observó que tanto el comportamiento dinámico como el estacionario son similares en corriente y en tensión. En este aspecto, la red neuronal adoptó de manera similar que el control FCS-MPC las diferencias de parámetros de la carga en el laboratorio. Al evaluar el desempeño mediante el THD, se observó que la mayor diferencia de distorsión entre ambas estrategias es de un 0.6 % para el prototipo de laboratorio.

En cuanto al control basado en redes neuronales, se pudo observar que para relaciones simples como el control PI, la caracterización es más difícil de lograr ya que en el entrenamiento hay una reducción importante del error a la salida de la red al relacionar proporcionalmente la corriente de referencia con el índice de modulación, dejando en segundo plano la acción integral. En este trabajo se intentó darle mayor prioridad a la acción

integral añadiendo más datos con error entre la corriente de referencia, lo cual no condujo a resultados satisfactorios. Por otra parte, cuando la relación entre las mediciones y la acción de control es más compleja, como en el caso de la estrategia FCS-MPC, se vuelve necesario enfocarse tanto en que la información proporcionada a la red sea la necesaria como en que esta información tenga la forma adecuada para evitar que la red tenga que descifrar relaciones no relevantes del preprocesado.

Del desarrollo de este trabajo, se observa que las redes neuronales aplicadas a convertidores pueden ser útiles para reducir el tiempo de cálculo asociado a la iteración de los estados de conmutación en el control FCS-MPC, los cuales aumentan exponencialmente con el número de niveles de un convertidor de capacitor flotante. En cambio, en una RNA, el tiempo de cálculo se ve afectado de manera proporcional solo en la capa de salida, ya que la matriz de pesos y sesgos se recorre una sola vez. Además, la posibilidad de entrenar la red con datos obtenidos en un entorno de simulación permite emplear modelos de control complejos y con altos tiempos de cálculo, logrando una red que generalice el control. Esto es posible gracias a la naturaleza de "caja negra" de la RNA, donde el tiempo de cálculo se relaciona principalmente con la cantidad de neuronas en la red neuronal artificial.

Apéndice A

Códigos de simulación

En este apéndice se presentan parte de los códigos utilizados en PLECS para la estrategia de control PI con modulación PS-PWM, control RNA-PI con modulación PS-PWM, control FCS-MPC y control RNA-MPC.

A.1. Control PI con modulación PS-PWM

```
// def. variables de control
float i1_r=0;
float i2_r=0;
float i3_r=0;
ia_r1=0;
ib_r1=0;
ic_r1=0;

float alfa=0;
float beta=0;
float teta=0;
float tetaf=0;
float Da=0;
float Db=0;
float Dc=0;
float Qa=0;
float Qb=0;
float Qc=0;
float t1=0;
float errorD=0;
float errorQ=0;
float iD_r=0;
float iD_m=0;
float iQ_r=0;
float iQ_m=0;
float xd_t1=0;
ud_t1=0;
float xq_t1=0;
uq_t1=0;
float DET=0;
float DET_1=0;

float mA=0;
```

```

float mB=0;
float mC=0;
/*****
// Referencia corriente
aux=aux+h;
t1=aux-1*h;

i1_r=cos(2*pi*50*aux);
i2_r=cos(2*pi*50*aux-x2_3*pi);
i3_r=cos(2*pi*50*aux+x2_3*pi);

ia_r1=A_r*cos(2*pi*50*t1);
ib_r1=A_r*cos(2*pi*50*t1-x2_3*pi);
ic_r1=A_r*cos(2*pi*50*t1+x2_3*pi);

ia_r=A_r*cos(2*pi*50*aux);
ib_r=A_r*cos(2*pi*50*aux-x2_3*pi);
ic_r=A_r*cos(2*pi*50*aux+x2_3*pi);

/*****
//Calculo de Teta
alfa = i1_r;
beta = (i2_r-i3_r) * raiz1_3;
teta = fabsf(atanf(beta/alfa));

if (alfa<0 && beta>=0)
    tetaf = pi - teta;
else
{
    if (alfa<0 && beta<0)
        tetaf = pi + teta;
    else
    {
        if (alfa>=0 && beta<0)
            tetaf = 2 * pi-teta;
        else
            if (alfa>=0 && beta>=0)
                tetaf = teta;
    }
}

// Transformada ABC-DQ de corriente
Da = cosf(tetaf) * x2_3;
Db = cosf(tetaf-g120) * x2_3;
Dc = cosf(tetaf+g120) * x2_3;

Qa = -sinf(tetaf) * x2_3;
Qb = -sinf(tetaf-g120) * x2_3;
Qc = -sinf(tetaf+g120) * x2_3;

/*****
// Corrientes en eje D y eje Q
iD_m = Da*ia_m + Db*ib_m + Dc*ic_m;
iQ_m = Qa*ia_m + Qb*ib_m + Qc*ic_m;
iD_r = Da*ia_r1 + Db*ib_r1 + Dc*ic_r1;
//iQ_r = Qa*ia_r1 + Qb*ib_r1 + Qc*ic_r1;
iQ_r=0;
/*****/

```

```

id_m = iD_m;
iq_m = iQ_m;
id_r = iD_r;
iq_r = iQ_r;

// Calculo eje D
errorD = iD_r-iD_m;

xd_t1 = (ud_t * n2) - (xd_t * d2);
ud_t1 = (errorD-xd_t1) * kpi;

if(ud_t1>=VDC_2)
    ud_t1 = VDC_2;
else
{
    if(ud_t1<=-VDC_2)
        ud_t1 = -VDC_2;
}
/*****/
// Calculo eje Q
errorQ = iQ_r-iQ_m;

xq_t1 = (uq_t * n2) - (xq_t * d2);
uq_t1 = (errorQ-xq_t1) * kpi;

if(uq_t1>=VDC_2)
    uq_t1 = VDC_2;
else
{
    if(uq_t1<=-VDC_2)
        uq_t1 = -VDC_2;
}
/*****/
// Transformada DQ-ABC de tension
DET = Da * (Qb-Qc) - Db * (Qa-Qc) + Dc * (Qa-Qb);
DET_1 =1/DET;
/*****/
mA = ((Qb-Qc) * ud_t1 - (Db-Dc) * uq_t1) * DET_1 / VDC;
mB = ((Qc-Qa) * ud_t1 + (Da-Dc) * uq_t1) * DET_1 / VDC;
mC = ((Qa-Qb) * ud_t1 - (Da-Db) * uq_t1) * DET_1 / VDC;
/*****/
// Salida de variables
s1_A=mA+0.5;
s2_A=mA+0.5;
s3_A=mA+0.5;
s1_B=mB+0.5;
s2_B=mB+0.5;
s3_B=mB+0.5;
s1_C=mC+0.5;
s2_C=mC+0.5;
s3_C=mC+0.5;

mat = m1at;
mbt = m1bt;
mct = m1ct;
m1at = s1_A;
m1bt = s1_B;
m1ct = s1_C;

```

```

xd_t=xd_t1;
ud_t=ud_t1;
xq_t=xq_t1;
uq_t=uq_t1;

```

A.2. Control RNA-PI con modulación PS-PWM

```

// def. variables de control
float i1_r=0;
float i2_r=0;
float i3_r=0;
float ia_r1=0;
float ib_r1=0;
float ic_r1=0;

float alfa=0;
float beta=0;
float teta=0;
float tetaf=0;
float Da=0;
float Db=0;
float Dc=0;
float Qa=0;
float Qb=0;
float Qc=0;
float t1=0;

float iD_r=0;
float iD_m=0;
float iQ_r=0;
float iQ_m=0;

/*****/
// Referencia corriente
aux=aux+h;
t1=aux-1*h;

i1_r=cos(2*pi*50*aux);
i2_r=cos(2*pi*50*aux-x2_3*pi);
i3_r=cos(2*pi*50*aux+x2_3*pi);

ia_r1=A_r*cos(2*pi*50*t1);
ib_r1=A_r*cos(2*pi*50*t1-x2_3*pi);
ic_r1=A_r*cos(2*pi*50*t1+x2_3*pi);

ia_r=A_r*cos(2*pi*50*aux);
ib_r=A_r*cos(2*pi*50*aux-x2_3*pi);
ic_r=A_r*cos(2*pi*50*aux+x2_3*pi);

/*****/
// Calculo de Teta
alfa = i1_r;
beta = (i2_r-i3_r) * raiz1_3;
teta = fabsf(atanf(beta/alfa));

if (alfa<0 && beta>=0)
    tetaf = pi - teta;

```

```

else
{
    if (alfa<0 && beta<0)
        tetaf = pi + teta;
    else
    {
        if (alfa>=0 && beta<0)
            tetaf = 2 * pi - teta;
        else
            if (alfa>=0 && beta>=0)
                tetaf = teta;
    }
}

// Transformada ABC-DQ de corriente
Da = cosf(tetaf) * x2_3;
Db = cosf(tetaf-g120) * x2_3;
Dc = cosf(tetaf+g120) * x2_3;

Qa = -sinf(tetaf) * x2_3;
Qb = -sinf(tetaf-g120) * x2_3;
Qc = -sinf(tetaf+g120) * x2_3;

/*****
// Corrientes en eje D y eje Q
iD_m = Da*ia_m + Db*ib_m + Dc*ic_m;
iQ_m = Qa*ia_m + Qb*ib_m + Qc*ic_m;
iD_r = Da*ia_r1 + Db*ib_r1 + Dc*ic_r1;
iQ_r = Qa*ia_r1 + Qb*ib_r1 + Qc*ic_r1;
*****/

id_m = iD_m;
iq_m = iQ_m;
id_r = iD_r;
iq_r = iQ_r;

/*****
//ANN

double input[12]={ia_r1, ia_m, ia_r1-ia_m, ib_r1, ib_m, ib_r1-ib_m, ic_r1,
    ic_m, ic_r1-ic_m, m1at, m1bt, m1ct};

double input2[12];
for (int i = 0; i < 12; i++){
    input2[i] = (2/(xmax[i]-xmin[i]))*(input[i] - xmin[i])-1;
}

double outlayer1[12];

for (int i = 0; i < 12; i++) {
    double suma = 0;
    for (int j = 0; j < 12; j++) {
        suma += iw[i][j] * input2[j];
    }
    outlayer1[i] = 2/(1+exp(-2*(suma + ib[i][0])))-1;
}

```

```

double outlayer2[3];
for (int i = 0; i < 3; i++) {
    double suma = 0;
    for (int j = 0; j < 12; j++) {
        suma += ow[i][j] * outlayer1[j];
    }
    outlayer2[i] = suma + ob[i][0];
}
double output[3];
for (int i = 0; i < 3; i++){
    output[i] = ((ymax[i]-ymin[i])/2)*(outlayer2[i] + 1 ) + ymin[i];
}

// Salida de variables

s1_A=output[0];
s2_A=output[0];
s3_A=output[0];
s1_B=output[1];
s2_B=output[1];
s3_B=output[1];
s1_C=output[2];
s2_C=output[2];
s3_C=output[2];

m1at = s1_A;
m1bt = s1_B;
m1ct = s1_C;

```

A.3. Control FCS-MPC

```

// def. variables de salida
act=0;

// def. variables de control
float i1_r=0;
float i2_r=0;
float i3_r=0;
ia_r1=0;
ib_r1=0;
ic_r1=0;

VaN_t=0;
VbN_t=0;
VcN_t=0;
VoN=0;
float ia_t=0;
float ib_t=0;
float ic_t=0;
float ia_t2=0;
float ib_t2=0;
float ic_t2=0;
float vc1a_t=0;
float vc2a_t=0;
float vc1b_t=0;
float vc2b_t=0;
float vc1c_t=0;

