UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Electrónica

IMPLEMENTACIÓN Y CARACTERIZACIÓN DE TIEMPO DE EJECUCIÓN DE TÉCNICAS DE CONTROL PREDICTIVO POR MODELO UTILIZANDO PLATAFORMA DSPACE.

Tesis de Grado presentada por

FRANCISCO TOMÁS ABUSLEME PEÑAFIEL

como requisito parcial para optar al título de

Ingeniero Civil Electrónico

y al grado de

Magíster en Ciencias de la Ingeniería Electrónica

Director de Tesis
DR GONZALO CARVAJAL B.

Valparaíso, 2026.

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☐ Memoria o trabajo de título ☒ Tesis de Postgrado

Título del trabajo: Implementación y caracterización de tiempo de ejecución de técnicas de control predictivo por modelo utilizando plataforma dSPACE.

Nombre del candidato(a): Francisco Tomás Abusleme Peñafiel

Carrera / Grado: Magíster en Ciencias de la Ingeniería Electrónica.

Campus: Casa Central Valparaíso **Departamento:** Departamento de Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Gonzalo Carvajal, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis: Gonzalo Carvajal B.

Fecha: 14/05/2026 **Firma:** 

Estudiante o Candidato(a): Francisco Tomás Abusleme Peñafiel

Fecha: 14/05/2026 **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

AGRADECIMIENTOS

Gracias a mi familia, mis amigos y mi gato.

CONTENIDO

RESUMEN

El Control Predictivo Basado en Modelos (*Model Predictive Control*, MPC) es una técnica avanzada de control que predice el comportamiento futuro de un sistema para optimizar la acción de control en cada intervalo de muestreo. En general, el cálculo de la acción de control óptima en tiempo de ejecución se realiza mediante métodos iterativos o de búsqueda que imponen una alta demanda en términos de número de operaciones y/o utilización de memoria. Esto resulta especialmente desafiante al considerar implementación en plataformas embebidas con recursos limitados, donde la capacidad de cómputo puede ser insuficiente para resolver los problemas de optimización con garantías de tiempo real.

Esta propuesta de tesis aborda la necesidad de evaluar y plantear directrices para reducir la latencia de los lazos MPC. Para abordar este problema, se integrarán aceleradores basados en *Field Programmable Gate Array* (FPGA), aprovechando su capacidad de paralelismo para reducir la latencia del cálculo de la optimización. Además, los aceleradores se integrarán en un lazo de control completo, donde además del cálculo de la actuación óptima, se deben considerar tareas como la adquisición y conversión de señales, la estimación de estados y la comunicación entre dispositivos. Estas tareas adicionales, que tradicionalmente suelen despreciarse al compararlos con los tiempos asociados a resolver el problema de optimización, pueden convertirse en cuellos de botella cuando se requieren lazos de control con alta frecuencia de muestreo. La innovación de este trabajo radica en la validación a nivel de sistema de lazos MPC con aceleración por hardware en entornos realistas, para lo cual se utilizará una plataforma de prototipado rápido dSPACE que permite una rápida implementación y prueba de diferentes configuraciones, facilitando la identificación de cuellos de botella y el balanceo eficiente de recursos considerando el desempeño de control en lazo cerrado en la planta real.

Se espera obtener una evaluación detallada de la latencia de los lazos de control, desde la adquisición de datos hasta la generación de la actuación correspondiente, midiendo los tiempos de ejecución de cada etapa del proceso y proponiendo directrices para optimizar el uso de recursos. Esto es particularmente relevante en contextos con objetivos de latencia que pueden estar en el orden de los microsegundos, por ejemplo en aplicaciones de electrónica de potencia o robótica. Esta investigación proporcionará prototipos funcionales y documentación que faciliten la replicación de los resultados y el uso de la plataforma dSPACE en futuros desarrollos, proporcionando un ejemplo de aplicación real que requiera garantías, y una base sólida y validada para facilitar la adopción de estas tecnologías en sectores industriales como control de motores eléctricos de alta precisión, robótica, sistemas de energías renovables, entre otros.

Índice de figuras

Índice de tablas

Capítulo 1

INTRODUCCIÓN

Este capítulo expone el contexto que justifica la realización de este informe y plantea la problemática que se pretende abordar. Finalmente, se detallan los alcances y las contribuciones de esta tesis.

1.1. Motivación y Contexto

El Control Predictivo por Modelo (Model Predictive Control, MPC) es una técnica avanzada de control que determina la actuación resolviendo, en cada intervalo de muestreo, un problema de optimización formulado a partir de un modelo dinámico del sistema en variables de estado. Este enfoque busca minimizar una función de costo que penaliza tanto el error entre la salida del sistema y una referencia deseada, como el esfuerzo de control requerido para alcanzar dicha referencia, evaluado sobre un horizonte de predicción finito de muestras futuras.

Una de las principales ventajas de MPC respecto de técnicas clásicas, como el control PID, es su capacidad para manejar de forma explícita sistemas multivariables sujetos a restricciones operacionales sobre las entradas, salidas y estados internos. Sin embargo, esta funcionalidad implica un costo computacional considerable, debido a que en cada ciclo de control debe resolverse un problema de optimización con restricciones, lo que puede dificultar su implementación en aplicaciones con alta tasa de muestreo y requerimientos temporales estrictos, sobretodo al implementarse en sistemas embebidos que cuentan con recursos computacionales limitados.

En el caso de sistemas con modelos lineales e invariantes en el tiempo, el problema de optimización resultante se suele formular en la forma de *Quadratic Programming* (QP) [1], considerando una función objetivo cuadrática y restricciones lineales. Dependiendo del mecanismo utilizado para resolver el problema de optimización en cada intervalo de muestreo, las implementaciones de MPC pueden clasificarse en dos enfoques principales: formulaciones implícitas y formulaciones explícitas.

En la formulación implícita, en cada intervalo de muestreo se utiliza el estado actual del sistema para plantear un nuevo problema de optimización, el cual se resuelve de manera *online*, típicamente utilizando algoritmos iterativos que convergen a una aproximación de la solución óptima. Al plantear un nuevo problema en base al estado actual, las formulaciones implícitas permiten al controlador responder a perturbaciones y cambios en las dinámicas del modelo. Por otro lado, resolver el problema de optimización iterativamente impone una demanda computacional significativa. Para abordar esta carga en problemas del tipo QP, se han propuesto algoritmos de resolución eficientes, como el método de Alternating Directions Method of Multipliers (ADMM) [2], el cual presenta buena convergencia en pocas iteraciones, potencial de paralelización de las operaciones, y manejo flexible de restricciones,

tanto para formulaciones densas como *sparse* del problema QP [3].

Por otro lado, la formulación explícita consiste en resolver el problema QP una única vez en tiempo de diseño (*offline*), generando una partición del espacio de estados en regiones politópicas, donde en cada región se define una función afín que determina la acción de control óptima. La ley de control precalculada se almacena en memoria, y durante la ejecución, el controlador únicamente identifica la región correspondiente al estado actual y evalúa la función asociada, lo que elimina la necesidad de resolver un problema de optimización en línea y reduce significativamente la latencia de decisión para problemas de baja dimensionalidad [1]. A pesar de sus potenciales ventajas, las formulaciones explícitas presentan limitaciones prácticas en cuanto a la garantía del cumplimiento de las restricciones, y dificultad en el manejo de incertidumbres y perturbaciones que pueden ocurrir en tiempo de ejecución, y además, la representación estática de la ley de control sufre de una explosión combinatoria en el número de regiones a medida que se incrementan el horizonte de predicción, las dimensiones del sistema y el número de restricciones. Para mitigar esta complejidad, se han propuesto técnicas de reducción como la eliminación de regiones redundantes [4], así como el uso de redes neuronales profundas para aproximar la ley de control, permitiendo disminuir tanto los requerimientos de memoria como la latencia de ejecución [5, 6].

Trabajos recientes desarrollados en el Departamento de Electrónica han reportado aceleradores en FPGA para solvers de MPC, tanto en formulaciones implícitas basadas en ADMM [7] como en formulaciones explícitas que utilizan redes neuronales para aproximar la ley de control [8]. En estos trabajos se informan tiempos de resolución del orden de decenas de microsegundos para el caso implícito y de decenas de nanosegundos para el explícito, validando principalmente la funcionalidad del controlador en entornos simulados. Sin embargo, en un sistema de control real la obtención de la actuación óptima representa solo una parte del ciclo completo, ya que el lazo de control debe interactuar con el entorno físico e incorporar tareas adicionales como adquisición de señales, estimación de estados, sincronización y comunicación. A medida que los solvers especializados alcanzan tiempos de ejecución cada vez menores, las latencias asociadas a estas tareas adquieren mayor relevancia, lo que hace indispensable adoptar una perspectiva de sistema completo para la validación de esquemas MPC, especialmente en sistemas con altas tasas de muestreo. Esta complejidad ha limitado hasta ahora los trabajos del Departamento de Electrónica a validaciones a nivel de *Software-in-the-Loop*, lo que significa que el controlador es simulado utilizando su descripción de hardware en un lenguaje como Verilog.

En el contexto de validación experimental de algoritmos MPC, las plataformas de prototipado rápido representan una herramienta potencialmente valiosa para acelerar el desarrollo y validación a nivel de sistema. Estas plataformas proveen entornos de desarrollo integrados que combinan soporte de hardware con herramientas de diseño a distintos niveles de abstracción, y normalmente incluyen periféricos como conversores análogo-digitales con interfaces configurables, módulos de protección, herramientas de visualización, y soporte para entornos de desarrollo de alto nivel como Matlab y Simulink. Estas características permiten reducir significativamente las barreras de entrada asociadas a la implementación y experimentación. No obstante, el uso de plataformas de prototipado rápido presenta también limitaciones importantes. En muchos casos, estas plataformas están diseñadas para operar exclusivamente con librerías propietarias y bloques precompilados, lo que restringe el control del usuario sobre los procesos subyacentes. Esto puede dificultar la incorporación de aceleradores personalizados, optimizados a bajo nivel, que han sido reportados en la literatura precisamente para maximizar eficiencia en el uso de recursos. Como resultado, la portabilidad de estos desarrollos a plataformas de prototipado puede verse comprometida. Adicionalmente, estas plataformas presentan una curva de aprendizaje pronunciada y la documentación técnica disponible es limitada, en especial en lo relativo a detalles de implementación de esquemas de control predictivo. Esta falta de transparencia dificulta la replicabilidad de resultados y la evaluación comparativa entre diferentes

soluciones propuestas.

Esta tesis propone y valida procedimientos de diseño basado en modelos e integración de lazos MPC implícitos y explícitos acelerados con FPGA en una plataforma de control por prototipado rápido. Específicamente, se considera el uso de MicroLabBox [9] de dSPACE, la cual es ampliamente usada en la industria y en entornos de investigación académica. Se integrarán lazos de control, evaluando su efectividad en una planta física bajo condiciones operativas realistas, lo que significa considerar restricciones y desafíos presentes en un entorno de producción, como la interacción con hardware diverso o variabilidad en las señales de entrada y salida. Se usará la plataforma dSPACE para realizar una caracterización detallada de la latencia del lazo, considerando todas las etapas necesarias desde la adquisición de datos hasta la generación de la actuación correspondiente. Como plantas se considera un servomotor para la validación de los procedimientos y un prototipo de Recursos Energéticos Distribuidos (DER) a escala de laboratorio para una aplicación que se beneficie de la aceleración por hardware. A partir de esto, se busca derivar directrices para realizar un mejor uso de recursos y obtener estimaciones tempranas de desempeño y requerimientos que faciliten la posterior implementación en plataformas embebidas especializadas.

1.2. Planteamiento del problema

Esta tesis aborda como problemática central la necesidad de desarrollar procedimientos para la implementación y caracterización de lazos MPC, con énfasis en la validación funcional del lazo de control luego de integrar aceleradores de hardware con las demás tareas de control necesarias para cerrar el lazo.

Mediante la utilización de la plataforma MicroLabBox de dSPACE, se obtendrán mediciones precisas de los tiempos de procesamiento para diversas implementaciones de MPC, tanto implícitas como explícitas. Esto proporcionará directrices para alcanzar latencias objetivo en el orden de los microsegundos, facilitando la selección de períodos adecuados que garanticen la prevención de excedencias, mientras se consideran las variaciones en los tiempos de ejecución (*jitter*) y sus incertidumbres asociadas. La validación del flujo de diseño se realizará hasta nivel físico, empleando observadores de estado con acción integral en un motor DC, donde se experimentará con diferentes restricciones y representaciones numéricas, tanto en punto fijo como flotante. Posteriormente, este flujo se aplicará hasta el nivel de SIL en una aplicación real de electrónica de potencia que requiere garantías específicas de funcionamiento.

Para comprender mejor el proceso, la Figura 1.1 ilustra el esquema general de las etapas que constituyen un lazo MPC en una implementación práctica con observadores de estado. El proceso inicia con la fase de sensado, donde se captura y digitaliza la señal analógica mediante un conversor análogico-digital. Posteriormente, se ejecuta la fase de actuación, que transforma la señal digital en una acción física que actúa directamente sobre el sistema a través de un conversor digital-análogico. Esta secuencia se diseñó específicamente para evitar interferencias durante el muestreo. El sistema incorpora, además del cálculo de la actuación óptima, un sistema de observadores que opera en dos fases: primero, realiza una estimación de los estados no medibles antes del cálculo, y segundo, genera una predicción de la siguiente salida después del cálculo, la cual servirá como base para la estimación en el ciclo siguiente. Es importante destacar que en este esquema, la actuación precede al cálculo, utilizando la actuación determinada en el ciclo anterior.