```

```

float vc2c_t=0;
float vc1a_t2=0;
float vc2a_t2=0;
float vc1b_t2=0;
float vc2b_t2=0;
float vc1c_t2=0;
float vc2c_t2=0;

int i=0;
float g_min=0;
float g=0;
int S1a=0;
int S2a=0;
int S3a=0;
int S1b=0;
int S2b=0;
int S3b=0;
int S1c=0;
int S2c=0;
int S3c=0;

float alfa=0;
float beta=0;
float teta=0;
float tetaf=0;
float Da=0;
float Db=0;
float Dc=0;
float Qa=0;
float Qb=0;
float Qc=0;
float t1=0;

a_ref = 4;

/*****
// Referencia corriente
t=t+h;
t1=t+2*h;

i1_r=cos(2*pi*50*t);
i2_r=cos(2*pi*50*t-x2_3*pi);
i3_r=cos(2*pi*50*t+x2_3*pi);

ia_r1=a_ref*cos(2*pi*50*t1);
ib_r1=a_ref*cos(2*pi*50*t1-x2_3*pi);
ic_r1=a_ref*cos(2*pi*50*t1+x2_3*pi);

ia_r=a_ref*cos(2*pi*50*t);
ib_r=a_ref*cos(2*pi*50*t-x2_3*pi);
ic_r=a_ref*cos(2*pi*50*t+x2_3*pi);

*****/
// Calculo de Teta
alfa = i1_r;
beta = (i2_r-i3_r) * raiz1_3;
teta = fabsf(atanf(beta/alfa));

```

```

if (alfa<0 && beta>=0)
    tetaf = pi - teta;
else
{
    if (alfa<0 && beta<0)
        tetaf = pi + teta;
    else
    {
        if (alfa>=0 && beta<0)
            tetaf = 2 * pi - teta;
        else
            if (alfa>=0 && beta>=0)
                tetaf = teta;
    }
}

// Transformada ABC-DQ de corriente
Da = cosf(tetaf) * x2_3;
Db = cosf(tetaf-g120) * x2_3;
Dc = cosf(tetaf+g120) * x2_3;

Qa = -sinf(tetaf) * x2_3;
Qb = -sinf(tetaf-g120) * x2_3;
Qc = -sinf(tetaf+g120) * x2_3;

/*****/
// Actualizacion variables k+1
//VoN=(VaN+VbN+VcN)/3;
VoN=150;

ia_t=(VaN-VoN)*k1+ia_m*k2;
ib_t=(VbN-VoN)*k1+ib_m*k2;
ic_t=(VcN-VoN)*k1+ic_m*k2;

vc1a_t=(ia_t+ia_m)*kC1*(s2_A-s1_A)+vc1a_m;
vc2a_t=(ia_t+ia_m)*kC2*(s3_A-s2_A)+vc2a_m;
vc1b_t=(ib_t+ib_m)*kC1*(s2_B-s1_B)+vc1b_m;
vc2b_t=(ib_t+ib_m)*kC2*(s3_B-s2_B)+vc2b_m;
vc1c_t=(ic_t+ic_m)*kC1*(s2_C-s1_C)+vc1c_m;
vc2c_t=(ic_t+ic_m)*kC2*(s3_C-s2_C)+vc2c_m;

/*****/
// Control MPC k+2

g_min=1e20;

for (i=0;i<=contX-1;i=i+1)
{
    VaN_t=VDC*S3A[i]+vc2a_t*(S2A[i]-S3A[i])+vc1a_t*(S1A[i]-S2A[i]);
    VbN_t=VDC*S3B[i]+vc2b_t*(S2B[i]-S3B[i])+vc1b_t*(S1B[i]-S2B[i]);
    VcN_t=VDC*S3C[i]+vc2c_t*(S2C[i]-S3C[i])+vc1c_t*(S1C[i]-S2C[i]);
    //VoN=(VaN_t+VbN_t+VcN_t)/3;
    VoN=150;

    ia_t2=(VaN_t-VoN)*k1+ia_t*k2;
    ib_t2=(VbN_t-VoN)*k1+ib_t*k2;
    ic_t2=(VcN_t-VoN)*k1+ic_t*k2;

    vc1a_t2=(ia_t2+ia_t)*kC1*(S2A[i]-S1A[i])+vc1a_t;

```

```

vc2a_t2=(ia_t2+ia_t)*kC2*(S3A[i]-S2A[i])+vc2a_t;
vc1b_t2=(ib_t2+ib_t)*kC1*(S2B[i]-S1B[i])+vc1b_t;
vc2b_t2=(ib_t2+ib_t)*kC2*(S3B[i]-S2B[i])+vc2b_t;
vc1c_t2=(ic_t2+ic_t)*kC1*(S2C[i]-S1C[i])+vc1c_t;
vc2c_t2=(ic_t2+ic_t)*kC2*(S3C[i]-S2C[i])+vc2c_t;

g=((ia_r1-ia_t2)*(ia_r1-ia_t2))+w1*((vc1_r-vc1a_t2)*(vc1_r-vc1a_t2))+w2
  *((vc2_r-vc2a_t2)*(vc2_r-vc2a_t2))+((ib_r1-ib_t2)*(ib_r1-ib_t2))+w1*((
  vc1_r-vc1b_t2)*(vc1_r-vc1b_t2))+w2*((vc2_r-vc2b_t2)*(vc2_r-vc2b_t2))+((
  ic_r1-ic_t2)*(ic_r1-ic_t2))+w1*((vc1_r-vc1c_t2)*(vc1_r-vc1c_t2))+w2*((
  vc2_r-vc2c_t2)*(vc2_r-vc2c_t2));

if (g<g_min)
{
  g_min=g;
  act=i;
}
}

S1a=S1A[act];
S2a=S2A[act];
S3a=S3A[act];
S1b=S1B[act];
S2b=S2B[act];
S3b=S3B[act];
S1c=S1C[act];
S2c=S2C[act];
S3c=S3C[act];

VaN=VDC*S3a+vc2a_t*(S2a-S3a)+vc1a_t*(S1a-S2a);
VbN=VDC*S3b+vc2b_t*(S2b-S3b)+vc1b_t*(S1b-S2b);
VcN=VDC*S3c+vc2c_t*(S2c-S3c)+vc1c_t*(S1c-S2c);

/*****
// Corrientes en eje D y eje Q
id_m = Da*ia_m + Db*ib_m + Dc*ic_m;
iq_m = Qa*ia_m + Qb*ib_m + Qc*ic_m;
id_r = Da*ia_r + Db*ib_r + Dc*ic_r;
iq_r = Qa*ia_r + Qb*ib_r + Qc*ic_r;
*****/

// Salida de variables
s1_A=S1a;
s2_A=S2a;
s3_A=S3a;
s1_B=S1b;
s2_B=S2b;
s3_B=S3b;
s1_C=S1c;
s2_C=S2c;
s3_C=S3c;

VaN_m=VDC*s3a_1t+vc2a_m*(s2a_1t-s3a_1t)+vc1a_m*(s1a_1t-s2a_1t);
VbN_m=VDC*s3b_1t+vc2b_m*(s2b_1t-s3b_1t)+vc1b_m*(s1b_1t-s2b_1t);
VcN_m=VDC*s3c_1t+vc2c_m*(s2c_1t-s3c_1t)+vc1c_m*(s1c_1t-s2c_1t);

aux = VaN_m-(VaN_m+VbN_m+VcN_m)/3;

s1a_1t = S1a;

```

```

s2a_1t = S2a;
s3a_1t = S3a;
s1b_1t = S1b;
s2b_1t = S2b;
s3b_1t = S3b;
s1c_1t = S1c;
s2c_1t = S2c;
s3c_1t = S3c;

```

A.4. Control RNA-MPC

```

// def. variables de control
ia_r1=0;
ib_r1=0;
ic_r1=0;

int    i=0;
float  t1=0;

float  VaN_m;
float  VbN_m;
float  VcN_m;

a_ref = 4;
/*****
// Referencia corriente
t=t+h;
t1=t+2*h;

ia_r1=a_ref*cos(2*pi*50*t1);
ib_r1=a_ref*cos(2*pi*50*t1-x2_3*pi);
ic_r1=a_ref*cos(2*pi*50*t1+x2_3*pi);

VaN_m=VDC*s3a_1t+vc2a_m*(s2a_1t-s3a_1t)+vc1a_m*(s1a_1t-s2a_1t);
VbN_m=VDC*s3b_1t+vc2b_m*(s2b_1t-s3b_1t)+vc1b_m*(s1b_1t-s2b_1t);
VcN_m=VDC*s3c_1t+vc2c_m*(s2c_1t-s3c_1t)+vc1c_m*(s1c_1t-s2c_1t);

//ANN
int  n_in = 24;
int  n_lyr1 = 36;
int  n_lyr2 = 24;

// ANN 3f

double input[n_in0]={ia_r1, ia_m, vc1a_m, vc2a_m, vc1_r-vc1a_m, vc2_r-
vc2a_m, ia_r1-ia_m, VaN_m, ib_r1, ib_m, vc1b_m, vc2b_m, vc1_r-vc1b_m,
vc2_r-vc2b_m, ib_r1-ib_m, VbN_m, ic_r1, ic_m, vc1c_m, vc2c_m, vc1_r-
vc1c_m, vc2_r-vc2c_m, ic_r1-ic_m, VcN_m};

double input2[n_in];
for (int i = 0; i < n_in; i++){
    input2[i] = (2/(xmax[i]-xmin[i]))*(input[i] - xmin[i])-1;
}

double outlayer1[n_lyr1];

for (int i = 0; i < n_lyr1; i++) {

```

```

double suma = 0;
for (int j = 0; j < n_in; j++) {
    suma += iw[i][j] * input2[j];
}
outlayer1[i] = 2/(1+exp(-2*(suma + ib[i][0]))) - 1;
}

double outlayer2[n_lyr2];
for (int i = 0; i < n_lyr2; i++) {
    double suma = 0;
    for (int j = 0; j < n_lyr1; j++) {
        suma += ow[i][j] * outlayer1[j];
    }
    outlayer2[i] = suma + ob[i][0];
}
double output[n_lyr2];
for (int i = 0; i < n_lyr2; i++){
    output[i] = 1/(1+exp(-outlayer2[i]));
}

double SSmax_a;
int indSSmax_a;

SSmax_a = output[0];

for (i=0; i<8; i++){
    if (output[i] > SSmax_a){
        SSmax_a = output[i];
        indSSmax_a = i;
    }
}

double SSmax_b;
int indSSmax_b;

SSmax_b = output[8];

for (i=8; i<16; i++){
    if (output[i] > SSmax_b){
        SSmax_b = output[i];
        indSSmax_b = i-8;
    }
}

double SSmax_c;
int indSSmax_c;

SSmax_c = output[16];

for (i=16; i<24; i++){
    if (output[i] > SSmax_c){
        SSmax_c = output[i];
        indSSmax_c = i-16;
    }
}

// Salida de variables

s1_A=SS1[indSSmax_a];

```

```
s2_A=SS2[indSSmax_a];  
s3_A=SS3[indSSmax_a];  
s1_B=SS1[indSSmax_b];  
s2_B=SS2[indSSmax_b];  
s3_B=SS3[indSSmax_b];  
s1_C=SS1[indSSmax_c];  
s2_C=SS2[indSSmax_c];  
s3_C=SS3[indSSmax_c];
```

```
s1a_1t = s1_A;  
s2a_1t = s2_A;  
s3a_1t = s3_A;  
s1b_1t = s1_B;  
s2b_1t = s2_B;  
s3b_1t = s3_B;  
s1c_1t = s1_C;  
s2c_1t = s2_C;  
s3c_1t = s3_C;
```

```
//  
aux = a_ref;  
van = VaN_m;  
vbn = VbN_m;  
vcn = VcN_m;
```

Apéndice B

Códigos de la Implementación

En este apéndice se presentan parte de los códigos utilizados en Code Composer Studio para la estrategia de control PI con modulación PS-PWM, control RNA-PI con modulación PS-PWM, control FCS-MPC y control RNA-MPC.

B.1. Control PI con modulación PS-PWM

```
/*
    ****
**      HEADER FILES
****
*/
/* HW Macros */

/* System Defines */
#include "aceC6748.h"
#include "assert.h"
#include "math.h"
#include "psc.h"
#include "hw_psc_C6748.h"
#include "ace_monitor.h"

#include <param_v3.h>
#include <PWM_app.h>
// #include "coss.h"

/*
 * Monitor.
 */
#define SYSCLK2_FREQ      (150000000u)
#define HMI_UART_BAUDRATE (115200u)
#define HMI_UART_TIMEOUT  (20000)

#define array_length 4000

bool tick_flag = 0;
#pragma DATA_SECTION (array0_A1, ".ddr")
#pragma DATA_SECTION (array0_A2, ".ddr")
#pragma DATA_SECTION (array0_B1, ".ddr")
```

```
#pragma DATA_SECTION (array0_B2, ".ddr")
#pragma DATA_SECTION (array1_A1, ".ddr")
#pragma DATA_SECTION (array1_A2, ".ddr")
#pragma DATA_SECTION (array1_B1, ".ddr")
#pragma DATA_SECTION (array1_B2, ".ddr")
#pragma DATA_SECTION (array2_A1, ".ddr")
#pragma DATA_SECTION (array2_A2, ".ddr")
#pragma DATA_SECTION (array2_B1, ".ddr")
#pragma DATA_SECTION (array2_B2, ".ddr")
#pragma DATA_SECTION (array3_A1, ".ddr")
#pragma DATA_SECTION (array3_A2, ".ddr")

extern void Init_Platform(void);
extern void Init_Interrupts(void);
extern void Init_Adc(void);
extern void Init_Iop(void);
extern void Init_Mod(void);
extern void Init_DS_LED(void);

double aux = 0;
float a_ref = 2;

float ia_r, ib_r, ic_r;
float ia_r1, ib_r1, ic_r1;
float i1_r, i2_r, i3_r;
float ia_m;
float ib_m;
float ic_m;
float vc1a_m;
float vc2a_m;
float vc1b_m;
float vc2b_m;
float vc1c_m;
float vc2c_m;

float m1_A = 0;
float m2_A = 0;
float m3_A = 0;
float m1_B = 0;
float m2_B = 0;
float m3_B = 0;
float m1_C = 0;
float m2_C = 0;
float m3_C = 0;

float ud_t = 0;
float uq_t = 0;
float xd_t = 0;
float xq_t = 0;

float alfa=0;
float beta=0;
float teta=0;
float tetaf=0;
float Da=0;
float Db=0;
float Dc=0;
float Qa=0;
float Qb=0;
```

```
float Qc=0;
double t1=0;
float errorD=0;
float errorQ=0;
float iD_r=0;
float iD_m=0;
float iQ_r=0;
float iQ_m=0;
float xd_t1=0;
float ud_t1=0;
float xq_t1=0;
float uq_t1=0;
float DET=0;
float DET_1=0;

float mA=0;
float mB=0;
float mC=0;

float gain_i = -2.51291866;
float offset_i = 0;
float gain_ic = -4.954716981;
float offset_ic = 0;

float gain_v1 = -39.86710963;
float offset_v1 = 0;
float gain_v2 = -99.45355191;
float offset_v2 = 0;

int16_t Med_ADC0_A1_16;
int16_t Med_ADC0_A2_16;
int16_t Med_ADC0_B1_16;
int16_t Med_ADC0_B2_16;
float array0_A1[array_length];
float array0_A2[array_length];
float array0_B1[array_length];
float array0_B2[array_length];

int16_t Med_ADC1_A1_16;
int16_t Med_ADC1_A2_16;
int16_t Med_ADC1_B1_16;
int16_t Med_ADC1_B2_16;
float array1_A1[array_length];
float array1_A2[array_length];
float array1_B1[array_length];
float array1_B2[array_length];

int16_t Med_ADC2_A1_16;
int16_t Med_ADC2_A2_16;
float array2_A1[array_length];
float array2_A2[array_length];
float array2_B1[array_length];
float array2_B2[array_length];

float array3_A1[array_length];
float array3_A2[array_length];

float auxiliar;
float leer = 0;
```

```

float Vdc;

int array_i = 0;

void main(void)
{
    /* All peripherals to be uses are configured here */

    /* Interruptions must be configured after peripherals configuration */
    Init_Platform();      // Configuración de PLL del DSP y de comunicación
                           con la FPGA
    Init_Mod();           // Configuración de los Timers y moduladores PWM
    Init_Adc();           // Configuración de los ADCs
    Init_Iop();           // Configuración de los puertos IO
    Init_DS_LED();        // Configuración de DipSwitch y LEDs
    Init_Interrupts();    // Configuración de las interrupciones. Debe ser
                           la última función a llamar

    ACE_USRIO_turnOffLd(ACE_USRIO_LD0);
    ACE_USRIO_turnOffLd(ACE_USRIO_LD1);
    ACE_USRIO_turnOffLd(ACE_USRIO_LD2);
    ACE_USRIO_turnOffLd(ACE_USRIO_LD3);

    /*
    -----
    */
    /* BACKGROUND ROUTINE
    */
    /*
    -----
    */
    /* LOCAL MONITOR CONFIGURATION */
    /* UART configuration */
    PSCModuleControl(SOC_PSC_1_REGS, HW_PSC_UART2, PSC_POWERDOMAIN_ALWAYS_ON,
    PSC_MDCTL_NEXT_ENABLE);
    ACE_DSPUART_PinMuxSetup(SOC_UART_2, FALSE);

    ACE_MON_init(SOC_UART_2_REGS, SYSCLK2_FREQ, HMI_UART_BAUDRATE,
    HMI_UART_TIMEOUT);
    ACE_MON_addVariableToMonitor(&ia_m, ACE_MON_FLOAT, sizeof(ia_m),
    ACE_MON_ACCESS_READONLY, array0_A1, sizeof(array0_A1));
    ACE_MON_addVariableToMonitor(&ib_m, ACE_MON_FLOAT, sizeof(ib_m),
    ACE_MON_ACCESS_READONLY, array0_A2, sizeof(array0_A2));
    ACE_MON_addVariableToMonitor(&ic_m, ACE_MON_FLOAT, sizeof(ic_m),
    ACE_MON_ACCESS_READONLY, array0_B1, sizeof(array0_B1));

    ACE_MON_addVariableToMonitor(&vc1a_m, ACE_MON_FLOAT, sizeof(vc1a_m),
    ACE_MON_ACCESS_READONLY, array0_B2, sizeof(array0_B2));
    ACE_MON_addVariableToMonitor(&vc2a_m, ACE_MON_FLOAT, sizeof(vc2a_m),
    ACE_MON_ACCESS_READONLY, array1_A1, sizeof(array1_A1));

    ACE_MON_addVariableToMonitor(&vc1b_m, ACE_MON_FLOAT, sizeof(vc1b_m),
    ACE_MON_ACCESS_READONLY, array1_A2, sizeof(array1_A2));
    ACE_MON_addVariableToMonitor(&vc2b_m, ACE_MON_FLOAT, sizeof(vc2b_m),
    ACE_MON_ACCESS_READONLY, array1_B1, sizeof(array1_B1));

    ACE_MON_addVariableToMonitor(&vc1c_m, ACE_MON_FLOAT, sizeof(vc1c_m),
    ACE_MON_ACCESS_READONLY, array1_B2, sizeof(array1_B2));

```

```

    ACE_MON_addVariableToMonitor(&vc2c_m, ACE_MON_FLOAT,    sizeof(vc2c_m),
    ACE_MON_ACCESS_READONLY, array2_A1, sizeof(array2_A1));

    ACE_MON_addVariableToMonitor(&ia_r1, ACE_MON_FLOAT,    sizeof(ia_r1),
    ACE_MON_ACCESS_READONLY, array2_A2, sizeof(array2_A2));
    ACE_MON_addVariableToMonitor(&iD_m, ACE_MON_FLOAT,    sizeof(iD_m),
    ACE_MON_ACCESS_READONLY, array2_B1, sizeof(array2_B1));
    ACE_MON_addVariableToMonitor(&iQ_m, ACE_MON_FLOAT,    sizeof(iQ_m),
    ACE_MON_ACCESS_READONLY, array2_B2, sizeof(array2_B2));

    ACE_MON_addVariableToMonitor(&Vdc, ACE_MON_FLOAT,    sizeof(Vdc),
    ACE_MON_ACCESS_READONLY, array3_A1, sizeof(array3_A1));

    ACE_MON_addVariableToMonitor(&auxiliar, ACE_MON_FLOAT,    sizeof(
    auxiliar), ACE_MON_ACCESS_READONLY, array3_A2, sizeof(array3_A2));
    //ACE_MON_addVariableToMonitor(&VcN_m, ACE_MON_FLOAT,    sizeof(VcN_m),
    ACE_MON_ACCESS_READONLY, array3_B1, sizeof(array3_B1));

    ACE_MON_addVariableToMonitor(&leer, ACE_MON_FLOAT,    sizeof(leer),
    ACE_MON_ACCESS_WRITABLE, ACE_MON_NULL, 0);
    ACE_MON_addVariableToMonitor(&a_ref, ACE_MON_FLOAT,    sizeof(a_ref),
    ACE_MON_ACCESS_WRITABLE, ACE_MON_NULL, 0);

    while(1)
    {
        ACE_MON_tickBackground();
    }
}
/*
*****
*/
extern void ADC_fpgaSyncInterrupt_isr(void){
//int var_temp_int;
/* Clear flag that caused the interruption */
INTOCLRFLG = INTO_ADC0_MASK; // Borra el Flag sobrescribiendo un 1
/*
-----
*/
aux=aux+h;
t1=aux-1*h;

i1_r=cos(2*pi*50*aux);
i2_r=cos(2*pi*50*aux-x2_3*pi);
i3_r=cos(2*pi*50*aux+x2_3*pi);

ia_r1=a_ref*cos(2*pi*50*t1);
ib_r1=a_ref*cos(2*pi*50*t1-x2_3*pi);
ic_r1=a_ref*cos(2*pi*50*t1+x2_3*pi);

ia_r=a_ref*cos(2*pi*50*aux);
ib_r=a_ref*cos(2*pi*50*aux-x2_3*pi);
ic_r=a_ref*cos(2*pi*50*aux+x2_3*pi);

Med_ADC0_A1_16 = ADC0CHA1;
Med_ADC0_A2_16 = ADC0CHA2;
Med_ADC0_B1_16 = ADC0CHB1;
Med_ADC0_B2_16 = ADC0CHB2;

```

```

    ia_m = ((int)Med_ADC0_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
    gain_i + offset_i;
    ib_m = ((int)Med_ADC0_A2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
    gain_i + offset_i;
    ic_m = ((int)Med_ADC0_B1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
    gain_ic + offset_ic;
    vc1a_m = ((int)Med_ADC0_B2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
    gain_v1 + offset_v1;

    array0_A1[array_i] = ia_m;
    array0_A2[array_i] = ib_m;
    array0_B1[array_i] = ic_m;
    array0_B2[array_i] = vc1a_m;

    Med_ADC1_A1_16 = ADC1CHA1;
    Med_ADC1_A2_16 = ADC1CHA2;
    Med_ADC1_B1_16 = ADC1CHB1;
    Med_ADC1_B2_16 = ADC1CHB2;

    vc2a_m = ((int)Med_ADC1_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
    gain_v2 + offset_v2;
    vc1b_m = ((int)Med_ADC1_A2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
    gain_v1 + offset_v1;
    vc2b_m = ((int)Med_ADC1_B1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
    gain_v2 + offset_v2;
    vc1c_m = ((int)Med_ADC1_B2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
    gain_v1 + offset_v1;

    array1_A1[array_i] = vc2a_m;
    array1_A2[array_i] = vc1b_m;
    array1_B1[array_i] = vc2b_m;
    array1_B2[array_i] = vc1c_m;

    Med_ADC2_A1_16 = ADC2CHA1;
    Med_ADC2_A2_16 = ADC2CHA2;

    vc2c_m = ((int)Med_ADC2_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
    gain_v2 + offset_v2;
    //Vdc = ((int)Med_ADC2_A2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
    gain_v2 + offset_v2;
    Vdc = ((int)Med_ADC2_A2_16*ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*-132;

    auxiliar = a_ref;
    tick_flag = ACE_MON_tickForeground();
    if (leer==1){
        array0_A1[array_i] = ia_m;
        array0_A2[array_i] = ib_m;
        array0_B1[array_i] = ic_m;
        array0_B2[array_i] = vc1a_m;
        array1_A1[array_i] = vc2a_m;
        array1_A2[array_i] = vc1b_m;
        array1_B1[array_i] = vc2b_m;
        array1_B2[array_i] = vc1c_m;
        array2_A1[array_i] = vc2c_m;
        array2_A2[array_i] = ia_r1;
        array2_B1[array_i] = iD_m;
        array2_B2[array_i] = iQ_m;
        array3_A1[array_i] = Vdc;
        array3_A2[array_i] = auxiliar;
    }