Este enfoque va más allá del mero cálculo de optimización, incorporando aspectos críticos como la adquisición y conversión de señales, la estimación de estados y la comunicación entre dispositivos. Si bien los recientes avances en aceleradores de hardware y algoritmos de optimización han logrado reducir significativamente los tiempos de ejecución en los cálculos de optimización, existe aún una

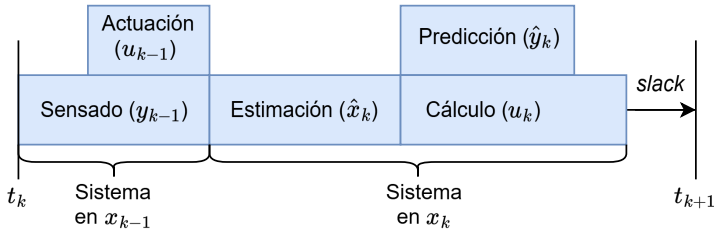


Fig. 1.1: Diagrama temporal de las etapas de un lazo MPC con observadores.

carencia notable en la validación práctica que permita medir con precisión la latencia total del lazo de control y sus etapas específicas, desde la adquisición inicial de datos hasta la aplicación final de la acción de control. Aunque la implementación de MPC en plataformas de prototipado rápido como dSPACE facilita la integración de estos avances, todavía no se ha realizado una caracterización exhaustiva de cómo cada componente del lazo influye en la latencia total bajo diferentes configuraciones de implementación.

1.3. Alcances y contribuciones

Esta tesis considera los siguientes alcances:

- No se hacen innovaciones en los algoritmos existentes, sino que se desarrollan y validan procedimientos para explorar y validar implementaciones para controlar sistemas físicos en un entorno con ruido y perturbaciones que pueden ya estar planteadas a nivel de simulación.

Las contribuciones de esta tesis son las siguientes:

- Validación y análisis temporal detallado de lazos MPC implícitos y explícitos con aceleración por hardware, controlando plantas físicas en entornos con ruido y perturbaciones.
- Documentación para el flujo de trabajo y proceso de implementación en la MicroLabBox. Esta documentación y los códigos utilizados quedarán disponibles, facilitando su reproducibilidad.
- Entregar directrices para diseñar un lazo MPC considerando restricciones, uso de recursos y garantías de tiempo real.
- Aplicación de los procedimientos propuestos para acelerar el lazo de control de un problema real asociado a electrónica de potencia que requiere garantías de latencia.

1.4. Organización del documento

El contenido que sigue en este informe está organizado de la siguiente forma:

- Capítulo 2: Se explica la teoría necesaria para entender las implementaciones. Esto incluye teoría general de control, el desarrollo matemático de MPC como problema de optimización, los métodos a utilizar tanto para MPC implícito como explícito y una explicación general de las herramientas relevantes.

- Capítulo 3: Se exponen detalladamente los flujos de diseño que se utilizan para las implementaciones de MPC implícito y explícito.
- Capítulo 4: Se validan los flujos de diseño en un servomotor, realizando exploraciones de recursos y análisis temporales de los lazos de control.
- Capítulo 5: Se aplica el flujo de diseño de MPC explícito hasta Software-in-the-Loop para una planta más compleja, específicamente un DER.
- Capítulo 6: Se resume lo logrado en este trabajo de tesis y se proponen direcciones de trabajo futuro.

Capítulo 2

ANTECEDENTES

Este capítulo presenta el marco teórico sobre el cual se construye esta tesis. La primera sección introduce conceptos fundamentales de MPC, comenzando por el problema general de control con observadores de estado incluidos, para luego plantear el problema de optimización asociado a MPC como un problema de programación cuadrática (QP). Se explica además como resolver la formulación implícita de MPC utilizando ADMM para resolver el problema QP, y la formulación explícita utilizando redes neuronales artificiales para aproximar la ley de control. En la última sección se describe brevemente qué es una FPGA y en qué consiste un plataforma de prototipado rápido, entregando detalles de la plataforma dSPACE con la cual se implementarán los lazos de control.

2.1. Control Predictivo por Modelo

El control predictivo basado en modelos (Model Predictive Control, MPC) es una técnica de control avanzada que se basa en utilizar un modelo en variables de estado de una planta para predecir su comportamiento a lo largo de un horizonte finito N de futuros instantes de muestreo. En cada instante k, la acción de control u_k se determina mediante la minimización de una función de costo que pondera el error de la salida y la magnitud de la entrada a lo largo del horizonte. El resultado de esta optimización es una secuencia de acciones óptimas, de las cuales solo se implementa la primera en el sistema. Este enfoque, conocido como *receding horizon*, implica que el problema de optimización se resuelve nuevamente en el siguiente instante, desplazando el horizonte hacia adelante considerando el nuevo instante inicial.

En general las implementaciones de MPC pueden categorizarse como implícitas o explícitas. Las formulaciones implícitas resuelven el problema de optimización para el estado actual del sistema de manera *online*, es decir, en tiempo real y continuamente mientras el sistema está en operación. Por otro lado, las formulaciones explícitas resuelven el problema de manera *offline*, esto significa que las soluciones para los posibles estados del sistema se resuelven de antemano y se almacenan en memoria, por lo que en tiempo de ejecución debe efectuarse un proceso de búsqueda de la solución.

Una de las principales ventajas de MPC en relación a técnicas de control clásicas como PID, es que permite el control de sistemas multivariables considerando explícitamente restricciones operacionales en los valores de entradas, salidas, y estados internos del sistema a controlar.

2.1.1. Formulación del problema de control

Para la formulación del problema de control en tiempo discreto, se utilizará un modelo determinístico, lineal e invariante en el tiempo de la planta a controlar. Esto permite expresar las

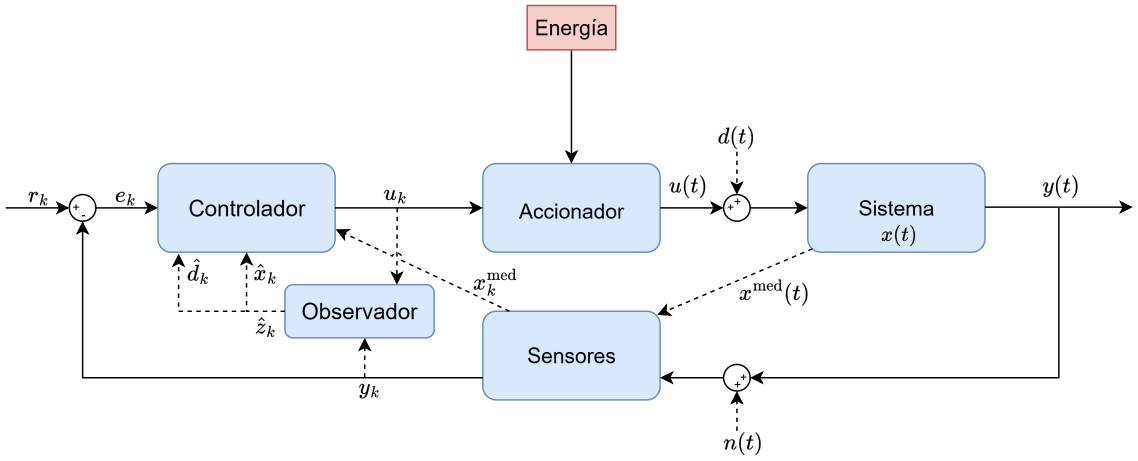


Fig. 2.1: Esquema de control digital.

ecuaciones del sistema a controlar en variables de estado discreto, que considera n estados, m entradas y p salidas discretizadas. Si no se consideran perturbaciones, para cada instante k los estados $x_k \in \mathbb{R}^n$, entradas $u_k \in \mathbb{R}^m$ y las salidas $y_k \in \mathbb{R}^p$ se relacionan mediante las ecuaciones:

$$x_{k+1} = \mathbf{A}x_k + \mathbf{B}u_k \tag{2.1.1}$$
$$y_k = \mathbf{C}x_k + \mathbf{D}u_k, \tag{2.1.2}$$

donde las matrices $\mathbf{A} \in \mathbb{R}^{n\times n}, \mathbf{B} \in \mathbb{R}^{n\times m}, \mathbf{C} \in \mathbb{R}^{p\times n}, \mathbf{D} \in \mathbb{R}^{p\times m}$ representan las dinámicas del sistema. La matriz $\mathbf{D}$ se denomina matriz de paso directo, y se asumirá como cero. Se asume también que los estados pertenecen a un conjunto $\mathcal{X} \subseteq \mathbb{R}^n$ de estados factibles y las actuaciones a un conjunto $\mathcal{U} \subseteq \mathbb{R}^m$ de actuaciones factibles.

Es importante mencionar que aunque el modelo utilizado del sistema sea en tiempo discreto, en general las dinámicas del sistema son en tiempo continuo. La Figura 2.1 presenta el esquema general de control discreto, donde las salidas del sistema (o a veces también los estados) son medidas y discretizadas a través de sensores, para luego resolver el problema de control en algún dispositivo. El objetivo del control es llevar la salida del sistema $y(t)$ a una referencia $r(t)$, que discretizada se representa como r_k. En base a las mediciones, estimaciones y la referencia r_k, el controlador determina una actuación u_k, la cual está representada digitalmente. Mediante un accionador que requiere energía, esta señal digital es convertida a una cantidad física $u(t)$ (por ejemplo, voltaje) , que es la que finalmente interactúa con el sistema físico. Además, las líneas punteadas representan elementos del lazo que a menudo se omiten en representaciones simplificadas. Entre ellos se encuentran las perturbaciones de entrada $d(t)$ y de salida $n(t)$, las suelen despreciarse aunque están presentes en sistemas reales. Asimismo, se consideran la medición y estimación de algunos estados internos del sistema, denotados como x_k^{med} y $\hat{x}_k$, respectivamente. Si bien algunos esquemas de control se basan únicamente en las salidas para compararlas con la referencia, en el caso de MPC es necesario disponer de un modelo del sistema que permita predecir los estados futuros. Dado que no todos los estados $x(t)$ son directamente medibles (o resulta poco conveniente hacerlo), suele recurrirse al uso de observadores de estado para su estimación.

2.1.2. Observadores de estado

Para realizar control predictivo basado en modelos, es necesario contar con un modelo del sistema a controlar para hacer las predicciones de los estados siguientes. Estas predicciones utilizan los estados del sistema, pero en general no todos los estados x_k son medibles (o económicamente viable hacerlo), por lo que puede ser necesario utilizar estimación de estados. En esta sección se comienza formulando un observador de estado básico, para luego incluir perturbaciones de entrada en un observador extendido.

Observador de estado básico

Consideraremos un sistema en variables de estado discreto de la siguiente forma:

$$x_{k+1} = \mathbf{A}x_k + \mathbf{B}u_k \tag{2.1.3}$$
$$y_k = \mathbf{C}x_k \tag{2.1.4}$$

Para la formulación de un observador de estado básico (o de Luenberger), comenzamos por definir el estado estimado $\hat{x}_k \in \mathbb{R}^n$ y la salida estimada $\hat{y}_{k+1} \in \mathbb{R}^p$:

$$\hat{x}_k = \mathbf{A}\hat{x}_{k-1} + \mathbf{B}u_{k-1} + \mathbf{L}_x(y_{k-1} - \hat{y}_{k-1}) \tag{2.1.5}$$
$$\hat{y}_k = \mathbf{C}\hat{x}_k, \tag{2.1.6}$$

donde la matriz de ganancia del observador $\mathbf{L}_x \in \mathbb{R}^{n\times p}$ es una matriz de ganancia que debe diseñarse y que corrige la estimación de los estados según el error observado en la salida. Para analizar este error, se define el error de estimación de los estados:

$$\tilde{x}_k = x_k - \hat{x}_k, \tag{2.1.7}$$

obteniendo con esto:

$$\tilde{x}_{k+1} = \mathbf{A}x_k + \mathbf{B}u_k - \mathbf{A}\hat{x}_k - \mathbf{B}u_k - \mathbf{L}_x(y_k - \hat{y}_k) \tag{2.1.8}$$
$$= \mathbf{A}x_k - \mathbf{A}\hat{x}_k - \mathbf{L}_x(\mathbf{C}x_k - \mathbf{C}\hat{x}_k) \tag{2.1.9}$$
$$= (\mathbf{A} - \mathbf{L}_x\mathbf{C})\tilde{x}_k. \tag{2.1.10}$$

Esta última ecuación implica que para garantizar que $\tilde{x}_k \to 0$, se debe diseñar la matriz de ganancia del observador de tal manera que la matriz $\mathbf{A} - \mathbf{L}_x\mathbf{C}$ tenga sus autovalores dentro del círculo unitario complejo abierto (véase [10], Capítulo 4, para un tratamiento detallado del diseño de observadores y la colocación de polos en sistemas discretos.).

Observador aumentado

En la práctica, el sistema puede verse afectado por perturbaciones de entrada $d(t)$ y/o perturbaciones de salida $n(t)$, como se ilustra en la Figura 2.1. Para incorporarlas en el modelo en espacio de estados discretizado, se consideran sus versiones discretas. Al incluir una perturbación de entrada $d_k \in \mathbb{R}^m$ y una perturbación de salida $n_k \in \mathbb{R}^p$, las ecuaciones del sistema se modifican de la siguiente forma:

$$x_{k+1} = \mathbf{A}x_k + \mathbf{B}u_k + \mathbf{B}d_k \tag{2.1.11}$$
$$y_k = \mathbf{C}x_k + n_k \tag{2.1.12}$$

Consideraremos en particular cuando existe una perturbación de entrada constante, es decir que cumple:

$$d_{k+1} = d_k, \tag{2.1.13}$$

la cual se incluye en un estado aumentado $z_k \in \mathbb{R}^{n+m}$:

$$z_k = \begin{bmatrix} x_k \\ d_k \end{bmatrix}, \tag{2.1.14}$$

con el cual se plantea un sistema aumentado como:

$$\begin{bmatrix} x_{k+1} \\ d_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0}_{m\times n} & \mathbf{1}_{m\times m} \end{bmatrix}}_{\mathbf{A}_z} \begin{bmatrix} x_k \\ d_k \end{bmatrix} + \underbrace{\begin{bmatrix} \mathbf{B} \\ \mathbf{0}_{m\times m} \end{bmatrix}}_{\mathbf{B}_z} u_k \tag{2.1.15}$$

$$y_k = \underbrace{\begin{bmatrix} \mathbf{C} & \mathbf{0}_{p\times m} \end{bmatrix}}_{\mathbf{C}_z} \begin{bmatrix} x_k \\ d_k \end{bmatrix}. \tag{2.1.16}$$

Luego la estimación del estado aumentado queda definida por:

$$\hat{z}_k = \mathbf{A}_z\hat{z}_{k-1} + \mathbf{B}_z u_{k-1} + \mathbf{L}_z(y_{k-1} - \hat{y}_{k-1}) \tag{2.1.17}$$
$$\hat{y}_k = \mathbf{C}_z\hat{z}_k, \tag{2.1.18}$$

donde la matriz $\mathbf{A}_z - \mathbf{L}_z\mathbf{C}_z$ tiene sus autovalores dentro del círculo unitario complejo abierto(véase [10], Capítulo 4, para un tratamiento detallado del diseño de observadores y la colocación de polos en sistemas discretos). La estimación $\hat{d}_k$ se utiliza para modificar la entrada a aplicar al sistema (ver Figura 2.1). La ecuación (2.1.32) considera cómo la perturbación afecta a los estados y la actuación en estado estacionario, y la expresión (2.1.35) muestra cómo modificar la actuación aplicada considerando la perturbación estimada.