```

```

//array3_B1[array_i] = VcN_m;
//if (array_i==array_length/4) a_ref = a_ref +2;
//if (array_i==array_length*2/4) a_ref = a_ref +2;
//if (array_i==array_length*3/4) a_ref = a_ref +2;

array_i = array_i+1;

if (array_i>= array_length){
    array_i = 0;
    leer=0;
}

// Calculo de Teta
alfa = i1_r;
beta = (i2_r-i3_r) * raiz1_3;
teta = fabsf(atanf(beta/alfa));

if (alfa<0 && beta>=0)
    tetaf = pi - teta;
else
{
    if (alfa<0 && beta<0)
        tetaf = pi + teta;
    else
    {
        if (alfa>=0 && beta<0)
            tetaf = 2 * pi-teta;
        else
            if (alfa>=0 && beta>=0)
                tetaf = teta;
    }
}

// Transformada ABC-DQ de corriente
Da = cosf(tetaf) * x2_3;
Db = cosf(tetaf-g120) * x2_3;
Dc = cosf(tetaf+g120) * x2_3;

Qa = -sinf(tetaf) * x2_3;
Qb = -sinf(tetaf-g120) * x2_3;
Qc = -sinf(tetaf+g120) * x2_3;

/*****/
// Corrientes en eje D y eje Q
iD_m = Da*ia_m + Db*ib_m + Dc*ic_m;
iQ_m = Qa*ia_m + Qb*ib_m + Qc*ic_m;
iD_r = Da*ia_r1 + Db*ib_r1 + Dc*ic_r1;
iQ_r = Qa*ia_r1 + Qb*ib_r1 + Qc*ic_r1;
//iQ_r = 0;
/*****/
// Calculo eje D
errorD = iD_r-iD_m;

xd_t1 = (ud_t * n2) - (xd_t * d2);
ud_t1 = (errorD-xd_t1) * kpi;

if(ud_t1>=VDC_2)
    ud_t1 = VDC_2;

```

```

else
{
    if(ud_t1<=-VDC_2)
        ud_t1 = -VDC_2;
}
/*****/
// Calculo eje Q
errorQ = iQ_r-iQ_m;

xq_t1 = (uq_t * n2) - (xq_t * d2);
uq_t1 = (errorQ-xq_t1) * kpi;

if(uq_t1>=VDC_2)
    uq_t1 = VDC_2;
else
{
    if(uq_t1<=-VDC_2)
        uq_t1 = -VDC_2;
}
/*****/
// Transformada DQ-ABC de tension
DET = Da * (Qb-Qc) - Db * (Qa-Qc) + Dc * (Qa-Qb);
DET_1 =1/DET;
/*****/
mA = ((Qb-Qc) * ud_t1 - (Db-Dc) * uq_t1) * DET_1 / VDC;
mB = ((Qc-Qa) * ud_t1 + (Da-Dc) * uq_t1) * DET_1 / VDC;
mC = ((Qa-Qb) * ud_t1 - (Da-Db) * uq_t1) * DET_1 / VDC;
/*****/

if(mA>0.5)
    mA = 0.5;
else
{
    if(mA<-0.5)
        mA = -0.5;
}
if(mB>0.5)
    mB = 0.5;
else
{
    if(mB<-0.5)
        mB = -0.5;
}
if(mC>0.5)
    mC = 0.5;
else
{
    if(mC<-0.5)
        mC = -0.5;
}

// Salida de variables
m1_A=(mA+0.5)*MO_MAX;
m2_A=(mA+0.5)*MO_MAX;
m3_A=(mA+0.5)*MO_MAX;
m1_B=(mB+0.5)*MO_MAX;
m2_B=(mB+0.5)*MO_MAX;
m3_B=(mB+0.5)*MO_MAX;
m1_C=(mC+0.5)*MO_MAX;

```

```

    m2_C=(mC+0.5)*MO_MAX;
    m3_C=(mC+0.5)*MO_MAX;
    xd_t=xd_t1;
    ud_t=ud_t1;
    xq_t=xq_t1;
    uq_t=uq_t1;
    //var_temp_int = (int)((((Med_ADC_A1 + 10)/20) * MO_MAX));

    MOCMPR1 = m1_A;
    MOCMPR2 = m1_B;
    MOCMPR3 = m1_C;

    M1CMPR1 = m2_A;
    M1CMPR2 = m2_B;
    M1CMPR3 = m2_C;

    M2CMPR1 = m3_A;
    M2CMPR2 = m3_B;
    M2CMPR3 = m3_C;

    M3CMPR1 = 0.5*MO_MAX;
}

/*
*****
*/

```

B.2. Control RNA-PI con modulación PS-PWM

```

/*
*****

**      HEADER FILES
*****
*****
*****
*****
*****3*****
*/
/* HW Macros */

/* System Defines */
#include "ANN_app.h"
#include "aceC6748.h"
#include "assert.h"
#include "math.h"
#include "psc.h"
#include "hw_psc_C6748.h"
#include "ace_monitor.h"
#include "soc_C6748.h"

#include "coss.p.h"
#include "tanhsp.h"
#include <
    pesosbias_ts_multiref_pwm_operacion_12n_indmod_PI_iabc_mt_1_6k_R151618.
h>

```

```

#include <param_v3.h>

/*
 * Monitor.
 */
#define SYSCLK2_FREQ      (150000000u)
#define HMI_UART_BAUDRATE (115200u)
#define HMI_UART_TIMEOUT  (20000)

bool tick_flag = 0;
#pragma DATA_SECTION (array0_A1, ".ddr")
#pragma DATA_SECTION (array0_A2, ".ddr")
#pragma DATA_SECTION (array0_B1, ".ddr")
#pragma DATA_SECTION (array0_B2, ".ddr")
#pragma DATA_SECTION (array1_A1, ".ddr")
#pragma DATA_SECTION (array1_A2, ".ddr")
#pragma DATA_SECTION (array1_B1, ".ddr")
#pragma DATA_SECTION (array1_B2, ".ddr")
#pragma DATA_SECTION (array2_A1, ".ddr")
#pragma DATA_SECTION (array2_A2, ".ddr")
#pragma DATA_SECTION (array2_B1, ".ddr")
#pragma DATA_SECTION (array2_B2, ".ddr")
#pragma DATA_SECTION (array3_A1, ".ddr")
#pragma DATA_SECTION (array3_A2, ".ddr")

extern void Init_Platform(void);
extern void Init_Interrupts(void);
extern void Init_Adc(void);
extern void Init_Iop(void);
extern void Init_Mod(void);
extern void Init_DS_LED(void);

#define array_length 4000

float m1_A = 0;
float m2_A = 0;
float m3_A = 0;
float m1_B = 0;
float m2_B = 0;
float m3_B = 0;
float m1_C = 0;
float m2_C = 0;
float m3_C = 0;

float a_ref = 2;
float ia_r1, ib_r1, ic_r1;
float ia_m;
float ib_m;
float ic_m;
float vc1a_m;
float vc2a_m;
float vc1b_m;
float vc2b_m;
float vc1c_m;
float vc2c_m;

float alfa;
float beta;
float Da, Db, Dc;

```

```

float Qa, Qb, Qc;
float id_m, id_r;
float iq_m, iq_r;
float teta;
float tetaf;

double t1;
double aux;
float ia_r, ib_r, ic_r;
float i1_r, i2_r, i3_r;
float m1at = 0;
float m1bt = 0;
float m1ct = 0;
//float uq_t1 = 0;
float DET=0;
float DET_1=0;

float mA=0;
float mB=0;
float mC=0;

float gain_i = -2.51291866;
float offset_i = 0;
float gain_ic = -4.954716981;
float offset_ic = 0;

float gain_v1 = -39.86710963;
float offset_v1 = 0;
float gain_v2 = -99.45355191;
float offset_v2 = 0;

int16_t Med_ADC0_A1_16;
int16_t Med_ADC0_A2_16;
int16_t Med_ADC0_B1_16;
int16_t Med_ADC0_B2_16;
float array0_A1[array_length];
float array0_A2[array_length];
float array0_B1[array_length];
float array0_B2[array_length];

int16_t Med_ADC1_A1_16;
int16_t Med_ADC1_A2_16;
int16_t Med_ADC1_B1_16;
int16_t Med_ADC1_B2_16;
float array1_A1[array_length];
float array1_A2[array_length];
float array1_B1[array_length];
float array1_B2[array_length];

int16_t Med_ADC2_A1_16;
int16_t Med_ADC2_A2_16;

float array2_A1[array_length];
float array2_A2[array_length];
float array2_B1[array_length];
float array2_B2[array_length];

float array3_A1[array_length];
float array3_A2[array_length];

```

```

int array_i = 0;
float leer = 0;
double t=0;

int i=0;
int j=0;

//ANN
int n_in = 12;
int n_lyr1 = 12;
int n_lyr2 = 3;

float input[12];
float input2[12];
float outlayer1[12];
float outlayer2[3];
float output[3];

float auxiliar;
float Vdc;

void main(void)
{
    /* All peripherals to be uses are configured here */

    /* Interruptions must be configured after peripherals configuration */
    Init_Platform();      // Configuracion de PLL del DSP y de comunicacion
                           con la FPGA
    Init_Mod();           // Configuracion de los Timers y moduladores PWM
    Init_Adc();           // Configuracion de los ADCs
    Init_Iop();           // Configuracion de los puertos IO
    Init_DS_LED();        // Configuracion de DipSwitch y LEDs

    Init_Interrupts();    // Configuracion de las interrupciones. Debe ser
                           la ultima funcion a llamar
    /*
    -----
    */
    /* BACKGROUND ROUTINE
       */
    /*
    -----
    */
    /* LOCAL MONITOR CONFIGURATION */
    /* UART configuration */
    PSCModuleControl(SOC_PSC_1_REGS, HW_PSC_UART2, PSC_POWERDOMAIN_ALWAYS_ON,
PSC_MDCTL_NEXT_ENABLE);
    ACE_DSPUART_PinMuxSetup(SOC_UART_2, FALSE);

    ACE_MON_init(SOC_UART_2_REGS, SYSCLK2_FREQ, HMI_UART_BAUDRATE,
HMI_UART_TIMEOUT);
    ACE_MON_addVariableToMonitor(&ia_m, ACE_MON_FLOAT, sizeof(ia_m),
ACE_MON_ACCESS_READONLY, array0_A1, sizeof(array0_A1));
    ACE_MON_addVariableToMonitor(&ib_m, ACE_MON_FLOAT, sizeof(ib_m),
ACE_MON_ACCESS_READONLY, array0_A2, sizeof(array0_A2));
    ACE_MON_addVariableToMonitor(&ic_m, ACE_MON_FLOAT, sizeof(ic_m),
ACE_MON_ACCESS_READONLY, array0_B1, sizeof(array0_B1));