Etapas del lazo de control con observadores

Considerando las estimaciones de estados recién planteadas, el lazo de control estaría compuesto por las etapas que se muestran en la Figura 2.2, y que se detallan a continuación:

1. Sensado: El sistema se encuentra en el estado x_{k-1} y se realizan las mediciones de la salida y_{k-1}.
2. Actuación: Se aplica la entrada u_{k-1} calculada en la iteración anterior, y ahora el sistema se encuentra en el estado x_k.

3. Estimación: Se realiza la estimación $\hat{x}_k$ del nuevo estado. Notar que se cuenta con la estimación de la salida $\hat{y}_{k-1}$ realizada en la iteración anterior.

4. Cálculo: Con la información disponible, se realiza el cálculo de la actuación u_k que se aplicará en la siguiente iteración.

5. Predicción: Se realiza la estimación $\hat{y}_k$ de la siguiente salida a medir.

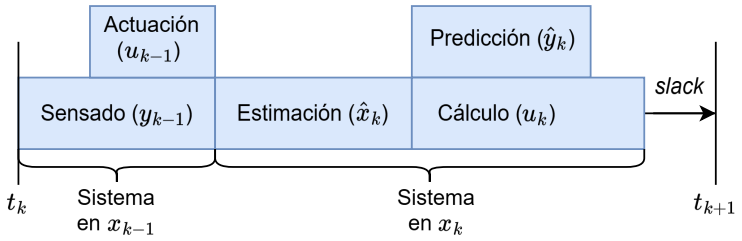


Fig. 2.2: Diagrama temporal de las etapas de un lazo MPC típico con observadores.

En la Figura 2.2 se muestra que el sensado comienza antes de la actuación para que esta no interfiera con la medición y el sistema se estabilice mientras se realizan los cálculos posteriores. Además, el cálculo de la actuación y la predicción pueden hacerse paralelamente, y en el diagrama se asume que el cálculo de la predicción demora menos tiempo que el cálculo de la actuación.

2.1.3. Problema de optimización

Los problemas MPC que cuentan con restricciones lineales y un modelo lineal e invariante en el tiempo de la planta, pueden plantearse como problemas de programación cuadrática (QP), que consiste en un problema de optimización con una función de costo cuadrática y restricciones lineales.

Funcional de costo

El funcional de costo asociado a este problema de optimización pondera el error en la salida y la magnitud de la entrada en cada instante del horizonte. Para definir la función de costo, se define el error $e_k \in \mathbb{R}^p$ como la diferencia entre la referencia r_k y la salida y_k:

$$e_k = r_k - y_k. \tag{2.1.19}$$

Por simplicidad del análisis, se comienza suponiendo que la referencia es $r_k = 0$, para luego agregar referencias no nulas una vez ya planteado el problema. Se considera el error cuadrático para la función de costo preliminar J_k^{pre}:

$$J_k^{\text{pre}} = e_k^{\mathrm{T}} e_k = y_k^{\mathrm{T}} y_k, \tag{2.1.20}$$

en la cual al utilizar la expresión para la salida en (2.1.4) se obtiene:

$$J_k^{\text{pre}} = x_k^{\mathrm{T}} \underbrace{\mathbf{C}^{\mathrm{T}}\mathbf{C}}_{\mathbf{\Omega}} x_k = x_k^{\mathrm{T}} \mathbf{\Omega} x_k, \tag{2.1.21}$$

donde $\mathbf{\Omega} \in \mathbb{R}^{n\times n}$ es la matriz de pesos que pondera los estados en la función de costo. Para plantear la función de costo completa, se considera además la magnitud de las actuaciones, ya que al implementar un algoritmo de control deben considerarse factores el desgaste de los componentes, seguridad de operación, consumo de energía, entre otros. Se introduce la matriz de pesos $\mathbf{\Gamma} \in \mathbb{R}^{m\times m}$ para ponderar las entradas en la función de costo:

$$J_k = \alpha \cdot x_k^{\mathrm{T}} \mathbf{\Omega} x_k + u_k^{\mathrm{T}} \mathbf{\Gamma} u_k, \tag{2.1.22}$$

donde $\alpha > 0$ es un factor de regularización que pondera los elementos de la función de costo. Al elegir $\alpha < 1$, se disminuye la magnitud de los picos en la actuación si alguna señal de control, como por ejemplo la referencia, cambia de un instante a otro. Por simplicidad, en el resto del desarrollo se considera que $\alpha = 1$.

Para MPC se considera el costo asociado a todas las futuras entradas dentro del horizonte de predicción N, por lo que la función de costo total resultante es:

$$J(x_0, \vec{x}, \vec{u}) = \sum_{k=0}^{N-1} \left(x_k^{\mathrm{T}} \mathbf{\Omega} x_k + u_k^{\mathrm{T}} \mathbf{\Gamma} u_k \right) + x_N^{\mathrm{T}} \mathbf{\Omega}_N x_N, \tag{2.1.23}$$

donde $\vec{u} = [u_0^{\mathrm{T}}, ..., u_{N-1}^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{m\cdot N}$ es un vector que contiene la secuencia de actuaciones dentro del horizonte de predicción y $\vec{x} = [x_1^{\mathrm{T}}, ..., x_N^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{n\cdot N}$ es el vector que contiene la secuencia de estados. Las matrices de pesos son elegidas tales que son semi-definidas positivas, y la matriz $\mathbf{\Omega}_N$ se elige resolviendo la ecuación de Riccati para LQR, con el fin de aproximar el comportamiento de un lazo MPC de horizonte infinito.

Restricciones

En cuanto a las restricciones, estas se consideran como lineales para los estados y actuaciones, es decir, los conjuntos factibles se expresan como:

$$\mathcal{X} = \{x \in \mathbb{R}^n : \mathbf{C_x} x \leq c_x\}, \tag{2.1.24}$$

$$\mathcal{U} = \{u \in \mathbb{R}^n : \mathbf{C_u} u \leq c_u\}, \tag{2.1.25}$$

donde $\mathbf{C_x} \in \mathbb{R}^{r_x \times n}, c_x \in \mathbb{R}^{r_x}, \mathbf{C_u} \in \mathbb{R}^{r_u \times m}, c_u \in \mathbb{R}^{r_u}$ siendo r_x y r_u el número de restricciones por vector de estados y por vector entradas respectivamente.

Problema QP

Considerando la función de costo planteada y las restricciones, el problema de optimización para encontrar la secuencia de entradas óptimas $\vec{u}^* = [u_0^{*\mathrm{T}}, ..., u_{N-1}^{*\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{m\cdot N}$ puede plantearse como

$$\begin{aligned} \min_{\vec{u}} \quad & \sum_{k=0}^{N-1} \left(x_k^{\mathrm{T}} \mathbf{\Omega} x_k + u_k^{\mathrm{T}} \mathbf{\Gamma} u_k \right) + x_N^{\mathrm{T}} \mathbf{\Omega}_N x_N \\ \text{sujeto a} \quad & x_{k+1} = \mathbf{A} x_k + \mathbf{B} u_k, \\ & \mathbf{C_x} x_{k+1} \leq c_x, \\ & \mathbf{C_u} u_k \leq c_u, \\ & \forall k = 0, ..., N-1. \end{aligned} \tag{2.1.26}$$

Para obtener restricciones de tipo caja, se pueden elegir las matrices $\mathbf{C_x}$ y $\mathbf{C_u}$ utilizando matrices identidad como $\mathbf{C_x} = [\mathbf{I}_{n\times n}, -\mathbf{I}_{n\times n}]^{\mathrm{T}}$ y $\mathbf{C_u} = [\mathbf{I}_{m\times m}, -\mathbf{I}_{m\times m}]^{\mathrm{T}}$ respectivamente, y los vectores c_x y c_u como $c_x = [x_{max}, -x_{min}]^{\mathrm{T}}$ y $c_u = [u_{max}, -u_{min}]^{\mathrm{T}}$ respectivamente. Para este tipo de restricciones, se tiene que $r_x = 2n$ y $r_u = 2m$.

En esta formulación, x_0 es el vector de estado inicial, el cual representa el vector de estado actual del sistema y por lo tanto puede cambiar luego de cada iteración. Es importante destacar que esto requiere conocer los estados actuales del sistema, ya sea midiéndolos directamente o estimándolos, por ejemplo, mediante observadores. Por el momento, el vector de estado inicial es la única información no constante necesaria para resolver el problema de optimización; sin embargo, a continuación se incluirá también la referencia como parte del problema de optimización.

2.1.4. Seguimiento de referencia y consideración de perturbación de entrada

En la formulación del problema de optimización (2.1.26) se asume la referencia como cero, y por lo tanto el estado inicial x_0 es la única información no constante que se utiliza para resolver el problema. Para incluir referencias no nulas y lograr que la salida en estado estacionario tienda a la referencia (es decir $y_\infty \to r$), se reescriben (2.1.11) y (2.1.12) en estado estacionario para, considerando un error de salida despreciable y una perturbación de entrada constante:

$$\begin{aligned} x_\infty &= \mathbf{A}x_\infty + \mathbf{B}u_\infty + \mathbf{B}\hat{d} \\ y_\infty &= \mathbf{C}x_\infty, \end{aligned} \tag{2.1.27}$$

donde x_∞ representa a los estados en estado estacionario y u_∞ a la actuación en estado estacionario. Notar que si se hace el cambio de variable $\bar{y}_k = y_k - y_\infty$, el problema de control se vuelve equivalente a un problema con referencia cero. Para replantear el problema de esta manera, se restan las ecuaciones de estado de las ecuaciones en estado estacionario (dado que la perturbación está modelada como constante, se considera que $\hat{d}_k = \hat{d}$):

$$\begin{aligned} x_{k+1} - x_\infty &= \mathbf{A}(x_k - x_\infty) + \mathbf{B}(u_k - u_\infty) \\ y_k - y_\infty &= \mathbf{C}(x_k - x_\infty) \end{aligned} \tag{2.1.28}$$

Se observa que el problema con referencia es equivalente a un problema de referencia cero cuando se considera un cambio de variable para los estados y las actuaciones:

$$\bar{x}_k = x_k - x_\infty \tag{2.1.29}$$

$$\bar{u}_k = u_k - u_\infty \tag{2.1.30}$$

Para encontrar x_∞ y u_∞ se reescriben las ecuaciones en estado estacionario como sistema de ecuaciones matricial y se reemplaza y_∞ por la referencia r:

$$\underbrace{\begin{bmatrix} \mathbf{A} - \mathbf{I}_{n\times n} & \mathbf{B} \\ \mathbf{C} & \mathbf{0}_{p\times m} \end{bmatrix}}_{\mathbf{T}} \begin{bmatrix} x_\infty \\ u_\infty \end{bmatrix} = \begin{bmatrix} -\mathbf{B}\hat{d} \\ r \end{bmatrix} \tag{2.1.31}$$

$$\implies \begin{bmatrix} x_\infty \\ u_\infty \end{bmatrix} = \mathbf{T}^{-1} \begin{bmatrix} -\mathbf{B}\hat{d} \\ r \end{bmatrix}, \tag{2.1.32}$$

donde la matriz $\mathbf{T} \in \mathbb{R}^{(n+p)\times(n+m)}$ se asume invertible. Este sistema de ecuaciones permite calcular los vectores x_∞ y u_∞ en función de la referencia deseada. Luego, si consideramos el sistema con las ecuaciones

$$\begin{aligned} \bar{x}_{k+1} &= \mathbf{A}\bar{x}_k + \mathbf{B}\bar{u}_k \\ \bar{y}_k &= \mathbf{C}\bar{x}_k, \end{aligned} \tag{2.1.33}$$

se puede repetir el desarrollo utilizado para plantear el problema de optimización (2.1.26). De esta manera, se llega al problema

$$\begin{aligned} \min_{\vec{u}} \quad & \sum_{k=0}^{N-1} \left(\bar{x}_k^{\mathrm{T}}\mathbf{\Omega}\bar{x}_k + \bar{u}_k^{\mathrm{T}}\mathbf{\Gamma}\bar{u}_k\right) + \bar{x}_N^{\mathrm{T}}\mathbf{\Omega}_N\bar{x}_N \\ \text{sujeto a} \quad & \bar{x}_{k+1} = \mathbf{A}\bar{x}_k + \mathbf{B}\bar{u}_k, \\ & \mathbf{C_x}\bar{x}_{k+1} \leq c_x - \mathbf{C_x}x_\infty, \\ & \mathbf{C_u}\bar{u}_k \leq c_u - \mathbf{C_u}u_\infty, \\ & \forall k = 0, ..., N-1. \end{aligned} \tag{2.1.34}$$

Es importante destacar que este problema de optimización encuentra la secuencia de entradas desviadas óptimas, por lo que la siguiente actuación a aplicar es:

$$u_0^* = \bar{u}_0^* + u_\infty. \tag{2.1.35}$$

2.1.5. Formulación densa del problema QP

El objetivo ahora es plantear la formulación QP densa de un problema MPC, a partir del planteamiento en variables de estado del sistema y del problema de optimización obtenido en la sección anterior. Al final de la sección, los problemas expresados en (2.1.26) y en (2.1.34) quedan expresados como problemas QP estándar. Por simplicidad se utiliza la notación del primero, pero estos problemas (sin y con referencia) tienen la misma forma y por lo tanto el desarrollo puede aplicarse a ambos. Un problema QP estándar tiene la siguiente forma:

$$\begin{aligned} \min_{\vec{u}} \quad & \frac{1}{2}\vec{u}^{\mathrm{T}}\mathbf{Q}\vec{u} + \vec{q}^{\mathrm{T}}\vec{u} \\ \text{sujeto a} \quad & \mathbf{R}\vec{u} \leq \vec{c}. \end{aligned} \tag{2.1.36}$$

Formulación densa vs formulación *sparse*

Para formular un problema MPC como un problema QP, este puede representarse en una de dos formas: densa o *sparse*. La formulación densa utiliza como variables de decisión únicamente las acciones de control $\vec{u}$, mientras que la formulación *sparse* incluye también los estados $\vec{x}$ como parte de las variables de decisión. Dependiendo de la formulación elegida, el número de variables de decisión N_{QP} y el número total de restricciones M_{QP} se determinan según las siguientes expresiones:

$$\begin{aligned} \textbf{Densa:} \quad & N_{QP} = Nm \quad && \text{y} \quad M_{QP} = N(r_x + r_u) \\ \textbf{Sparse:} \quad & N_{QP} = N(n+m) \quad && \text{y} \quad M_{QP} = N(r_x + r_u) \end{aligned}$$

Aunque la formulación *sparse* del problema sea de mayor dimensión que la formulación densa, tiene la ventaja de que la matriz $\mathbf{Q}$ tiene un elevado número de valores nulos. Para aprovechar esto, se requiere el uso de estructuras de datos especializadas que compriman las matrices y eviten almacenar y realizar operaciones con los valores nulos [11].

En este trabajo se utilizará la formulación densa, debido a que resulta en un problema de dimensión menor sin la necesidad de utilizar estructuras de datos especializadas, lo cual puede ser más desafiante a nivel de implementación. A continuación se desarrolla matemáticamente la formulación densa para los problemas de optimización planteados en (2.1.26) y en (2.1.34), utilizando la notación del primero por simplicidad.

Restricciones de evolución de estados

Se comienza por expresar las restricciones (de igualdad) de la evolución de los estados y el funcional de costo de manera más compacta, utilizando la misma notación vectorial que en (2.1.23) para los estados y actuaciones. Para la evolución de los estados se reemplaza iterativamente la ecuación del estado anterior en el siguiente, obteniendo:

$$\vec{x} = \underbrace{\begin{bmatrix} \mathbf{A} \\ \mathbf{A}^2 \\ \vdots \\ \mathbf{A}^{N-1} \\ \mathbf{A}^N \end{bmatrix}}_{\mathbf{E}} x_0 + \underbrace{\begin{bmatrix} \mathbf{B} & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{AB} & \mathbf{B} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{A}^{N-2}\mathbf{B} & \mathbf{A}^{N-3}\mathbf{B} & \dots & \dots & \ddots & \mathbf{0} \\ \mathbf{A}^{N-1}\mathbf{B} & \mathbf{A}^{N-2}\mathbf{B} & \dots & \dots & \mathbf{AB} & \mathbf{B} \end{bmatrix}}_{\mathbf{F}} \vec{u} \tag{2.1.37}$$

$$= \mathbf{E}x_0 + \mathbf{F}\vec{u}, \tag{2.1.38}$$

donde $\mathbf{E} \in \mathbb{R}^{(n\cdot N)\times n}$ y $\mathbf{F} \in \mathbb{R}^{(n\cdot N)\times(m\cdot N)}$, y cada elemento $\mathbf{0}$ en la matriz $\mathbf{F}$ representa a la matriz $\mathbf{0}_{n\times m}$.

Funcional de costo

En cuanto al funcional de costo, este se expresa como:

$$J\left(x_0, \vec{x}, \vec{u}\right) = x_0^{\mathrm{T}}\mathbf{\Omega}x_0 + \vec{x}^{\mathrm{T}} \underbrace{\begin{bmatrix} \mathbf{\Omega} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{\Omega} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \mathbf{\Omega}_N \end{bmatrix}}_{\mathbf{G}} \vec{x} + \vec{u}^{\mathrm{T}} \underbrace{\begin{bmatrix} \mathbf{\Gamma} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{\Gamma} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \dots & \mathbf{\Gamma} \end{bmatrix}}_{\mathbf{H}} \vec{u} \tag{2.1.39}$$

$$= x_0^{\mathrm{T}}\mathbf{\Omega}x_0 + \vec{x}^{\mathrm{T}}\mathbf{G}\vec{x} + \vec{u}^{\mathrm{T}}\mathbf{H}\vec{u}, \tag{2.1.40}$$

donde $\mathbf{G} \in \mathbb{R}^{(n\cdot N)\times(n\cdot N)}$ y $\mathbf{H} \in \mathbb{R}^{(m\cdot N)\times(m\cdot N)}$, y cada elemento $\mathbf{0}$ en las matrices $\mathbf{G}$ y $\mathbf{H}$ representa a la matrices $\mathbf{0}_{n\times n}$ y $\mathbf{0}_{m\times m}$ respectivamente.

Para expresar el funcional de costo solo en función del estado inicial y las actuaciones, se reemplaza la expresión (2.1.38) en (2.1.40), obteniendo:

$$J(x_0, \vec{u}) = x_0^{\mathrm{T}} \mathbf{\Omega} x_0 + (\mathbf{E}x_0 + \mathbf{F}\vec{u})^{\mathrm{T}} \mathbf{G}(\mathbf{E}x_0 + \mathbf{F}\vec{u}) + \vec{u}^{\mathrm{T}} \mathbf{H} \vec{u} \tag{2.1.41}$$

$$= x_0^{\mathrm{T}} \mathbf{\Omega} x_0 + x_0^{\mathrm{T}} \mathbf{E}^{\mathrm{T}} \mathbf{G} \mathbf{E} x_0 + x_0^{\mathrm{T}} \mathbf{E}^{\mathrm{T}} \mathbf{G} \mathbf{F} \vec{u} + \vec{u}^{\mathrm{T}} \mathbf{F}^{\mathrm{T}} \mathbf{G} \mathbf{E} x_0 + \vec{u}^{\mathrm{T}} \mathbf{F}^{\mathrm{T}} \mathbf{G} \mathbf{F} \vec{u} + \vec{u}^{\mathrm{T}} \mathbf{H} \vec{u} \tag{2.1.42}$$

$$= x_0^{\mathrm{T}} (\mathbf{\Omega} + \mathbf{E}^{\mathrm{T}} \mathbf{G} \mathbf{E}) x_0 + x_0^{\mathrm{T}} \mathbf{E}^{\mathrm{T}} \mathbf{G} \mathbf{F} \vec{u} + \vec{u}^{\mathrm{T}} \mathbf{F}^{\mathrm{T}} \mathbf{G} \mathbf{E} x_0 + \vec{u}^{\mathrm{T}} (\mathbf{F}^{\mathrm{T}} \mathbf{G} \mathbf{F} + \mathbf{H}) \vec{u}. \tag{2.1.43}$$

En esta última expresión, el primer término es una constante que no depende de las variables de decisión, por lo que se puede remover del funcional de costo. Además, dado que la matriz $\mathbf{\Omega}$ es simétrica (y asumiendo que $\mathbf{\Omega}_N$ también lo es), la matriz $\mathbf{E}$ también es simétrica, por lo que los términos intermedios son el traspuesto uno del otro, y considerando que son escalares, se concluye que estos términos son iguales. Por lo tanto, el funcional de costo finalmente se expresa como:

$$J(x_0, \vec{u}) = \vec{u}^{\mathrm{T}} (\mathbf{F}^{\mathrm{T}} \mathbf{G} \mathbf{F} + \mathbf{H}) \vec{u} + 2 x_0^{\mathrm{T}} \mathbf{E}^{\mathrm{T}} \mathbf{G} \mathbf{F} \vec{u}. \tag{2.1.44}$$

Esta expresión ya tiene la forma del funcional de costo de un problema QP en forma estándar con variables de decisión $\vec{u}$, donde la matriz $\mathbf{Q} \in \mathbb{R}^{(m \cdot N) \times (m \cdot N)}$ y el vector $\vec{q} \in \mathbb{R}^{m \cdot N}$ quedan definidos por las expresiones:

$$\frac{1}{2}\mathbf{Q} = \mathbf{F}^{\mathrm{T}} \mathbf{G} \mathbf{F} + \mathbf{H}, \tag{2.1.45}$$

$$\vec{q}^{\mathrm{T}} = 2 x_0^{\mathrm{T}} \mathbf{E}^{\mathrm{T}} \mathbf{G} \mathbf{F}. \tag{2.1.46}$$

Restricciones de desigualdad

Para expresar las restricciones de desigualdad de manera más compacta, se definen las matrices $\mathbf{R}_x \in \mathbb{R}^{(r_x \cdot N) \times (n \cdot N)}, \mathbf{R}_u \in \mathbb{R}^{(r_u \cdot N) \times (m \cdot N)}$ y los vectores $\vec{c_x} \in \mathbb{R}^{r_x \cdot N}, \vec{c_u} \in \mathbb{R}^{r_u \cdot N}$ mediante las expresiones:

$$\underbrace{\begin{bmatrix} \mathbf{C}_x & \dots & \mathbf{0}_{r_x \times n} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{r_x \times n} & \dots & \mathbf{C}_x \end{bmatrix}}_{\mathbf{R}_x} \vec{x} \leq \underbrace{\begin{bmatrix} c_x \\ \vdots \\ c_x \end{bmatrix}}_{\vec{c_x}}; \quad \underbrace{\begin{bmatrix} \mathbf{C}_u & \dots & \mathbf{0}_{r_u \times m} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{r_u \times m} & \dots & \mathbf{C}_u \end{bmatrix}}_{\mathbf{R}_u} \vec{u} \leq \underbrace{\begin{bmatrix} c_u \\ \vdots \\ c_u \end{bmatrix}}_{\vec{c_u}}, \tag{2.1.47}$$

y al reemplazar la ecuación (2.1.38) en estas expresiones, se obtiene:

$$\mathbf{R}_x \mathbf{F} \vec{u} \leq \vec{c_x} - \mathbf{R}_x \mathbf{E} x_0, \tag{2.1.48}$$

$$\mathbf{R}_u \vec{u} \leq \vec{c_u}. \tag{2.1.49}$$

Con estas desigualdades, se puede obtener la desigualdad para un problema QP en forma estándar con variables de decisión $\vec{u}$, donde la matriz $\mathbf{R} \in \mathbb{R}^{(r_u \cdot N + r_x \cdot N) \times (m \cdot N)}$ y el vector $\vec{c} \in \mathbb{R}^{(r_u \cdot N + r_x \cdot N)}$ se definen como:

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_x \mathbf{F} \\ \mathbf{R}_u \end{bmatrix}, \tag{2.1.50}$$

$$\vec{c} = \begin{bmatrix} \vec{c_x} - \mathbf{R}_x \mathbf{E} x_0 \\ \vec{c_u} \end{bmatrix}. \tag{2.1.51}$$

Con esto, las matrices y vectores utilizados en el problema QP estándar (2.1.36), quedan definidos por las expresiones (2.1.45), (2.1.46), (2.1.50) y (2.1.51), con lo que se cumple el objetivo de esta sección. En la siguiente sección se explica cómo utilizar el algoritmo numérico ADMM para resolver este problema QP.

2.2. ADMM para MPC implícito

Una vez que el problema QP definido en (2.1.34) queda expresado en forma de problema QP estándar (2.1.36), se pueden plantear algoritmos iterativos para aproximarse a la solución óptima. En esta sección se describe el algoritmo numérico *Alternating Direction Method of Multipliers* (ADMM) [2, 12], el cual dado sus características, se elige para resolver en FPGA el problema QP asociado a la formulación implícita de MPC.

Las formulaciones implícitas de MPC resuelven el problema de optimización para el estado actual del sistema de manera *online*, es decir, en tiempo real y continuamente mientras el sistema está en operación. Esto tiene como desventaja que el problema de optimización debe resolverse por cada cálculo de un solo valor de actuación, y por lo tanto requiere una cantidad significativa de procesamiento y memoria en comparación con las técnicas de control más tradicionales como PID. En este contexto, la elección del método numérico a utilizar se torna particularmente relevante.

Para resolver el problema QP en FPGA, en general se han utilizado tres tipos de métodos: *Interior Point* (IPM) [13], *Active Set* (ASM) [14] y métodos de primer orden. Los métodos de punto interior (IPM) resuelven de manera iterativa un conjunto de problemas lineales que aproximan el óptimo, ofreciendo alta precisión, pero con una complejidad computacional elevada que los hace poco atractivos para implementaciones en hardware. Los métodos de conjunto activo (ASM), por su parte, identifican y actualizan iterativamente el conjunto de restricciones activas, lo que puede ser eficiente en problemas de baja dimensión, aunque su rendimiento decrece en sistemas grandes debido a las operaciones de factorización requeridas. Finalmente, los métodos de primer orden, que sólo requieren gradientes y operaciones matriciales simples, ofrecen menor precisión asintótica pero una gran ventaja en términos de velocidad y paralelización en FPGA. Dentro de esta última categoría se ha propuesto el uso de Fast Gradient Method (FGM) [15], así como también combinaciones de FGM con un Lagrangiano Aumentado [16], *Alternating Method of Multipliers* (ADMM) [3], entre otros.

En particular, ADMM es un método de primer orden que ha recibido atención recientemente, debido a que en la práctica se logra la convergencia en relativamente pocas iteraciones, las operaciones realizadas en cada iteración son paralelizables, y el manejo de restricciones es más flexible comparado con otros métodos de primer orden [3].