```

```

    ACE_MON_addVariableToMonitor(&vc1a_m, ACE_MON_FLOAT,    sizeof(vc1a_m),
    ACE_MON_ACCESS_READONLY, array0_B2, sizeof(array0_B2));
    ACE_MON_addVariableToMonitor(&vc2a_m, ACE_MON_FLOAT,    sizeof(vc2a_m),
    ACE_MON_ACCESS_READONLY, array1_A1, sizeof(array1_A1));

    ACE_MON_addVariableToMonitor(&vc1b_m, ACE_MON_FLOAT,    sizeof(vc1b_m),
    ACE_MON_ACCESS_READONLY, array1_A2, sizeof(array1_A2));
    ACE_MON_addVariableToMonitor(&vc2b_m, ACE_MON_FLOAT,    sizeof(vc2b_m),
    ACE_MON_ACCESS_READONLY, array1_B1, sizeof(array1_B1));

    ACE_MON_addVariableToMonitor(&vc1c_m, ACE_MON_FLOAT,    sizeof(vc1c_m),
    ACE_MON_ACCESS_READONLY, array1_B2, sizeof(array1_B2));
    ACE_MON_addVariableToMonitor(&vc2c_m, ACE_MON_FLOAT,    sizeof(vc2c_m),
    ACE_MON_ACCESS_READONLY, array2_A1, sizeof(array2_A1));

    ACE_MON_addVariableToMonitor(&ia_r1, ACE_MON_FLOAT,    sizeof(ia_r1),
    ACE_MON_ACCESS_READONLY, array2_A2, sizeof(array2_A2));
    ACE_MON_addVariableToMonitor(&id_m, ACE_MON_FLOAT,    sizeof(id_m),
    ACE_MON_ACCESS_READONLY, array2_B1, sizeof(array2_B1));
    ACE_MON_addVariableToMonitor(&iq_m, ACE_MON_FLOAT,    sizeof(iq_m),
    ACE_MON_ACCESS_READONLY, array2_B2, sizeof(array2_B2));

    ACE_MON_addVariableToMonitor(&Vdc, ACE_MON_FLOAT,    sizeof(Vdc),
    ACE_MON_ACCESS_READONLY, array3_A1, sizeof(array3_A1));

    ACE_MON_addVariableToMonitor(&auxiliar, ACE_MON_FLOAT,    sizeof(
    auxiliar), ACE_MON_ACCESS_READONLY, array3_A2, sizeof(array3_A2));
    //ACE_MON_addVariableToMonitor(&VcN_m, ACE_MON_FLOAT,    sizeof(VcN_m),
    ACE_MON_ACCESS_READONLY, array3_B1, sizeof(array3_B1));

    ACE_MON_addVariableToMonitor(&leer, ACE_MON_FLOAT,    sizeof(leer),
    ACE_MON_ACCESS_WRITABLE, ACE_MON_NULL, 0);
    ACE_MON_addVariableToMonitor(&a_ref, ACE_MON_FLOAT,    sizeof(a_ref),
    ACE_MON_ACCESS_WRITABLE, ACE_MON_NULL, 0);
    while(1)
    {
        ACE_MON_tickBackground();
    }
}
/*
*****
*/
extern void ADC_fpgaSyncInterrupt_isr(void){
    //int var_temp_int;
    /* Clear flag that caused the interruption */
    INTOCLRFLG = INTO_ADC0_MASK; // Borra el Flag sobrescribiendo un 1
    /*
    -----
    */
    aux=aux+h;
    t1=aux-h;

    i1_r=coss(PI100*aux);
    i2_r=coss(PI100*aux-x2_3*PI);
    i3_r=coss(PI100*aux+x2_3*PI);

    ia_r1 = a_ref*coss(PI100*t1);
    ib_r1 = a_ref*coss(PI100*t1 - x2_3*PI);

```

```

ic_r1 = a_ref*cossp(PI100*t1 + x2_3*PI);

ia_r=a_ref*cossp(PI100*aux);
ib_r=a_ref*cossp(PI100*aux-x2_3*PI);
ic_r=a_ref*cossp(PI100*aux+x2_3*PI);

Med_ADC0_A1_16 = ADC0CHA1;
Med_ADC0_A2_16 = ADC0CHA2;
Med_ADC0_B1_16 = ADC0CHB1;
Med_ADC0_B2_16 = ADC0CHB2;

ia_m = ((int)Med_ADC0_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_i + offset_i;
ib_m = ((int)Med_ADC0_A2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_i + offset_i;
ic_m = ((int)Med_ADC0_B1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_ic + offset_ic;
vc1a_m = ((int)Med_ADC0_B2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v1 + offset_v1;

Med_ADC1_A1_16 = ADC1CHA1;
Med_ADC1_A2_16 = ADC1CHA2;
Med_ADC1_B1_16 = ADC1CHB1;
Med_ADC1_B2_16 = ADC1CHB2;

vc2a_m = ((int)Med_ADC1_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v2 + offset_v2;
vc1b_m = ((int)Med_ADC1_A2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v1 + offset_v1;
vc2b_m = ((int)Med_ADC1_B1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v2 + offset_v2;
vc1c_m = ((int)Med_ADC1_B2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v1 + offset_v1;

Med_ADC2_A1_16 = ADC2CHA1;
Med_ADC2_A2_16 = ADC2CHA2;

vc2c_m = ((int)Med_ADC2_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v2 + offset_v2;
Vdc = ((int)Med_ADC2_A2_16*ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*-132;

tick_flag = ACE_MON_tickForeground();
if (leer==1){
    array0_A1[array_i] = ia_m;
    array0_A2[array_i] = ib_m;
    array0_B1[array_i] = ic_m;
    array0_B2[array_i] = vc1a_m;
    array1_A1[array_i] = vc2a_m;
    array1_A2[array_i] = vc1b_m;
    array1_B1[array_i] = vc2b_m;
    array1_B2[array_i] = vc1c_m;
    array2_A1[array_i] = vc2c_m;
    array2_A2[array_i] = ia_r1;
    array2_B1[array_i] = id_m;
    array2_B2[array_i] = iq_m;
    array3_A1[array_i] = Vdc;
    array3_A2[array_i] = auxiliar;
    //array3_B1[array_i] = VcN_m;
    //if (array_i==array_length/4) a_ref = a_ref +2;

```

```

//if (array_i==array_length*2/4) a_ref = a_ref +2;
//if (array_i==array_length*3/4) a_ref = a_ref +2;

array_i = array_i+1;

if (array_i>= array_length){
    array_i = 0;
    leer=0;
}
}

// Calculo de Teta
alfa = i1_r;
beta = (i2_r-i3_r) * raiz1_3;
teta = fabsf(atanf(beta/alfa));

if (alfa<0 && beta>=0)
    tetaf = pi - teta;
else
{
    if (alfa<0 && beta<0)
        tetaf = pi + teta;
    else
    {
        if (alfa>=0 && beta<0)
            tetaf = 2 * pi-teta;
        else
            if (alfa>=0 && beta>=0)
                tetaf = teta;
    }
}

// Transformada ABC-DQ de corriente
Da = cosf(tetaf) * x2_3;
Db = cosf(tetaf-g120) * x2_3;
Dc = cosf(tetaf+g120) * x2_3;

Qa = -sinf(tetaf) * x2_3;
Qb = -sinf(tetaf-g120) * x2_3;
Qc = -sinf(tetaf+g120) * x2_3;

/*****/
// Corrientes en eje D y eje Q
id_m = Da*ia_m + Db*ib_m + Dc*ic_m;
iq_m = Qa*ia_m + Qb*ib_m + Qc*ic_m;
id_r = Da*ia_r1 + Db*ib_r1 + Dc*ic_r1;
iq_r = Qa*ia_r1 + Qb*ib_r1 + Qc*ic_r1;
//iq_r = 0;

input[0] = ia_r1;
input[1] = ia_m;
input[2] = ia_r1-ia_m;
input[3] = ib_r1;
input[4] = ib_m;
input[5] = ib_r1-ib_m;
input[6] = ic_r1;
input[7] = ic_m;
input[8] = ic_r1-ic_m;
input[9] = mlat;

```

```

input[10] = m1bt;
input[11] = m1ct;

for (i = 0; i < n_in; ++i)
{
    input2[i] = (2/(xmax[i]-xmin[i]))*(input[i] - xmin[i]) -1;
}

for (i = 0; i < n_lyr1; i++) {
    float suma = 0;
    for (j = 0; j < n_in; j++) {
        suma += iw[i][j] * input2[j];
    }
    outlayer1[i] = tanhsp(suma + ib[i][0]);
}

for (i = 0; i < n_lyr2; i++) {
    float suma = 0;
    for (j = 0; j < n_lyr1; j++) {
        suma += ow[i][j] * outlayer1[j];
    }
    outlayer2[i] = suma + ob[i][0];
}

for (i = 0; i < n_lyr2; i++){
    output[i] = ((ymax[i]-ymin[i])/2)*(outlayer2[i] + 1 ) + ymin[i];
}

/*****/
mA = output[0];
mB = output[1];
mC = output[2];
/*****/

// Salida de variables
m1_A=mA*MO_MAX;
m1_B=mB*MO_MAX;
m1_C=mC*MO_MAX;

MOCMPR1 = m1_A;
MOCMPR2 = m1_B;
MOCMPR3 = m1_C;

M1CMPR1 = m1_A;
M1CMPR2 = m1_B;
M1CMPR3 = m1_C;

M2CMPR1 = m1_A;
M2CMPR2 = m1_B;
M2CMPR3 = m1_C;

auxiliar = a_ref;
m1at=mA;
m1bt=mB;
m1ct=mC;

M3CMPR1 = 0;
}

```

```

/*
*****
*/

```

B.3. Control FCS-MPC

```

/*
*****

**      HEADER FILES
*****
*/
/* HW Macros */

/* System Defines */
#include <funciones_MPC.h>
#include "ANN_app.h"
#include "aceC6748.h"
#include "assert.h"
#include "math.h"

#include "psc.h"
#include "hw_psc_C6748.h"
#include "ace_monitor.h"
#include "soc_C6748.h"

#include "coss.p.h"
#include <param_10khz.h>
#include <mat_1.h>

/* * Monitor */
#define SYSCLK2_FREQ      (150000000u)
#define HMI_UART_BAUDRATE (115200u)
#define HMI_UART_TIMEOUT  (20000)
bool tick_flag = 0;
#pragma DATA_SECTION (array0_A1, ".ddr")
#pragma DATA_SECTION (array0_A2, ".ddr")
#pragma DATA_SECTION (array0_B1, ".ddr")
#pragma DATA_SECTION (array0_B2, ".ddr")
#pragma DATA_SECTION (array1_A1, ".ddr")
#pragma DATA_SECTION (array1_A2, ".ddr")
#pragma DATA_SECTION (array1_B1, ".ddr")
#pragma DATA_SECTION (array1_B2, ".ddr")
#pragma DATA_SECTION (array2_A1, ".ddr")
#pragma DATA_SECTION (array2_A2, ".ddr")
#pragma DATA_SECTION (array2_B1, ".ddr")
#pragma DATA_SECTION (array2_B2, ".ddr")
#pragma DATA_SECTION (array3_A1, ".ddr")
#pragma DATA_SECTION (array3_A2, ".ddr")

extern void Init_Platform(void);
extern void Init_Interrupts(void);
extern void Init_Adc(void);
extern void Init_Iop(void);
extern void Init_Mod(void);
extern void Init_DS_LED(void);

```

```

#define array_length 4000

int s1_A = 0;
int s2_A = 0;
int s3_A = 0;
int s1_B = 0;
int s2_B = 0;
int s3_B = 0;
int s1_C = 0;
int s2_C = 0;
int s3_C = 0;

float temp1, temp2, temp3;
float d_ia, d_vc1a, d_vc2a, d_ib, d_vc1b, d_vc2b, d_ic, d_vc1c, d_vc2c;

float VaN = 0;
float VbN = 0;
float VcN = 0;

float VaN_m, VbN_m, VcN_m;

float a_ref = 2;
float ia_r, ib_r, ic_r;
float ia_r1, ib_r1, ic_r1;
float i1_r, i2_r, i3_r;
float ia_m;
float ib_m;
float ic_m;
float vc1a_m;
float vc2a_m;
float vc1b_m;
float vc2b_m;
float vc1c_m;
float vc2c_m;

float VaN_t=0;
float VbN_t=0;
float VcN_t=0;
float VoN=0;

float g_min=0;
float g=0;

float ia_t=0;
float ib_t=0;
float ic_t=0;
float ia_t2=0;
float ib_t2=0;
float ic_t2=0;
float vc1a_t=0;
float vc2a_t=0;
float vc1b_t=0;
float vc2b_t=0;
float vc1c_t=0;
float vc2c_t=0;
float vc1a_t2=0;
float vc2a_t2=0;
float vc1b_t2=0;

```

```
float vc2b_t2=0;
float vc1c_t2=0;
float vc2c_t2=0;

float alfa=0;
float beta=0;
float teta=0;
float tetaf=0;
float Da=0;
float Db=0;
float Dc=0;
float Qa=0;
float Qb=0;
float Qc=0;

float id_m, iq_m;

double t1;
int act=0;

float gain_i = -2.51291866;
float offset_i = 0;
float gain_ic = -4.954716981;
float offset_ic = 0;

float gain_v1 = -39.86710963;
float offset_v1 = 0;
float gain_v2 = -99.45355191;
float offset_v2 = 0;

int16_t Med_ADC0_A1_16;
int16_t Med_ADC0_A2_16;
int16_t Med_ADC0_B1_16;
int16_t Med_ADC0_B2_16;
float array0_A1[array_length];
float array0_A2[array_length];
float array0_B1[array_length];
float array0_B2[array_length];

int16_t Med_ADC1_A1_16;
int16_t Med_ADC1_A2_16;
int16_t Med_ADC1_B1_16;
int16_t Med_ADC1_B2_16;
float array1_A1[array_length];
float array1_A2[array_length];
float array1_B1[array_length];
float array1_B2[array_length];

int16_t Med_ADC2_A1_16;
int16_t Med_ADC2_A2_16;

float array2_A1[array_length];
float array2_A2[array_length];
float array2_B1[array_length];
float array2_B2[array_length];

float array3_A1[array_length];
float array3_A2[array_length];
```

```

int array_i = 0;
float leer = 0;
double t=0;

int i=0;
int j=0;
float sum;

float auxiliar;
float Vdc;

void main(void)
{
    /* All peripherals to be uses are configured here */

    /* Interruptions must be configured after peripherals configuration */
    Init_Platform();      // Configuracion de PLL del DSP y de comunicacion
con la FPGA
    Init_Mod();           // Configuracion de los Timers y moduladores PWM
    Init_Adc();           // Configuracion de los ADCs
    Init_Iop();           // Configuracion de los puertos IO
    Init_DS_LED();        // Configuracion de DipSwitch y LEDs

    Init_Interrupts();    // Configuracion de las interrupciones. Debe ser
la ultima funcion a llamar
    /*
-----
*/
/* BACKGROUND ROUTINE
    */
/*
-----
*/
/* LOCAL MONITOR CONFIGURATION */
/* UART configuration */
PSCModuleControl(SOC_PSC_1_REGS,HW_PSC_UART2,PSC_POWERDOMAIN_ALWAYS_ON,
PSC_MDCTL_NEXT_ENABLE);
ACE_DSPUART_PinMuxSetup(SOC_UART_2, FALSE);

    ACE_MON_init(SOC_UART_2_REGS, SYSCLK2_FREQ, HMI_UART_BAUDRATE,
HMI_UART_TIMEOUT);
    ACE_MON_addVariableToMonitor(&ia_m, ACE_MON_FLOAT, sizeof(ia_m),
ACE_MON_ACCESS_READONLY, array0_A1, sizeof(array0_A1));
    ACE_MON_addVariableToMonitor(&ib_m, ACE_MON_FLOAT, sizeof(ib_m),
ACE_MON_ACCESS_READONLY, array0_A2, sizeof(array0_A2));
    ACE_MON_addVariableToMonitor(&ic_m, ACE_MON_FLOAT, sizeof(ic_m),
ACE_MON_ACCESS_READONLY, array0_B1, sizeof(array0_B1));

    ACE_MON_addVariableToMonitor(&vc1a_m, ACE_MON_FLOAT, sizeof(vc1a_m),
ACE_MON_ACCESS_READONLY, array0_B2, sizeof(array0_B2));
    ACE_MON_addVariableToMonitor(&vc2a_m, ACE_MON_FLOAT, sizeof(vc2a_m),
ACE_MON_ACCESS_READONLY, array1_A1, sizeof(array1_A1));

    ACE_MON_addVariableToMonitor(&vc1b_m, ACE_MON_FLOAT, sizeof(vc1b_m),
ACE_MON_ACCESS_READONLY, array1_A2, sizeof(array1_A2));
    ACE_MON_addVariableToMonitor(&vc2b_m, ACE_MON_FLOAT, sizeof(vc2b_m),
ACE_MON_ACCESS_READONLY, array1_B1, sizeof(array1_B1));

```

```

    ACE_MON_addVariableToMonitor(&vc1c_m, ACE_MON_FLOAT,    sizeof(vc1c_m),
    ACE_MON_ACCESS_READONLY, array1_B2, sizeof(array1_B2));
    ACE_MON_addVariableToMonitor(&vc2c_m, ACE_MON_FLOAT,    sizeof(vc2c_m),
    ACE_MON_ACCESS_READONLY, array2_A1, sizeof(array2_A1));

    ACE_MON_addVariableToMonitor(&ia_r1, ACE_MON_FLOAT,    sizeof(ia_r1),
    ACE_MON_ACCESS_READONLY, array2_A2, sizeof(array2_A2));
    ACE_MON_addVariableToMonitor(&id_m, ACE_MON_FLOAT,    sizeof(id_m),
    ACE_MON_ACCESS_READONLY, array2_B1, sizeof(array2_B1));
    ACE_MON_addVariableToMonitor(&iq_m, ACE_MON_FLOAT,    sizeof(iq_m),
    ACE_MON_ACCESS_READONLY, array2_B2, sizeof(array2_B2));

    ACE_MON_addVariableToMonitor(&Vdc, ACE_MON_FLOAT,    sizeof(Vdc),
    ACE_MON_ACCESS_READONLY, array3_A1, sizeof(array3_A1));

    ACE_MON_addVariableToMonitor(&auxiliar, ACE_MON_FLOAT,    sizeof(
    auxiliar), ACE_MON_ACCESS_READONLY, array3_A2, sizeof(array3_A2));
    //ACE_MON_addVariableToMonitor(&VbN_m, ACE_MON_FLOAT,    sizeof(VbN_m),
    ACE_MON_ACCESS_READONLY, array3_A2, sizeof(array3_A2));
    //ACE_MON_addVariableToMonitor(&VcN_m, ACE_MON_FLOAT,    sizeof(VcN_m),
    ACE_MON_ACCESS_READONLY, array3_B1, sizeof(array3_B1));

    ACE_MON_addVariableToMonitor(&leer, ACE_MON_FLOAT,    sizeof(leer),
    ACE_MON_ACCESS_WRITABLE, ACE_MON_NULL,0);
    ACE_MON_addVariableToMonitor(&a_ref, ACE_MON_FLOAT,    sizeof(a_ref),
    ACE_MON_ACCESS_WRITABLE, ACE_MON_NULL,0);

    while(1)
    {
        ACE_MON_tickBackground();
    }
}
/*
*****
*/
extern void ADC_fpgaSyncInterrupt_isr(void){
//int var_temp_int;
/* Clear flag that caused the interruption */
INTOCLRFLG = INTO_ADC0_MASK; // Borra el Flag sobreescribiendo un 1
/*
-----
*/
t = t + h;
t1 = t + h + h;

i1_r=cosp(pi100*t);
i2_r=cosp(pi100*t-x2_3*pi);
i3_r=cosp(pi100*t+x2_3*pi);

//ia_r=a_ref*cosp(pi100*t);
//ib_r=a_ref*cosp(pi100*t-x2_3*pi);
//ic_r=a_ref*cosp(pi100*t+x2_3*pi);

ia_r1 = a_ref*cosp(pi100*t1);
ib_r1 = a_ref*cosp(pi100*t1 - x2_3*pi);
ic_r1 = a_ref*cosp(pi100*t1 + x2_3*pi);

/*****

```

```

// Calculo de Teta
alfa = i1_r;
beta = (i2_r-i3_r) * raiz1_3;
teta = fabsf(atanf(beta/alfa));

if (alfa<0 && beta>=0)
    tetaf = pi - teta;
else
{
    if (alfa<0 && beta<0)
        tetaf = pi + teta;
    else
    {
        if (alfa>=0 && beta<0)
            tetaf = 2 * pi-teta;
        else
            if (alfa>=0 && beta>=0)
                tetaf = teta;
    }
}

// Transformada ABC-DQ de corriente
Da = cosf(tetaf) * x2_3;
Db = cosf(tetaf-g120) * x2_3;
Dc = cosf(tetaf+g120) * x2_3;

Qa = -sinf(tetaf) * x2_3;
Qb = -sinf(tetaf-g120) * x2_3;
Qc = -sinf(tetaf+g120) * x2_3;

/*****/
// Corrientes en eje D y eje Q
id_m = Da*ia_m + Db*ib_m + Dc*ic_m;
iq_m = Qa*ia_m + Qb*ib_m + Qc*ic_m;

Med_ADC0_A1_16 = ADC0CHA1;
Med_ADC0_A2_16 = ADC0CHA2;
Med_ADC0_B1_16 = ADC0CHB1;
Med_ADC0_B2_16 = ADC0CHB2;

ia_m = ((int)Med_ADC0_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_i + offset_i;
ib_m = ((int)Med_ADC0_A2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_i + offset_i;
ic_m = ((int)Med_ADC0_B1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_ic + offset_ic;
vc1a_m = ((int)Med_ADC0_B2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v1 + offset_v1;

Med_ADC1_A1_16 = ADC1CHA1;
Med_ADC1_A2_16 = ADC1CHA2;
Med_ADC1_B1_16 = ADC1CHB1;
Med_ADC1_B2_16 = ADC1CHB2;

vc2a_m = ((int)Med_ADC1_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v2 + offset_v2;
vc1b_m = ((int)Med_ADC1_A2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v1 + offset_v1;

```

```

vc2b_m = ((int)Med_ADC1_B1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v2 + offset_v2;
vc1c_m = ((int)Med_ADC1_B2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v1 + offset_v1;

Med_ADC2_A1_16 = ADC2CHA1;
Med_ADC2_A2_16 = ADC2CHA2;

vc2c_m = ((int)Med_ADC2_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v2 + offset_v2;
Vdc = ((int)Med_ADC2_A2_16*ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*-132;

VaN_m=VDC*s3_A+vc2a_m*(s2_A-s3_A)+vc1a_m*(s1_A-s2_A);
VbN_m=VDC*s3_B+vc2b_m*(s2_B-s3_B)+vc1b_m*(s1_B-s2_B);
VcN_m=VDC*s3_C+vc2c_m*(s2_C-s3_C)+vc1c_m*(s1_C-s2_C);

tick_flag = ACE_MON_tickForeground();
if (leer==1){
    array0_A1[array_i] = ia_m;
    array0_A2[array_i] = ib_m;
    array0_B1[array_i] = ic_m;
    array0_B2[array_i] = vc1a_m;
    array1_A1[array_i] = vc2a_m;
    array1_A2[array_i] = vc1b_m;
    array1_B1[array_i] = vc2b_m;
    array1_B2[array_i] = vc1c_m;
    array2_A1[array_i] = vc2c_m;
    array2_A2[array_i] = ia_r1;
    array2_B1[array_i] = id_m;
    array2_B2[array_i] = iq_m;
    array3_A1[array_i] = Vdc;
    array3_A2[array_i] = auxiliar;
    //array3_A2[array_i] = VbN_m;
    //array3_B1[array_i] = VcN_m;

    //if (array_i==array_length/4) a_ref = a_ref +2;
    //if (array_i==array_length*2/4) a_ref = a_ref +2;
    //if (array_i==array_length*3/4) a_ref = a_ref +2;

    array_i = array_i+1;

    if (array_i>= array_length){
        array_i = 0;
        leer=0;
    }
}

/*****
// Actualizacion variables k+1
VoN=150;

ia_t=(VaN-VoN)*k1+ia_m*k2;
ib_t=(VbN-VoN)*k1+ib_m*k2;
ic_t=(VcN-VoN)*k1+ic_m*k2;

vc1a_t=(ia_t+ia_m)*kC1*(s2_A-s1_A)+vc1a_m;
vc2a_t=(ia_t+ia_m)*kC2*(s3_A-s2_A)+vc2a_m;
vc1b_t=(ib_t+ib_m)*kC1*(s2_B-s1_B)+vc1b_m;
vc2b_t=(ib_t+ib_m)*kC2*(s3_B-s2_B)+vc2b_m;