2.2.1. Formulación de ADMM general

Se comienza por plantear ADMM para un problema de optimización convexo con restricciones lineales de igualdad tiene la siguiente forma:

$$\begin{aligned} &\min_{x} \ f(x) \\ &\text{sujeto a} \quad \mathbf{A}x = b, \end{aligned} \tag{2.2.1}$$

donde $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ y $f(x)$ es una función convexa. El problema QP estándar (2.1.36) es un caso particular del problema de optimización planteado en (2.2.1), y en la siguiente sección se explica

cómo adaptar la formulación de ADMM de esta sección para aplicarlo específicamente a un problema QP.

ADMM en particular, se enfoca en optimizar dos variables de decisión introduciendo la variable $z \in \mathbb{R}^n$, dividiendo además la función de costo en dos partes convexas. El nuevo problema a optimizar es:

$$\begin{aligned} \min_{x,z} \ & f(x) + g(z) \\ \text{sujeto a} \quad & \mathbf{A}x + \mathbf{B}z = c, \end{aligned} \tag{2.2.2}$$

donde $c \in \mathbb{R}^m$ y $f(x)$ y $g(z)$ son funciones convexas.

En base a este problema se propone un Lagrangiano Aumentado:

$$L_\rho(x, z, \lambda) = f(x) + g(z) + \lambda^{\mathrm{T}}(\mathbf{A}x + \mathbf{B}z - c) + \frac{\rho}{2}||\mathbf{A}x + \mathbf{B}z - c||_2^2, \tag{2.2.3}$$

donde $\rho > 0$. Utilizando el cambio de variable $u = \lambda/\rho$, el Lagrangiano puede expresarse como:

$$L_\rho(x, z, u) = f(x) + g(z) + \frac{\rho}{2}||\mathbf{A}x + \mathbf{B}z - c + u||_2^2 + \frac{\rho}{2}||u||_2^2. \tag{2.2.4}$$

En base a este Lagrangiano, se plantean las ecuaciones para obtener los valores óptimos de las variables de decisión y la variable dual:

$$x^{(k+1)} = \arg\min_x \left(f(x) + \frac{\rho}{2} \left\| \mathbf{A}x + \mathbf{B}z^{(k)} - c + u^{(k)} \right\|_2^2 \right), \tag{2.2.5}$$

$$z^{(k+1)} = \arg\min_z \left(g(z) + \frac{\rho}{2} \left\| \mathbf{A}x^{(k+1)} + \mathbf{B}z - c + u^{(k)} \right\|_2^2 \right), \tag{2.2.6}$$

$$u^{(k+1)} = u^{(k)} + \mathbf{A}x^{(k+1)} + \mathbf{B}z^{(k+1)} - c. \tag{2.2.7}$$

El proceso de optimización se hace iterativamente alternando las búsquedas de cada variable por separado, comenzando con la variable $x^{(k+1)}$, para luego utilizar este valor óptimo para encontrar la variable $z^{(k+1)}$ óptima, y finalmente se utilizan estos valores para actualizar a la variable dual $u^{(k+1)}$. De esta manera, se realizan varias iteraciones con el fin de encontrar el mínimo global de la función $f(x) + g(z)$.

2.2.2. Formulación de ADMM para un problema QP

El objetivo ahora es formular ADMM para un problema QP en forma estándar como el planteado en (2.1.36), utilizando un *solver* cuyas características presentan ventajas para ser implementado en FPGA [12]. El problema QP es de n variables de decisión y m restricciones, y notar que las restricciones de ADMM son de igualdad pero las restricciones del problema QP estándar son de desigualdad. Para esto se introduce la variable de holgura $z \in \mathbb{R}^m$, y se formula ADMM de la manera mostrada en (2.2.2):

$$\begin{aligned} \min_{x,z} \ & \overbrace{\frac{1}{2}x^{\mathrm{T}}\mathbf{Q}x + \vec{q}^{\mathrm{T}}x}^{f(x)} + \overbrace{\sum_{i=1}^{m} I_{z\geq 0}(z_i)}^{g(z)} \\ \text{sujeto a} \quad & \mathbf{R}x + \mathbf{I}z = c, \end{aligned} \tag{2.2.8}$$

donde la matriz $\mathbf{I}$ representa a la identidad de dimensión $m \times m$, y la función indicatriz $I_{z\geq 0}(z_i)$ se define como:

$$I_{z\geq 0}(z_i) = \begin{cases} 0 & z_i \geq 0 \\ \infty & z_i < 0. \end{cases}$$

La función $g(z)$ en la función de costo asegura que para un problema factible, ninguna componente de z es negativa, de manera que las restricciones de igualdad de este problema se cumplan si solo si se cumplen las restricciones de desigualdad del problema QP original. Reemplazando las funciones y matrices del problema QP en las ecuaciones de ADMM se obtiene:

$$x^{(k+1)} = \arg\min_x \left(\frac{1}{2} x^{\mathrm{T}} \mathbf{Q} x + \vec{q}^{\mathrm{T}} x + \frac{\rho}{2} \left\| \mathbf{R}x + z^{(k)} - c + u^{(k)} \right\|_2^2 \right), \tag{2.2.9}$$

$$z^{(k+1)} = \arg\min_z \left(\sum_{i=1}^{m} I_{z\geq 0}(z_i) + \frac{\rho}{2} \left\| \mathbf{R}x^{(k+1)} + z - c + u^{(k)} \right\|_2^2 \right), \tag{2.2.10}$$

$$u^{(k+1)} = u^{(k)} + \mathbf{R}x^{(k+1)} + z^{(k+1)} - c. \tag{2.2.11}$$

Derivando las expresiones e igualando a cero para encontrar los valores óptimos, se obtienen las siguientes expresiones, y con eso se plantea el algoritmo en pseudocódigo:

$$x^{(k+1)} = (\mathbf{Q} + \rho \mathbf{R}^{\mathrm{T}} \mathbf{R})^{-1} (-q - \rho \mathbf{R}^{\mathrm{T}} (z^{(k)} - c + u^{(k)})), \tag{2.2.12}$$

$$z^{(k+1)} = \max \left\{ \mathbf{0}_{m\times 1},\ \ c - \mathbf{R}x^{(k+1)} - u^{(k)} \right\}, \tag{2.2.13}$$

$$u^{(k+1)} = u^{(k)} + \mathbf{R}x^{(k+1)} + z^{(k+1)} - c. \tag{2.2.14}$$

Notar que las variables se inicializan al valor final obtenido en el cálculo anterior para obtener lo que se denomina arranque en caliente (*warm-start*). Destacar además que deben elegirse de antemano los parámetros ρ e IT, que corresponden a la tasa de aprendizaje y al número de iteraciones respectivamente. La elección de estos parámetros tiene un efecto en qué tanto se acercará la solución al valor óptimo. El valor de ρ puede elegirse utilizando alguna heurística [12], y en cuanto al número de iteraciones debe hacerse un compromiso entre tiempo de ejecución y exactitud de la solución.

Algorithm 1 ADMM para MPC

Require: q, c ▷ Dependen del estado medido x_0
  $\mathbf{W}^{-1} \leftarrow (\mathbf{Q} + \rho \mathbf{R}^{\mathrm{T}} \mathbf{R})^{-1}$ ▷ Constante del sistema
  $x_0, z_0, u_0 \leftarrow x_{IT}, z_{IT}, u_{IT}$ ▷ Arranque en caliente
  for $k = 0$ to $k = IT - 1$ **do**
    $v_k = z_k - c + u_k$ ▷ Variable auxiliar
    $x_{k+1} = \mathbf{W}^{-1}(-q - \rho \mathbf{R}^{\mathrm{T}} v_k)$
    $z_{k+1} = \max \{ \mathbf{0}_{m\times 1},\ \ c - \mathbf{R}x_{k+1} - u_k \}$
    $u_{k+1} = u_k + \mathbf{R}x_{k+1} + z_{k+1} - c$
  end for
  return x_{IT}

2.3. MPC explícito con redes neuronales artificiales

En esta sección se comienza formulando MPC explícito como un problema multiparamétrico, y se explican los elementos básicos de redes neuronales artificiales y cómo estas pueden utilizarse para aproximar la solución a este problema y las consideraciones que deben tenerse en cuenta al implementarse.

2.3.1. Formulación explícita de MPC

En las implementaciones explícitas de MPC, el problema QP asociado se resuelve de manera *offline*, esto significa que las soluciones para los posibles estados del sistema y referencias se resuelven de antemano y se almacenan en memoria, por lo que en tiempo de ejecución debe efectuarse un proceso de búsqueda de la solución. Este tipo de formulaciones pueden plantearse como un problema de optimización multiparamétrico, cuya solución define la actuación óptima $\mathbf{u}$ a partir del estado factible $\mathbf{x_r}$ que incluye las referencias como estados adicionales. De esta manera se separa el espacio de estados factibles $\mathcal{X}_r$ en regiones:

$$\mathbf{u}(x_r) = \begin{cases} \mathbf{F}_1 x_r + g_1 & \text{si } x_r \in \mathcal{R}_1, \\ \vdots & \vdots \\ \mathbf{F}_\eta x_r + g_\eta & \text{si } x_r \in \mathcal{R}_\eta, \end{cases} \tag{2.3.1}$$

donde $\mathbf{F_i} \in \mathbb{R}^{m\times(n+p)}$ y $g_i \in \mathbb{R}^m$ para cada región $\mathcal{R}_i$, las cuales se definen como:

$$\mathcal{R}_i = \{x_r \in \mathbb{R}^{n+p} \mid \mathbf{H}_i x_r \leq k_i\} \quad \forall i = 1, \ldots, \eta, \tag{2.3.2}$$

donde $\mathbf{H}_i \in \mathbb{R}^{c_i\times(n+p)}$ y $k_i \in \mathbb{R}^{c_i}$ describen los límites de la región $\mathcal{R}_i$ (notar que las referencias $r \in \mathbb{R}^p$ se incluyen como estados adicionales). En esta formulación, η es la cantidad de regiones y el espacio de estados factibles queda dividido de tal manera que $\mathcal{X}_r = \cup_{i=1}^{\eta} \mathcal{R}_i$. Además, para asegurar que exista una única acción de control óptima para cada estado, las regiones se eligen de manera que $\text{int}(\mathcal{R}_i) \cap \text{int}(\mathcal{R}_j) = \emptyset$, $\forall i \neq j$ y que $x_0 \in \partial R_i \cap \partial R_j \implies \mathbf{F}_i x_0 + g_i = \mathbf{F}_j x_0 + g_j$.

La literatura muestra distintos enfoques para reducir los recursos computacionales requeridos para MPC explicito, como por ejemplo eliminar regiones redundantes [4], o utilizar diversas estructuras de datos como por ejemplo árboles de búsqueda binarios [17] para reducir el tiempo de búsqueda de la acción óptima. Recientemente, se ha planteado el uso de redes neuronales profundas, logrando reducir el uso de memoria y la latencia asociada a la elección de la acción [5, 6].

2.3.2. Redes neuronales artificiales

Una red neuronal artificial (ANN, del inglés *Artificial Neural Network*) es un modelo de aprendizaje de máquinas que consta de múltiples capas, conformadas por varias unidades básicas de cómputo llamadas neuronas artificiales. En este trabajo solo se considerarán redes neuronales de tipo *feedforward* y completamente conectadas, lo que quiere decir que cada capa está únicamente conectada con la capa anterior y la siguiente, y que además cuando se conectan dos capas existen conexiones para todos los pares de neuronas entre una capa y la otra. Decimos que la red es una red neuronal profunda (DNN, del inglés *Deep Neural Network*)) si $L \geq 2$.

La Figura 2.3 muestra una DNN de tipo *feedforward* y completamente conectada, con 2 capas ocultas y 4 neuronas cada una. Las salidas de cada capa son la entrada de la capa siguiente, y en

el contexto de MPC las componentes del estado x_r (con las referencias como estados adicionales) son las entradas de la primera capa, y las salidas de la última capa son la acción de control óptima estimada $\hat{u}$. De esta manera, la red consiste de una capa de entrada en la cual cada componente del estado $x_r \in \mathbb{R}^{n+p}$ pasa directamente hacia la primera capa oculta, una capa de salida donde la salida de cada neurona es una componente de la acción óptima estimada $\hat{u} \in \mathbb{R}^m$, y L capas ocultas entre las capas de entrada y salida, que consideraremos de M neuronas cada una.

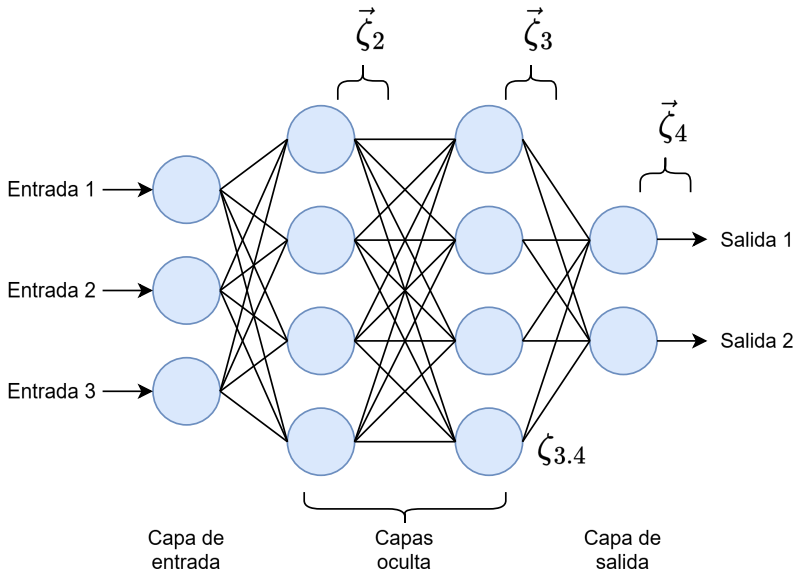


Fig. 2.3: DNN de tres entradas, dos salidas, dos capas ocultas y cuatro neuronas por capa.

La red contiene parámetros internos ajustables θ para las capas ocultas y la de salida, que consisten de un peso $W_{\ell,i,j}$ para cada par de neuronas conectadas y un *bias* $b_{\ell,i}$ para cada neurona. En esta notación ℓ representa a la capa, i a la i-ésima neurona de la capa ℓ, y j a la j-ésima neurona de la capa anterior. Los parámetros θ son ajustados durante la etapa de entrenamiento de la red, y una vez ya entrenada esta se puede usar para realizar inferencias, es decir, obtener resultados en su salida. La calidad de estos resultados son evaluados utilizando un conjunto de datos distintos al de entrenamiento.