```

```

vc1c_t=(ic_t+ic_m)*kC1*(s2_C-s1_C)+vc1c_m;
vc2c_t=(ic_t+ic_m)*kC2*(s3_C-s2_C)+vc2c_m;

/*****/
// Control MPC k+2

act=calculate_g(S3A, S2A, S1A, S3B, S2B, S1B, S3C, S2C, S1C, VDC, vc1_r,
vc2_r,
vc1a_t, vc2a_t, vc1b_t, vc2b_t, vc1c_t, vc2c_t, ia_t, ib_t,
ic_t,
ia_r1, ib_r1, ic_r1, k1, k2, kC1, kC2, w1, w2, contX, g,
act);

// Salida de variables
auxiliar = a_ref;//act;

s1_A=S1A[act];
s2_A=S2A[act];
s3_A=S3A[act];
s1_B=S1B[act];
s2_B=S2B[act];
s3_B=S3B[act];
s1_C=S1C[act];
s2_C=S2C[act];
s3_C=S3C[act];

VaN=VDC*s3_A+vc2a_t*(s2_A-s3_A)+vc1a_t*(s1_A-s2_A);
VbN=VDC*s3_B+vc2b_t*(s2_B-s3_B)+vc1b_t*(s1_B-s2_B);
VcN=VDC*s3_C+vc2c_t*(s2_C-s3_C)+vc1c_t*(s1_C-s2_C);

/*****/

MOCMPR1 = s1_A * MO_MAX;
MOCMPR2 = s1_B * MO_MAX;
MOCMPR3 = s1_C * MO_MAX;

M1CMPR1 = s2_A * MO_MAX;
M1CMPR2 = s2_B * MO_MAX;
M1CMPR3 = s2_C * MO_MAX;

M2CMPR1 = s3_A * MO_MAX;
M2CMPR2 = s3_B * MO_MAX;
M2CMPR3 = s3_C * MO_MAX;

M3CMPR1 = MO_MAX/2;
}

/*
*****
*/

```

B.4. Control RNA-MPC

```

/*****
**      HEADER FILES
*****/
/* HW Macros */

```

```

/* System Defines */
#include "aceC6748.h"
#include "assert.h"
#include "math.h"
#include "psc.h"
#include "hw_psc_C6748.h"
#include "ace_monitor.h"

#include <param_v3.h>
#include <PWM_app.h>
// #include "cosp.h"

/*
 * Monitor.
 */
#define SYSCLK2_FREQ      (150000000u)
#define HMI_UART_BAUDRATE (115200u)
#define HMI_UART_TIMEOUT  (20000)

#define array_length 4000

bool tick_flag = 0;
#pragma DATA_SECTION (array0_A1, ".ddr")
#pragma DATA_SECTION (array0_A2, ".ddr")
#pragma DATA_SECTION (array0_B1, ".ddr")
#pragma DATA_SECTION (array0_B2, ".ddr")
#pragma DATA_SECTION (array1_A1, ".ddr")
#pragma DATA_SECTION (array1_A2, ".ddr")
#pragma DATA_SECTION (array1_B1, ".ddr")
#pragma DATA_SECTION (array1_B2, ".ddr")
#pragma DATA_SECTION (array2_A1, ".ddr")
#pragma DATA_SECTION (array2_A2, ".ddr")
#pragma DATA_SECTION (array2_B1, ".ddr")
#pragma DATA_SECTION (array2_B2, ".ddr")
#pragma DATA_SECTION (array3_A1, ".ddr")
#pragma DATA_SECTION (array3_A2, ".ddr")

extern void Init_Platform(void);
extern void Init_Interrupts(void);
extern void Init_Adc(void);
extern void Init_Iop(void);
extern void Init_Mod(void);
extern void Init_DS_LED(void);

double aux = 0;
float a_ref = 2;

float ia_r, ib_r, ic_r;
float ia_r1, ib_r1, ic_r1;
float i1_r, i2_r, i3_r;
float ia_m;
float ib_m;
float ic_m;
float vc1a_m;
float vc2a_m;
float vc1b_m;
float vc2b_m;
float vc1c_m;

```

```
float vc2c_m;

float m1_A = 0;
float m2_A = 0;
float m3_A = 0;
float m1_B = 0;
float m2_B = 0;
float m3_B = 0;
float m1_C = 0;
float m2_C = 0;
float m3_C = 0;

float ud_t = 0;
float uq_t = 0;
float xd_t = 0;
float xq_t = 0;

float alfa=0;
float beta=0;
float teta=0;
float tetaf=0;
float Da=0;
float Db=0;
float Dc=0;
float Qa=0;
float Qb=0;
float Qc=0;
double t1=0;
float errorD=0;
float errorQ=0;
float iD_r=0;
float iD_m=0;
float iQ_r=0;
float iQ_m=0;
float xd_t1=0;
float ud_t1=0;
float xq_t1=0;
float uq_t1=0;
float DET=0;
float DET_1=0;

float mA=0;
float mB=0;
float mC=0;

float gain_i = -2.51291866;
float offset_i = 0;
float gain_ic = -4.954716981;
float offset_ic = 0;

float gain_v1 = -39.86710963;
float offset_v1 = 0;
float gain_v2 = -99.45355191;
float offset_v2 = 0;

int16_t Med_ADC0_A1_16;
int16_t Med_ADC0_A2_16;
int16_t Med_ADC0_B1_16;
int16_t Med_ADC0_B2_16;
```

```

float array0_A1[array_length];
float array0_A2[array_length];
float array0_B1[array_length];
float array0_B2[array_length];

int16_t Med_ADC1_A1_16;
int16_t Med_ADC1_A2_16;
int16_t Med_ADC1_B1_16;
int16_t Med_ADC1_B2_16;
float array1_A1[array_length];
float array1_A2[array_length];
float array1_B1[array_length];
float array1_B2[array_length];

int16_t Med_ADC2_A1_16;
int16_t Med_ADC2_A2_16;
float array2_A1[array_length];
float array2_A2[array_length];
float array2_B1[array_length];
float array2_B2[array_length];

float array3_A1[array_length];
float array3_A2[array_length];

float auxiliar;
float leer = 0;
float Vdc;

int array_i = 0;

void main(void)
{
    /* All peripherals to be uses are configured here */

    /* Interruptions must be configured after peripherals configuration */
    Init_Platform();      // Configuracion de PLL del DSP y de comunicacion
con la FPGA
    Init_Mod();           // Configuracion de los Timers y moduladores PWM
    Init_Adc();           // Configuracion de los ADCs
    Init_Iop();           // Configuracion de los puertos IO
    Init_DS_LED();        // Configuracion de DipSwitch y LEDs

    Init_Interrupts();    // Configuracion de las interrupciones. Debe ser
la ultima funcion a llamar

    ACE_USRIO_turnOffLd(ACE_USRIO_LD0);
    ACE_USRIO_turnOffLd(ACE_USRIO_LD1);
    ACE_USRIO_turnOffLd(ACE_USRIO_LD2);
    ACE_USRIO_turnOffLd(ACE_USRIO_LD3);

    /*
-----
*/
    /* BACKGROUND ROUTINE
        */
    /*
-----
*/
}

```

```

/* LOCAL MONITOR CONFIGURATION */
/* UART configuration */
PSCModuleControl(SOC_PSC_1_REGS, HW_PSC_UART2, PSC_POWERDOMAIN_ALWAYS_ON,
PSC_MDCTL_NEXT_ENABLE);
ACE_DSPUART_PinMuxSetup(SOC_UART_2, FALSE);

ACE_MON_init(SOC_UART_2_REGS, SYSCLK2_FREQ, HMI_UART_BAUDRATE,
HMI_UART_TIMEOUT);
ACE_MON_addVariableToMonitor(&ia_m, ACE_MON_FLOAT, sizeof(ia_m),
ACE_MON_ACCESS_READONLY, array0_A1, sizeof(array0_A1));
ACE_MON_addVariableToMonitor(&ib_m, ACE_MON_FLOAT, sizeof(ib_m),
ACE_MON_ACCESS_READONLY, array0_A2, sizeof(array0_A2));
ACE_MON_addVariableToMonitor(&ic_m, ACE_MON_FLOAT, sizeof(ic_m),
ACE_MON_ACCESS_READONLY, array0_B1, sizeof(array0_B1));

ACE_MON_addVariableToMonitor(&vc1a_m, ACE_MON_FLOAT, sizeof(vc1a_m),
ACE_MON_ACCESS_READONLY, array0_B2, sizeof(array0_B2));
ACE_MON_addVariableToMonitor(&vc2a_m, ACE_MON_FLOAT, sizeof(vc2a_m),
ACE_MON_ACCESS_READONLY, array1_A1, sizeof(array1_A1));

ACE_MON_addVariableToMonitor(&vc1b_m, ACE_MON_FLOAT, sizeof(vc1b_m),
ACE_MON_ACCESS_READONLY, array1_A2, sizeof(array1_A2));
ACE_MON_addVariableToMonitor(&vc2b_m, ACE_MON_FLOAT, sizeof(vc2b_m),
ACE_MON_ACCESS_READONLY, array1_B1, sizeof(array1_B1));

ACE_MON_addVariableToMonitor(&vc1c_m, ACE_MON_FLOAT, sizeof(vc1c_m),
ACE_MON_ACCESS_READONLY, array1_B2, sizeof(array1_B2));
ACE_MON_addVariableToMonitor(&vc2c_m, ACE_MON_FLOAT, sizeof(vc2c_m),
ACE_MON_ACCESS_READONLY, array2_A1, sizeof(array2_A1));

ACE_MON_addVariableToMonitor(&ia_r1, ACE_MON_FLOAT, sizeof(ia_r1),
ACE_MON_ACCESS_READONLY, array2_A2, sizeof(array2_A2));
ACE_MON_addVariableToMonitor(&iD_m, ACE_MON_FLOAT, sizeof(iD_m),
ACE_MON_ACCESS_READONLY, array2_B1, sizeof(array2_B1));
ACE_MON_addVariableToMonitor(&iQ_m, ACE_MON_FLOAT, sizeof(iQ_m),
ACE_MON_ACCESS_READONLY, array2_B2, sizeof(array2_B2));

ACE_MON_addVariableToMonitor(&Vdc, ACE_MON_FLOAT, sizeof(Vdc),
ACE_MON_ACCESS_READONLY, array3_A1, sizeof(array3_A1));

ACE_MON_addVariableToMonitor(&auxiliar, ACE_MON_FLOAT, sizeof(
auxiliar), ACE_MON_ACCESS_READONLY, array3_A2, sizeof(array3_A2));
//ACE_MON_addVariableToMonitor(&VcN_m, ACE_MON_FLOAT, sizeof(VcN_m),
ACE_MON_ACCESS_READONLY, array3_B1, sizeof(array3_B1));

ACE_MON_addVariableToMonitor(&leer, ACE_MON_FLOAT, sizeof(leer),
ACE_MON_ACCESS_WRITABLE, ACE_MON_NULL, 0);
ACE_MON_addVariableToMonitor(&a_ref, ACE_MON_FLOAT, sizeof(a_ref),
ACE_MON_ACCESS_WRITABLE, ACE_MON_NULL, 0);

while(1)
{
    ACE_MON_tickBackground();
}
}
/*****
extern void ADC_fpgaSyncInterrupt_isr(void){
//int var_temp_int;
/* Clear flag that caused the interruption */

```

```

INTOCLRFLG = INTO_ADC0_MASK; // Borra el Flag sobrescribiendo un 1
/*
-----
*/
aux=aux+h;
t1=aux-1*h;

i1_r=cos(2*pi*50*aux);
i2_r=cos(2*pi*50*aux-x2_3*pi);
i3_r=cos(2*pi*50*aux+x2_3*pi);

ia_r1=a_ref*cos(2*pi*50*t1);
ib_r1=a_ref*cos(2*pi*50*t1-x2_3*pi);
ic_r1=a_ref*cos(2*pi*50*t1+x2_3*pi);

ia_r=a_ref*cos(2*pi*50*aux);
ib_r=a_ref*cos(2*pi*50*aux-x2_3*pi);
ic_r=a_ref*cos(2*pi*50*aux+x2_3*pi);

Med_ADC0_A1_16 = ADC0CHA1;
Med_ADC0_A2_16 = ADC0CHA2;
Med_ADC0_B1_16 = ADC0CHB1;
Med_ADC0_B2_16 = ADC0CHB2;

ia_m = ((int)Med_ADC0_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_i + offset_i;
ib_m = ((int)Med_ADC0_A2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_i + offset_i;
ic_m = ((int)Med_ADC0_B1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_ic + offset_ic;
vc1a_m = ((int)Med_ADC0_B2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v1 + offset_v1;

array0_A1[array_i] = ia_m;
array0_A2[array_i] = ib_m;
array0_B1[array_i] = ic_m;
array0_B2[array_i] = vc1a_m;

Med_ADC1_A1_16 = ADC1CHA1;
Med_ADC1_A2_16 = ADC1CHA2;
Med_ADC1_B1_16 = ADC1CHB1;
Med_ADC1_B2_16 = ADC1CHB2;

vc2a_m = ((int)Med_ADC1_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v2 + offset_v2;
vc1b_m = ((int)Med_ADC1_A2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v1 + offset_v1;
vc2b_m = ((int)Med_ADC1_B1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v2 + offset_v2;
vc1c_m = ((int)Med_ADC1_B2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v1 + offset_v1;

array1_A1[array_i] = vc2a_m;
array1_A2[array_i] = vc1b_m;
array1_B1[array_i] = vc2b_m;
array1_B2[array_i] = vc1c_m;

Med_ADC2_A1_16 = ADC2CHA1;
Med_ADC2_A2_16 = ADC2CHA2;

```

```

vc2c_m = ((int)Med_ADC2_A1_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v2 + offset_v2;
//Vdc = ((int)Med_ADC2_A2_16 * ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*
gain_v2 + offset_v2;
Vdc = ((int)Med_ADC2_A2_16*ACE_ADC_ESS_M10P10_CONVERSION_FACTOR)*-132;

auxiliar = a_ref;
tick_flag = ACE_MON_tickForeground();
if (leer==1){
    array0_A1[array_i] = ia_m;
    array0_A2[array_i] = ib_m;
    array0_B1[array_i] = ic_m;
    array0_B2[array_i] = vc1a_m;
    array1_A1[array_i] = vc2a_m;
    array1_A2[array_i] = vc1b_m;
    array1_B1[array_i] = vc2b_m;
    array1_B2[array_i] = vc1c_m;
    array2_A1[array_i] = vc2c_m;
    array2_A2[array_i] = ia_r1;
    array2_B1[array_i] = iD_m;
    array2_B2[array_i] = iQ_m;
    array3_A1[array_i] = Vdc;
    array3_A2[array_i] = auxiliar;
    //array3_B1[array_i] = VcN_m;
    //if (array_i==array_length/4) a_ref = a_ref +2;
    //if (array_i==array_length*2/4) a_ref = a_ref +2;
    //if (array_i==array_length*3/4) a_ref = a_ref +2;

    array_i = array_i+1;

    if (array_i>= array_length){
        array_i = 0;
        leer=0;
    }
}

// Calculo de Teta
alfa = i1_r;
beta = (i2_r-i3_r) * raiz1_3;
teta = fabsf(atanf(beta/alfa));

if (alfa<0 && beta>=0)
    tetaf = pi - teta;
else
{
    if (alfa<0 && beta<0)
        tetaf = pi + teta;
    else
    {
        if (alfa>=0 && beta<0)
            tetaf = 2 * pi-teta;
        else
            if (alfa>=0 && beta>=0)
                tetaf = teta;
    }
}

// Transformada ABC-DQ de corriente

```

```

Da = cosf(tetaf) * x2_3;
Db = cosf(tetaf-g120) * x2_3;
Dc = cosf(tetaf+g120) * x2_3;

Qa = -sinf(tetaf) * x2_3;
Qb = -sinf(tetaf-g120) * x2_3;
Qc = -sinf(tetaf+g120) * x2_3;

/*****/
// Corrientes en eje D y eje Q
iD_m = Da*ia_m + Db*ib_m + Dc*ic_m;
iQ_m = Qa*ia_m + Qb*ib_m + Qc*ic_m;
iD_r = Da*ia_r1 + Db*ib_r1 + Dc*ic_r1;
iQ_r = Qa*ia_r1 + Qb*ib_r1 + Qc*ic_r1;
//iQ_r = 0;
/*****/
// Calculo eje D
errorD = iD_r-iD_m;

xd_t1 = (ud_t * n2) - (xd_t * d2);
ud_t1 = (errorD-xd_t1) * kpi;

if(ud_t1>=VDC_2)
    ud_t1 = VDC_2;
else
{
    if(ud_t1<=-VDC_2)
        ud_t1 = -VDC_2;
}
/*****/
// Calculo eje Q
errorQ = iQ_r-iQ_m;

xq_t1 = (uq_t * n2) - (xq_t * d2);
uq_t1 = (errorQ-xq_t1) * kpi;

if(uq_t1>=VDC_2)
    uq_t1 = VDC_2;
else
{
    if(uq_t1<=-VDC_2)
        uq_t1 = -VDC_2;
}
/*****/
// Transformada DQ-ABC de tension
DET = Da * (Qb-Qc) - Db * (Qa-Qc) + Dc * (Qa-Qb);
DET_1 =1/DET;
/*****/
mA = ((Qb-Qc) * ud_t1 - (Db-Dc) * uq_t1) * DET_1 / VDC;
mB = ((Qc-Qa) * ud_t1 + (Da-Dc) * uq_t1) * DET_1 / VDC;
mC = ((Qa-Qb) * ud_t1 - (Da-Db) * uq_t1) * DET_1 / VDC;
/*****/

if(mA>0.5)
    mA = 0.5;
else
{
    if(mA<-0.5)
        mA = -0.5;
}

```

```

    }
    if(mB>0.5)
        mB = 0.5;
    else
    {
        if(mB<-0.5)
            mB = -0.5;
    }
    if(mC>0.5)
        mC = 0.5;
    else
    {
        if(mC<-0.5)
            mC = -0.5;
    }

    // Salida de variables
    m1_A=(mA+0.5)*MO_MAX;
    m2_A=(mA+0.5)*MO_MAX;
    m3_A=(mA+0.5)*MO_MAX;
    m1_B=(mB+0.5)*MO_MAX;
    m2_B=(mB+0.5)*MO_MAX;
    m3_B=(mB+0.5)*MO_MAX;
    m1_C=(mC+0.5)*MO_MAX;
    m2_C=(mC+0.5)*MO_MAX;
    m3_C=(mC+0.5)*MO_MAX;
    xd_t=xd_t1;
    ud_t=ud_t1;
    xq_t=xq_t1;
    uq_t=uq_t1;
    //var_temp_int = (int)(((Med_ADC_A1 + 10)/20) * MO_MAX);

    MOCMPR1 = m1_A;
    MOCMPR2 = m1_B;
    MOCMPR3 = m1_C;

    M1CMPR1 = m2_A;
    M1CMPR2 = m2_B;
    M1CMPR3 = m2_C;

    M2CMPR1 = m3_A;
    M2CMPR2 = m3_B;
    M2CMPR3 = m3_C;

    M3CMPR1 = 0.5*MO_MAX;
}

/*
*****
*/

```

Bibliografía

- [1] P. Cortes, S. Kouro, B. La Rocca, R. Vargas, J. Rodriguez, J. I. Leon, S. Vazquez, and L. G. Franquelo, “Guidelines for weighting factors design in model predictive control of power converters and drives,” in *2009 IEEE International Conference on Industrial Technology*, 2009, pp. 1–7.
- [2] B. K. Bose, “Neural network applications in power electronics and motor drives—an introduction and perspective,” *IEEE Transactions on Industrial Electronics*, vol. 54, no. 1, pp. 14–33, 2007.
- [3] S. S. Haykin, *Neural networks and learning machines*, 3rd ed. Upper Saddle River, NJ: Pearson Education, 2009.
- [4] P. Lezana, R. Aguilera, and D. Quevedo, “Steady-state issues with finite control set model predictive control,” in *2009 35th Annual Conference of IEEE Industrial Electronics*, 2009, pp. 1776–1781.
- [5] M. Norambuena, C. Garcia, J. Rodriguez, and P. Lezana, “Finite control set model predictive control reduced computational cost applied to a flying capacitor converter,” in *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, 2017, pp. 4903–4907.
- [6] I. S. Mohamed, S. Rovetta, T. D. Do, T. Dragicević, and A. A. Z. Diab, “A neural-network-based model predictive control of three-phase inverter with an output LC filter,” *IEEE Access*, vol. 7, pp. 124 737–124 749, 2019.
- [7] D. Wang, Z. J. Shen, X. Yin, S. Tang, X. Liu, C. Zhang, J. Wang, J. Rodriguez, and M. Norambuena, “Model predictive control using artificial neural network for power converters,” *IEEE Transactions on Industrial Electronics*, vol. 69, no. 4, pp. 3689–3699, 2022.
- [8] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, <http://www.deeplearningbook.org>.
- [9] J. Huang and K. Corzine, “Extended operation of flying capacitor multilevel inverters,” *IEEE Transactions on Power Electronics*, vol. 21, no. 1, pp. 140–147, 2006.
- [10] J. Rodriguez, J.-S. Lai, and F. Z. Peng, “Multilevel inverters: a survey of topologies, controls, and applications,” *IEEE Transactions on Industrial Electronics*, vol. 49, no. 4, pp. 724–738, 2002.

- [11] S. Vazquez, J. Rodriguez, M. Rivera, L. G. Franquelo, and M. Norambuena, "Model predictive control for power converters and drives: Advances and trends," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 2, pp. 935–947, 2017.
- [12] P. Cortes, M. P. Kazmierkowski, R. M. Kennel, D. E. Quevedo, and J. Rodriguez, "Predictive control in power electronics and drives," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 12, pp. 4312–4324, 2008.
- [13] P. Karamanakos and T. Geyer, "Guidelines for the design of finite control set model predictive controllers," *IEEE Transactions on Power Electronics*, vol. 35, no. 7, pp. 7434–7450, 2020.
- [14] P. Lezana, R. Aguilera, and D. E. Quevedo, "Model predictive control of an asymmetric flying capacitor converter," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 6, pp. 1839–1846, 2009.
- [15] J. Rodriguez and P. Cortes, *Predictive Control of Power Converters and Electrical Drives*, ser. IEEE Press. Wiley, 2012.
- [16] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992.
- [17] B. Widrow and M. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.
- [18] T. Kamio, S. Tanaka, and M. Morisue, "Backpropagation algorithm for logic oriented neural networks," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 2, 2000, pp. 123–128 vol.2.
- [19] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
- [20] T. Sasaki, K. Kinoshita, S. Kishida, Y. Hirata, and S. Yamada, "Effect of number of input layer units on performance of neural network systems for detection of abnormal areas from x-ray images of chest," in *2011 IEEE 5th International Conference on Cybernetics and Intelligent Systems (CIS)*, 2011, pp. 374–379.
- [21] Z. Shi, "Study on the influence of different number of neuron layers and activation functions on neural networks," in *2021 IEEE 3rd International Conference on Frontiers Technology of Information and Computer (ICFTIC)*, 2021, pp. 101–104.
- [22] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.
- [23] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, no. 10, p. 78–87, oct 2012. [Online]. Available: <https://doi.org/10.1145/2347736.2347755>
- [24] P. Lezana, *Manejo de la Plataforma Digital BRAIn para el control de Accionamientos. UTFSM - DIE*, 2022.