Cada neurona (que no sea de entrada) tiene un valor numérico en la salida que viene dada por la aplicación de una función de activación g_ℓ a una función afín de las salidas de las neuronas de la capa anterior usando los pesos como coeficientes. De esta manera, la salida de la neurona i de la capa $\ell > 2$ que determinada por:

$$\zeta_{\ell,i} = g_\ell \left(b_{\ell,i} + \sum_{j=1}^{M} W_{\ell,i,j} \zeta_{\ell-1,j} \right) \tag{2.3.3}$$

Para el caso $\ell = 2$ la diferencia es que la suma debe hacerse sobre el número de entradas de la red, lo que en el caso de aplicación a MPC corresponde a $n + p$. La Figura 2.4 muestra las operaciones realizadas en una neurona.

Utilizando notación vectorial, se puede expresar el vector de todas las salidas de las neuronas de

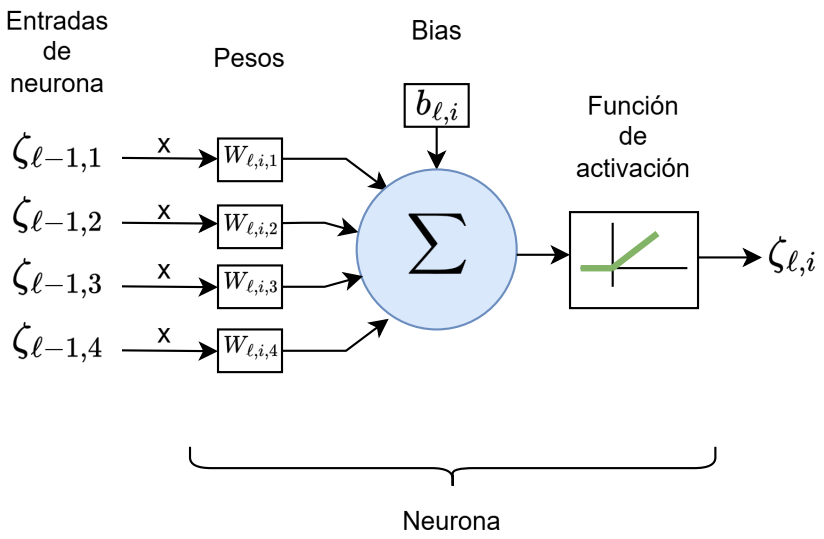


Fig. 2.4: Neurona i-ésima de la capa $\ell > 2$ de una red con $M = 4$ y función de activación ReLU.

la capa ℓ como la aplicación de la función de activación g_ℓ a la función afín f_ℓ aplicada al vector de salidas de la capa anterior.

$$\vec{\zeta}_\ell = g_\ell \circ f_\ell \left(\vec{\zeta}_{\ell-1}\right) \tag{2.3.4}$$

$$= g_\ell \left(\vec{b}_\ell + W_\ell \vec{\zeta}_{\ell-1}\right) \tag{2.3.5}$$

Considerando que en el contexto de aplicación a MPC la entrada de la red es el vector de estados x_r (con las referencias como estados adicionales) y la salida es la acción de control óptima estimada $\hat{u}$, la red neuronal corresponde a una función $\mathcal{N} : \mathbb{R}^{n+p} \mapsto \mathbb{R}^m$ que está dada por la composición sucesiva (para cada capa) de las funciones expresadas en (2.3.4), la cual se describe como:

$$\hat{u} = \mathcal{N}(x_r, \theta, M, L) = \begin{cases} g_{L+1} \circ f_{L+1} \circ g_L \circ f_L \circ \cdots \circ g_1 \circ f_1(x) & \text{si } L \geq 2, \\ g_{L+1} \circ f_{L+1} \circ g_1 \circ f_1(x) & \text{si } L = 1. \end{cases} \tag{2.3.6}$$

La función de activación g_ℓ es una función no lineal, y en efecto es el único elemento no lineal de la red, lo que le otorga la capacidad de aproximar funciones no lineales. Cómo se explica en la próxima sección, para la aplicación a MPC la función de activación ReLU (del inglés, Rectifier Linear Unit) tiene la ventaja de proporcionar una cota inferior en el número máximo de regiones afines que puede representar la red. La ecuación (2.3.7) describe a esta función en su forma vectorial, la cual toma los máximos entre cada componente del vector y el valor cero. En la Figura 2.4 se muestra esta función de activación aplicada a un solo valor numérico en la salida de una neurona.

$$g_\ell(\vec{v}) = \text{ReLU}(\vec{v}) = \begin{bmatrix} \text{máx}(0, v_1) \\ \text{máx}(0, v_2) \\ \vdots \\ \text{máx}(0, v_M) \end{bmatrix} \tag{2.3.7}$$

Una ANN como la planteada en la ecuación (2.3.6) en consiste en la composición alternada de funciones afines y no lineales, y si se utiliza exclusivamente la función ReLU (2.3.7) como función de activación, la red representa una función definida por tramos afines. Esto es especialmente adecuado para aproximar la ley de control multiparamétrica expresada en (2.3.1). Además, si se elige $M \geq n+p$, se tiene una cota inferior para el número máximo de regiones afines que la red puede representar [18]:

$$\left(\prod_{\ell=1}^{L-1} \left\lfloor \frac{M}{n+p} \right\rfloor^{n+p} \right) \sum_{j=0}^{n+p} \binom{L}{j} \tag{2.3.8}$$

Notar que esta expresión implica que el número de regiones representables por la red crece considerablemente con el número de capas L, siempre y cuando $M \geq n + p$. Si además se cumple que $M \geq 3(n + p)$, es posible obtener una cota mejor [19].

2.3.3. Aproximación de la ley de control con redes neuronales

Conjunto de datos

Para aproximar la ley de control con redes neuronales, es necesario un conjunto de datos representativo del espacio de estados del sistema y sus correspondientes actuaciones óptimas. Estos datos pueden obtenerse de formulaciones tanto implícitas como explícitas, y su obtención es previa al diseño de la red que será utilizada.

Este conjunto de datos puede generarse utilizando diversas estrategias [20]. A grandes rasgos, los datos pueden generarse utilizando simulaciones de lazo abierto y lazo cerrado. Las simulaciones en lazo abierto realizan un muestreo aleatorio de los estados y referencias (típicamente utilizando una distribución uniforme dentro del espacio de estados factibles), para luego calcular la actuación óptima para cada muestra. Este método tiene como desventaja que se considera una gran cantidad de combinación de estados que son altamente improbables de obtener en la operación normal de la planta. Debido a esto, a veces se utilizan simulaciones de lazo cerrado, donde se eligen varios estados iniciales del sistema, y para cada estado inicial se realiza una simulación hasta alcanzar el estado estacionario, capturando la secuencia completa de estados y sus respectivas actuaciones calculadas. Este método tiene como desventaja que no suele cubrir tan representativamente el espacio de estados factibles como con simulaciones de lazo abierto.

Estas estrategias no son excluyentes y suelen usarse en combinación. Además, para la simulación en lazo abierto se recomienda generar datos relajando las restricciones para incluir estados fuera del rango normal de operación (sin aumentar demasiado la cantidad de estados generados), para que en caso de que las restricciones no se cumplan por un leve margen el controlador tenga información de cómo actuar.

Cabe destacar que estos datos son normalizados previamente para ser utilizados por la red neuronal, por lo que en las secciones siguientes se consideran estados y actuaciones normalizados. Las técnicas de normalización más utilizadas son *Min-Max Scaling* y normalización *Z-Score*. El método de *Min-Max Scaling* utiliza el mínimo y el máximo de los datos para transformarlos a

una escala entre 0 y 1, lo que tiene como ventaja que los límites de la escala son consistentes. La normalización *Z-Score* utiliza la media y la desviación estándar de los datos para transformarlos a una escala de media cero y desviación estándar unitaria, reduciendo la sensibilidad de la escala ante *outliers*. Dado que en general los estados y actuaciones en un problema de control tienen como límites valores preestablecidos, es preferible utilizar *Min-Max Scaling*. Para un dato X con un límite máximo de X_{max} y límite mínimo X_{min}, el valor escalado con *Min-Max Scaling* puede obtenerse como:

$$X_{scale} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{2.3.9}$$

Ajuste de parámetros

Para el ajuste de los parámetros, debe elegirse una función de pérdida $\mathcal{L}$ que cuantifica el error de aproximación de N_s vectores de salidas de la red. Una elección típica para problemas de regresión es el error cuadrático medio (MSE, Mean Squared Error) entre los valores entregados en la salida de la red $\hat{u}$ y los valores correctos según los datos disponibles u:

$$\mathcal{L}(\mathcal{N}(x_r, \theta), u) = \frac{1}{N_s} \sum_{i=1}^{N_s} (u_i - \hat{u}_i)^2. \tag{2.3.10}$$

Utilizando esta función, cada parámetro es ajustado iterativamente según cómo afecta cada parámetro en la pérdida. Formalmente, se calculan las derivadas parciales $\dfrac{\partial \mathcal{L}}{\partial \theta_i}$, usualmente mediante la técnica de *backpropagation* [21], para luego aplicar algún algoritmo de optimización para actualizar los parámetros. Este algoritmo suele ser alguna variante de descenso de gradiente [22].

Un aspecto importante a considerar en la fase de entrenamiento, es que la red logre generalizar correctamente las relaciones presentes en los datos. Esto quiere decir que esta sea capaz de entregar valores correctos para datos distintos a los utilizados para entrenar. Para cuantificar la generalización, se utiliza un conjunto de *test*, que evalúa el desempeño de la red con datos distintos a los utilizados para entrenar.

Para lograr una correcta generalización, debe elegirse una capacidad adecuada para la red, donde la capacidad corresponde al tamaño del espacio de funciones representables por la red. Esta capacidad es controlable mediante la elección de los hiperparámetros de la red, es decir, a variables elegidas durante el diseño de la red, como el número de capas L, el número de neuronas por capa M, la función de activación utilizada, entre otros. Una capacidad muy baja impide que la red logre representar a todas las funciones deseadas, y una capacidad muy alta puede ajustar la red excesivamente a los datos de entrenamiento, aumentando el error del conjunto de *test*, aún cuando el error de entrenamiento sea bajo. Este fenómeno denominado *overfitting*, puede evitarse mediante técnicas de regularización [23], que puede consistir por ejemplo considerar un término de penalización directa a la magnitud de los parámetros θ. El uso de regularización L2 modificaría la función de pérdida MSE (2.3.10), obteniendo

$$\mathcal{L}(\mathcal{N}(x_r, \theta), u, \theta) = \frac{1}{N_s} \sum_{i=1}^{N_s} (u_i - \hat{u}_i)^2 + \lambda ||\theta||_2^2, \tag{2.3.11}$$

donde λ es un factor de regularización que también se considera un hiperparámetro. Para utilizar regularización L1 se cambia el cuadrado de la norma L2 por la norma L1, que corresponde la suma de las magnitudes de los parámetros. Además, cada iteración en la que los parámetros de la red

se ajustan con el fin de reducir la función de pérdida se denomina época (o *epoch*), y la elección del número de épocas tiene un efecto considerable en la calidad del aprendizaje, ya que elegir un número muy bajo puede impedir que la red aprenda lo suficiente, pero si se itera excesivamente puede producirse *overfitting*. El número de épocas se considera un hiperparámetro y es importante considerar una estrategia adecuada para su elección.

2.3.4. Cuantización de redes neuronales

Una vez entrenada la red, esta debe implementarse en algún dispositivo para ser utilizada con fines de control, como por ejemplo en una CPU. En este trabajo, tanto para MPC implícito como explícito, se utilizará aceleración por hardware, específicamente con FPGA. En este contexto, la representación numérica de las variables se torna más relevante y se deben tomar decisiones que involucran un *trade-off* entre precisión y uso de recursos tanto temporales como espaciales.

Específicamente, debe elegirse entre una representación en punto flotante y en punto fijo para números decimales. La primera tiene la ventaja de tener una precisión elevada, al costo de que las operaciones son más costosas computacionalmente, y que además tiene formatos específicos de 32 o 64 bits. Las representaciones en punto fijo pueden tener una longitud arbitraria según la precisión deseada, y las operaciones en hardware son considerablemente más simples.

La cuantización de redes neuronales consiste en utilizar representaciones numéricas de punto fijo para los parámetros de la red. Esto permite utilizar menos memoria y hacer más eficientes los cálculos de la red, disminuyendo la latencia de la red. Esto la dificultad adicional de elegir la precisión adecuada considerando los rangos dinámicos de las variables según el problema.

Para problemas de clasificación, existen diversos estudios que analizan los efectos de la cuantización [24]; pero en cuanto a problemas de regresión, como lo es la aproximación de la ley de control, resulta necesario hacer una evaluación del efecto de las representaciones cuantizadas en el desempeño del lazo.

Para mitigar el impacto de la cuantización en la inferencia de la red, puede efectuarse un entrenamiento cuantizado. Esto consiste en incluir la cuantización en la fase de entrenamiento, lo que permite que el ajuste de los parámetros se efectúe considerando las limitaciones de la cuantización. Para diseñar y entrenar redes neuronales con precisión reducida, se pueden utilizar herramientas como QKeras [25], que entregas control sobre el número de bits utilizados en cada capa de la red. Una vez diseñada la red, la FPGA ofrece control fino sobre la configuración del hardware de inferencia, permitiendo ajustar la precisión necesaria para cada parámetro y cada operación. El uso de representaciones cuantizadas para aproximar la ley de control no ha sido explorado en la literatura reciente.

2.4. Herramientas de implementación

2.4.1. Field Programmable Gate Array (FPGA)

Las FPGAs son dispositivos electrónicos reconfigurables diseñados para implementar circuitos digitales de manera flexible y eficiente. A diferencia de los circuitos integrados de aplicación específica (ASICs), cuya funcionalidad queda determinada desde su fabricación, las FPGAs pueden ser reprogramadas según las necesidades del usuario, permitiendo modificaciones y mejoras sin necesidad de fabricar nuevos chips. Esta capacidad las hace especialmente útiles en entornos donde la adaptabilidad y la optimización del hardware son esenciales, como en telecomunicaciones, procesamiento de

señales, criptografía y sistemas embebidos [26].

Internamente, una FPGA está compuesta por una matriz de bloques lógicos interconectados, conocidos como CLBs (*Configurable Logic Blocks*), que contienen tablas de búsqueda (LUTs), *Flip-Flops* (FFs) y multiplexores. Estos bloques pueden configurarse para realizar cualquier operación lógica deseada, mientras que las interconexiones programables permiten establecer las conexiones adecuadas entre ellos. Además, las FPGAs incluyen bloques de memoria embebidos (como BRAMs), unidades de procesamiento de señales digitales (DSPs) para cálculos matemáticos intensivos, y una serie de puertos de entrada y salida que facilitan la comunicación con otros dispositivos. La Figura 2.5 muestra la arquitectura interna de una FPGA típica.

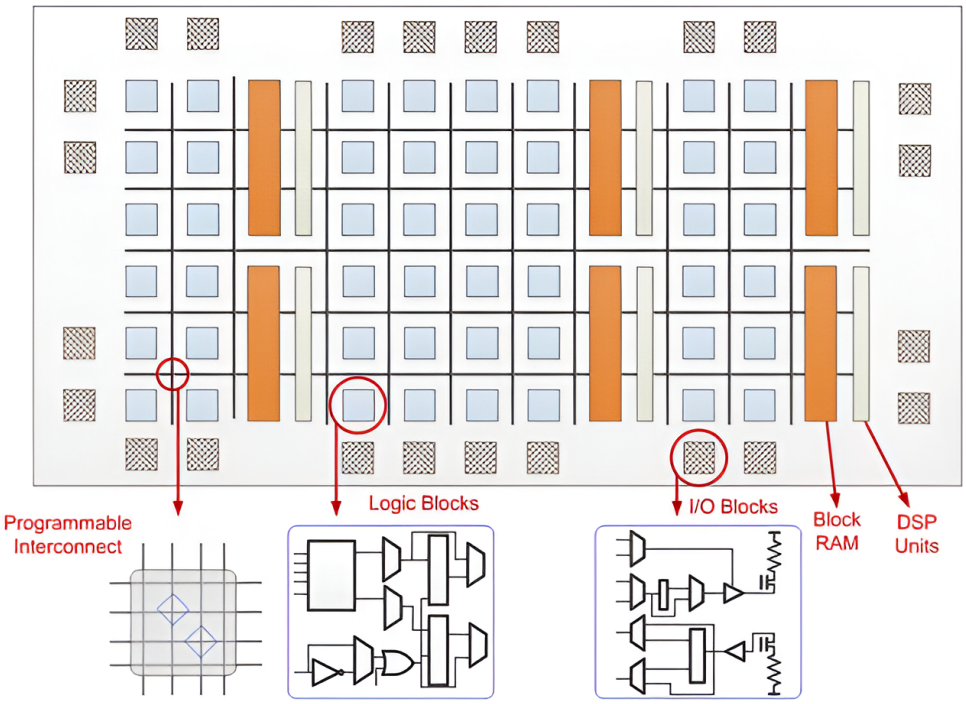


Fig. 2.5: Estructura interna de una FPGA típica [27].

Una de las principales ventajas de las FPGAs es su flexibilidad. A diferencia de una CPU o GPU, que ejecutan instrucciones en un flujo secuencial o masivamente paralelo según su arquitectura, las FPGAs permiten definir una arquitectura de hardware específica para cada aplicación. Esto significa que, en lugar de depender de un conjunto predefinido de instrucciones, el diseñador puede crear circuitos optimizados para resolver un problema de manera directa y eficiente. Gracias a esto, una FPGA puede ofrecer un rendimiento superior en tareas altamente especializadas, más aún cuando la latencia y la velocidad de cálculo son críticas. Además, si bien un ASIC puede superar a una FPGA en eficiencia y consumo energético una vez fabricado, su desarrollo es costoso y requiere largos tiempos de producción, lo que hace que una FPGA sea una alternativa viable en aplicaciones que requieren tanto flexibilidad como alto rendimiento.

El diseño de circuitos en FPGA se realiza mediante herramientas especializadas como Vivado (Xilinx) o Quartus (Intel), utilizando lenguajes de descripción de hardware como VHDL, Verilog o SystemVerilog; como también lenguajes de más alto nivel como C/C++ para generar automáticamente la descripción del hardware, con directivas específicas que permiten ajustar la manera en que

se genera la descripción. Esta última técnica se conoce como *High Level synthesis* (HLS) [28], y será ampliamente utilizada en este trabajo.

Para la implementación de redes neuronales en FPGA, existen herramientas dedicadas como HLS4ML [29], que es una biblioteca de Python diseñada para facilitar la generación de descripciones de hardware optimizadas para FPGAs a partir de modelos de ANNs desarrollados en frameworks como Keras o PyTorch. Específicamente, HLS4ML toma como entrada un modelo de red ya entrenado junto con un archivo de configuración que define parámetros de implementación para la arquitectura de hardware, como el nivel de paralelismo en las operaciones y la precisión numérica (cuantización) en cada capa de la red. A partir de esta información, la herramienta produce código en C/C++ con directivas de optimización debidamente adaptado para ser procesado por una herramienta de HLS.

Las aplicaciones de las FPGAs abarcan una amplia gama de áreas, desde sistemas de comunicación hasta criptografía y control de sistemas embebidos. En redes y telecomunicaciones, se utilizan para la implementación de protocolos de alta velocidad, como Ethernet y PCIe. En procesamiento de señales digitales, se emplean para mejorar el rendimiento en tareas como la compresión de imágenes y el filtrado de audio. También son fundamentales en criptografía, donde la capacidad de reconfiguración permite adaptar rápidamente algoritmos de cifrado a nuevas amenazas de seguridad. En computación de alto rendimiento, su uso ha crecido en aplicaciones de inteligencia artificial y aprendizaje profundo, donde actúan como aceleradores de hardware optimizados para tareas específicas.

En este trabajo, la FPGA será utilizada para acelerar el cálculo de la actuación óptima de control, permitiendo aplicar MPC en contextos con garantías de latencia y restricciones de recursos, aún cuando es una técnica de control computacionalmente demandante. Para mayor flexibilidad y control sobre la FPGA, se utilizará además una herramienta de control por prototipado rápido, la cual se detalla en la siguiente sección.

2.4.2. Control por prototipado rápido

El control por prototipado rápido (Rapid Control Prototyping, RCP) es una metodología que permite validar y experimentar sistemas de control de manera eficiente, evitando las largas etapas de implementación y depuración propias de los sistemas embebidos tradicionales. Su principal objetivo es facilitar la prueba directa de algoritmos sobre sistemas reales o simulados, utilizando hardware y herramientas que permiten una implementación semiautomática y una interacción flexible. De esta manera, es posible verificar la viabilidad y el rendimiento de las leyes de control antes de comprometer recursos en el diseño final del controlador o su implementación definitiva en hardware. Entre las aplicaciones más populares del control por prototipado rápido destacan la automatización de vehículos [30, 31], la industria aeroespacial [32], la robótica [33] y las energías renovables [34, 35].

En esencia, el prototipado rápido busca cerrar la brecha entre el diseño teórico y la aplicación práctica. Las plataformas de prototipado rápido suelen integrar periféricos, como por ejemplo conversores análogo-digital con interfaces configurables y circuitos de protección; también interfaces de usuario para visualización de datos, y soporte para diseño basado en abstracciones de alto nivel (por ejemplo, Matlab/Simulink), entre otras características que permiten flexibilizar la etapa de desarrollo y reducir los tiempos y riesgos asociados a la implementación. Este enfoque resulta especialmente ventajoso cuando se trabaja con controladores avanzados, tales como MPC, donde las fases de prueba y ajuste fino requieren múltiples iteraciones y validación experimental.

Por otro lado, dentro de las desventajas de este enfoque, se encuentra que estas plataformas suelen estar diseñadas para trabajar con librerías específicas propietarias que proveen abstracciones de alto nivel precompiladas para tareas típicas, limitando el control del usuario sobre la ejecución de los procesos subyacentes y dificultando la integración de bloques de procesamiento personalizados.

Estas restricciones implican que una implementación optimizada a bajo nivel, como por ejemplo los aceleradores reportados en la literatura que justamente se basan en el ajuste fino para el uso de recursos en forma eficiente, puedan no ser directamente portables a una plataforma de prototipado rápido manteniendo el desempeño. Además, estos productos suelen tener un elevado costo monetario, una curva de aprendizaje pronunciada y escasos trabajos académicos que indiquen los detalles de implementación, lo cual impide la replicación y validación de resultados.

Existen varias alternativas comerciales y académicas para llevar a cabo prototipado rápido. Entre ellas destacan plataformas basadas en microcontroladores, FPGAs genéricas y soluciones específicas como las de National Instruments (NI), OPAL-RT, Speedgoat o dSPACE. Cada una de estas alternativas ofrece un nivel distinto de integración con entornos de desarrollo y simulación, capacidad de procesamiento en tiempo real y facilidades de conexión con sistemas físicos. En particular, las plataformas de dSPACE son reconocidas y robustas en el ámbito académico e industrial, y es la alternativa utilizada en esta tesis.

MicroLabBox de dSPACE

Dentro de la gama de productos de dSPACE, la MicroLabBox [9] se posiciona como una herramienta versátil y accesible para realizar control por prototipado rápido. Esta plataforma provee un sistema central de cómputo heterogéneo que integra CPUs con FPGA, incluyendo un procesador dedicado al cálculo en tiempo real, un amplio conjunto de periféricos analógicos y digitales, entre otras cosas; lo que la convierte en una solución compacta pero potente para la implementación de lazos de control exigentes. La MicroLabBox está diseñada para integrarse directamente con MATLAB/Simulink, permitiendo la generación automática de código y la descarga del mismo sobre el hardware mediante el entorno Real-Time Interface [36] de dSPACE. Esta integración no solo facilita la implementación inicial de controladores, sino que también permite realizar modificaciones rápidas, pruebas iterativas y depuración eficiente sin necesidad de modificar directamente el código embebido.

La Figura 2.6 muestra el exterior de la MicroLabBox y la Figura 2.7 muestra un diagrama de su estructura interna. Esta estructura consiste de los módulos DS1202 y DS1302, que contienen a las CPUs y la FPGA respectivamente. El módulo DS1202 contiene un procesador NXP QorlQ P5020, *dual-core*, 2 GHz para realizar procesamientos en tiempo real, y otro procesador NXP QorlQ P1011, 800 MHz para la comunicación con el host (PC) vía Ethernet. El procesador para tiempo real se conecta mediante buses de 32 y 64 bits al módulo DS1302, que contiene una FPGA Xilinx® Kintex®-7 XC7K325TFBG900-1 y periféricos varios.

Fig. 2.6: MicroLabBox de dSPACE [9].

En el contexto de esta tesis, la MicroLabBox de dSPACE se utilizará como plataforma de experimentación para la implementación de lazos de control basados en técnicas de Control Predictivo (MPC), donde los algoritmos serán programados directamente sobre la FPGA de la plataforma. Esta capacidad resulta clave, ya que los algoritmos MPC suelen ser intensivos computacionalmente y requieren garantizar tiempos de ejecución consistentes en cada ciclo de control. Gracias a la arquitectura de la MicroLabBox, será posible evaluar diferentes variantes de MPC, modificando parámetros y observando su impacto en el desempeño del sistema controlado en tiempo real, además de realizar una caracterización detallada de la latencia del lazo de control, considerando todas las etapas necesarias desde la adquisición de datos hasta la generación de la actuación correspondiente.

La integración entre Matlab/Simulink y la MicroLabBox permite además realizar distintos tipos de simulaciones, que paulatinamente van incluyendo más elementos de la implementación física final. Esto incluye simulaciones en tiempo real, es decir, implementaciones donde la planta es simulada pero los cálculos se realizan instante a instante y los cambios de las variables pueden ser visualizados tal como sucedería en una implementación sin simulaciones. Esta metodología se conoce como flujo de diseño basado en modelos (o flujo HIL, de *Hardware-in-the-Loop*), y es sobre la cual se elabora el flujo de diseño propuesto y utilizado en esta tesis.

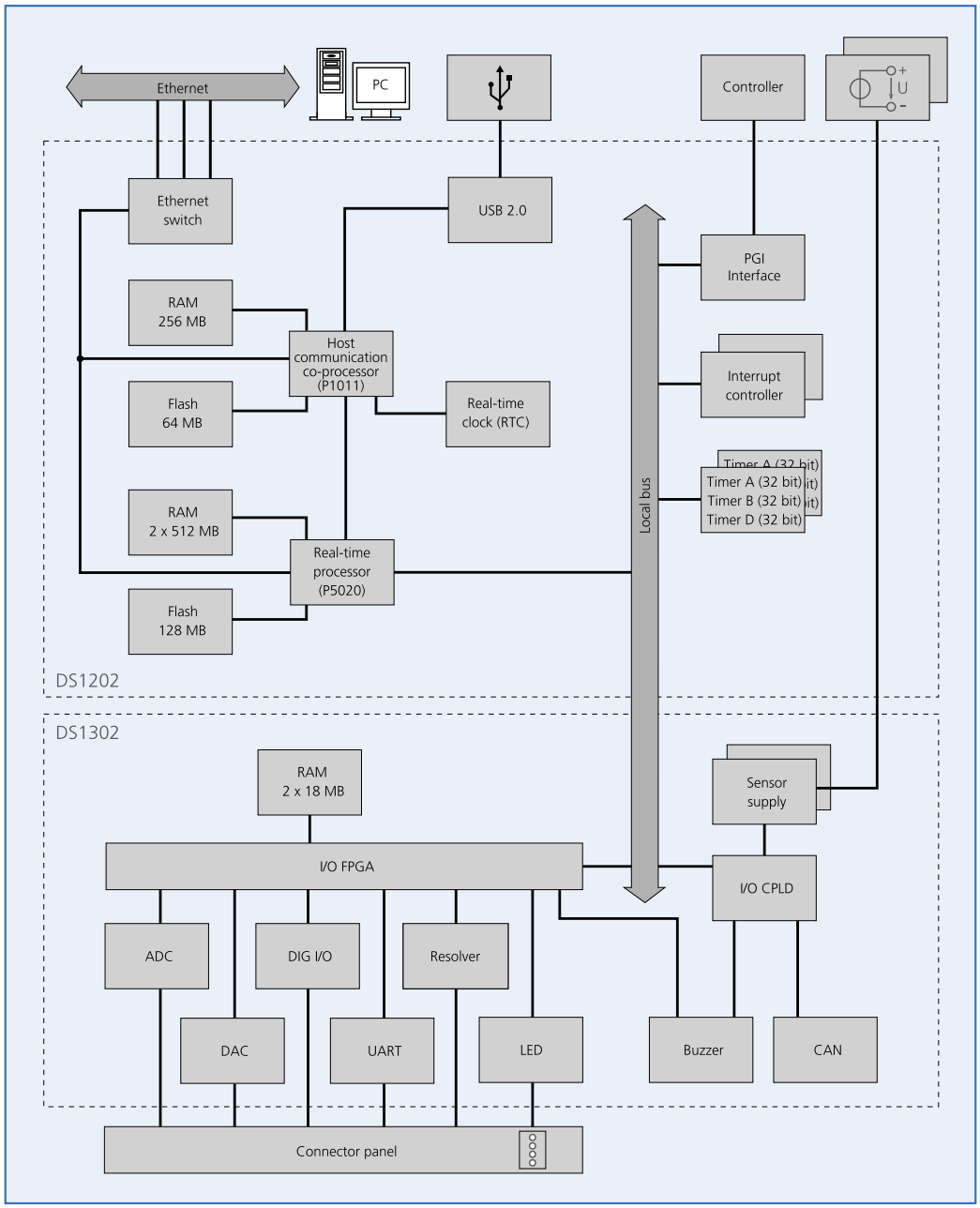


Fig. 2.7: Diagrama de MicroLabBox [9].

Capítulo 3

FLUJO DE DISEÑO BASADO EN MODELOS PARA MPC EN FPGA

En este capítulo se explica el flujo propuesto para el diseño de lazos MPC con aceleración en FPGA. Se proponen flujos separados para MPC implícito y para MPC explícito con redes neuronales. Se comienza explicando generalmente en qué consiste el flujo de diseño basado en modelos, sobre el cual se elaboran los flujos de diseño propuestos y utilizados en esta tesis. Luego se detalla cada etapa del proceso para diseñar un controlador MPC siguiendo estos flujos, incluyendo una explicación general de las herramientas y archivos necesarios en cada etapa del proceso, los cuales están disponibles en el repositorio [37] que complementa este documento.

3.1. Flujo de diseño basado en modelos

En esta sección se explica el flujo de diseño basado en modelos (o flujo HIL, de *Hardware-in-the-Loop*), el cual se utilizará para diseñar los flujos completos de diseño propuestos en la próxima sección. El flujo de diseño basado en modelos consiste en utilizar simulaciones a distinto nivel de abstracción para paulatinamente lograr la implementación completa en la planta física. A continuación se detalla cada etapa del flujo y las definiciones que serán consideradas en esta tesis, que están en línea general con las definiciones entregadas tanto por dSPACE [38] como Mathworks [39]. La figura 3.1 muestra resumidamente el flujo, con ejemplos de herramientas y lenguajes que podrían utilizarse en cada etapa.

1. Model-in-the-Loop (MIL): En esta primera etapa todos los elementos del lazo de control (controlador, planta y comunicación) son simulados en un lenguaje de programación a elección. El objetivo es validar a un nivel preliminar que el algoritmo utilizado logra controlar las dinámicas del sistema según se desea, probando diversas entradas y registrando las salidas obtenidas con el fin de hacer comparaciones con etapas posteriores. Además, al utilizar MIL los costos de probar distintas configuraciones del controlador son mínimos, por lo que en esta etapa los parámetros y ajustes que se planean utilizar en la implementación del lazo deberían definirse mediante experimentos. Esta etapa puede llevarse a cabo en cualquier lenguaje de programación o herramienta de cálculo adecuada, como lo son Python o Matlab respectivamente.
2. Software-in-the-Loop (SIL): Una vez terminada la etapa de MIL, el controlador ya debería estar matemáticamente diseñado y sus parámetros elegidos y probados. Con esto, ya es posible escribir el código que se utilizará para programar el hardware de control, que en general puede ser un procesador o una FPGA. Esta etapa también es completamente simulada y todavía no

se utiliza el hardware, sino que solamente se reemplaza el controlador diseñado en MIL por el mismo controlador pero utilizando el código que irá en la implementación final. De esta manera, se pueden detectar tempranamente varios errores que podrían ocurrir en la programación del dispositivo, antes de que este sea programado. Si el controlador es implementable y las señales registradas son consistentes con lo observado en MIL, es posible continuar a la siguiente etapa. Un ejemplo de lenguaje que se utiliza para procesadores es C/C++, y para FPGA algún HDL como Verilog, que podría ser generado mediante HLS. MATLAB Coder [40] y Vitis Model Composer [41] son ejemplos de herramientas que facilitan realizar simulaciones a nivel de SIL para C/C++ y Verilog respectivamente.

3. Processor/FPGA-in-the-Loop (PIL/FIL):

   En esta etapa se introduce el dispositivo de control (procesador para PIL, FPGA para FIL), el cual se programa con el mismo código utilizado en la etapa de SIL. En general podría usarse un emulador para el hardware objetivo específico o bien utilizar el dispositivo físico mismo, por lo cual ahora debe existir algún protocolo de comunicación entre el controlador (emulado o físico) y la planta simulada (típicamente SPI, I2C o UART). En esta etapa se verifica que el hardware de control es capaz de ejecutar el programa sin errores, y de no ser así deben considerarse alternativas de hardware o volver a alguna de las etapas anteriores. MATLAB Coder [40] y Vitis Model Composer [41] son ejemplos de herramientas que facilitan realizar simulaciones PIL y FIL respectivamente.

4. Hardware-in-the-Loop (HIL): En esta etapa ya se utiliza la mayoría del hardware correspondiente a la implementación final, a excepción de la planta. Esto también considera probar las entradas y salidas analógicas a nivel físico, verificando que las señales observadas corresponden a lo esperado. Un aspecto crucial de esta etapa, es que la simulación de la planta es en tiempo real, es decir, la interacción temporal con el hardware ocurre de la misma manera que ocurriría en la implementación física del lazo, permitiendo al usuario interactuar con los parámetros y visualizar las señales de interés en tiempo de ejecución. En esta etapa se verifica que todo el hardware permite ejecutar el lazo de control en tiempo real con un desempeño adecuado, y de no ser así se debe volver a alguna de las etapas anteriores. Una plataforma de prototipado rápido como las ofrecidas por dSPACE puede proveer la integración del hardware y software necesarios para esta etapa. En las implementaciones de esta tesis denominaremos HIL a un híbrido entre FIL y HIL (ver Sección 3.2.3).

5. Control: Finalmente se conecta el controlador a la planta física, logrando la implementación completa del lazo de control. Cualquier ajuste o modificación que se requiera debería volver a pasar por los pasos anteriores.

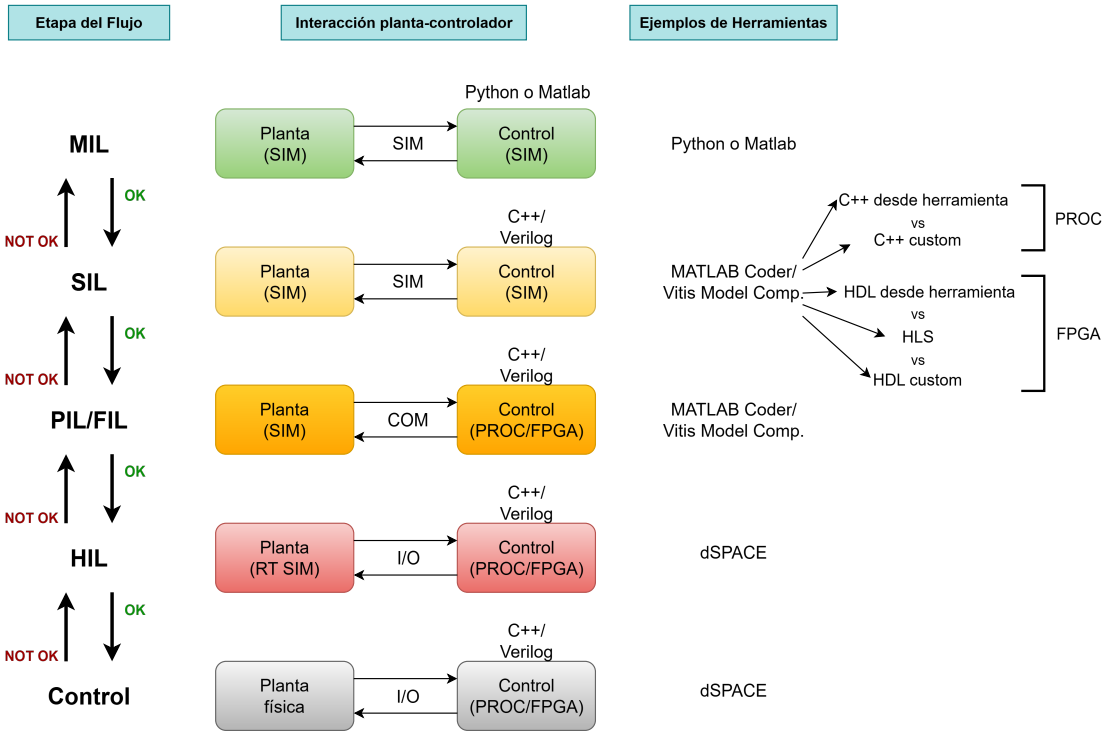


Fig. 3.1: Flujo general de diseño basado en modelos con herramientas y lenguajes de ejemplo para cada etapa.

3.2. Flujos de diseño propuestos para MPC implícito y explícito

En esta sección se detallan los flujos de diseño propuestos para MPC acelerado con FPGA, tanto para implícito y como para explícito con redes neuronales. Ambos flujos se construyen sobre el diseño basado en modelos, comenzando con simulaciones MIL, para luego generar la descripción del hardware de control y otros módulos útiles (archivos Verilog). Esta descripción obtenida es verificada funcionalmente mediante simulaciones SIL, incluyendo cosimulaciones para el caso de las redes neuronales. Una vez verificada la funcionalidad de las descripciones de hardware se procede a hacer simulaciones HIL (en tiempo real) utilizando la MicroLabBox de dSPACE, para finalmente lograr la implementación completa del lazo utilizando la planta física. Es posible previamente también realizar simulaciones FIL para verificar que el controlador es implementable en la tarjeta objetivo, pero dado que esto queda verificado en las simulaciones HIL eficazmente utilizando control por prototipado rápido, este paso será omitido (no obstante se incluye en el flujo general por completitud).

Esta sección se divide en tres partes, conteniendo las primeras dos el flujo de implícito y explícito respectivamente hasta la verificación funcional de las descripciones de hardware. La tercera parte es común para ambos flujos y explica de manera general cómo se utiliza la MicroLabBox para las simulaciones a nivel de FIL y HIL, y la implementación final del lazo.

3.2.1. Flujo de diseño para MPC implícito

La Figura 3.2 muestra el flujo completo para MPC implícito, y cada etapa es explicada en detalle a continuación.

- **MIL/Definición de matrices y parámetros**

  En esta etapa se realizan experimentos varios en Matlab para definir los parámetros del controlador y verificar el control en lazo cerrado a nivel de simulación. Los valores obtenidos en las simulaciones son utilizados para comparar con etapas posteriores.

  En particular para la implementación con ADMM realizada en esta tesis (ver Sección 2.2), los parámetros a elegir incluyen las restricciones del problema, el tiempo de muestreo del controlador T_s, el horizonte de predicción N, el número de iteraciones de ADMM, la tasa de aprendizaje ρ, el factor de regularización α y la precisión numérica (punto flotante o punto fijo) en la que será implementado el controlador.

  Los valores de los parámetros elegidos, junto con las matrices que son utilizadas en el problema QP y que describen el modelo de la planta simulada, son escritos en los archivos C++ que se encuentran en el repositorio adjunto, los cuales son utilizados en la siguiente etapa para obtener la descripción de hardware.

- **Generación de archivos de hardware**

  Para la obtención de la descripción de hardware se propone el uso de HLS, que genera el código en HDL a partir de una implementación en un lenguaje de más alto nivel, como lo es C++. Por lo tanto, en esta etapa se necesita tener previamente el controlador ya programado en C++ junto con los parámetros y matrices, que se obtienen con procesamiento de texto directamente desde el script de Matlab con el que se realizaron las simulaciones MIL. Con los archivos C++ y utilizando la herramienta Vitis HLS se obtiene la descripción de hardware en algún HDL (específicamente Verilog en este flujo).

  Vitis HLS permite utilizar diversas estrategias para optimizar el hardware diseñado según las restricciones y requisitos del controlador a implementar, mediante directivas específicas denominadas *pragmas*. Además, la herramienta genera un reporte de síntesis en el que se estima la latencia, el *timing* y el uso de recursos, particularmente de LUTs, FFs y DSPs (ver Sección 2.4.1). En general el reporte generado es conservador, por lo que no cumplir con ciertos criterios en esta etapa no necesariamente impedirá la implementación final. Un reporte preciso es generado posteriormente al realizar la implementación en la MicroLabBox.

  El código Verilog generado suele tener un formato que no es compatible con los requerimientos de Vitis Model Composer [41] y la MicroLabBox que son utilizados en etapas posteriores, es por esto que debe hacerse un preprocesamiento de los archivos para que estén en el formato adecuado. Además, se incluyen otros archivos Verilog diseñados previamente que cumplen funciones necesarias para interconectar todos los elementos en las etapas posteriores. Para obtener la descripción final del hardware, se utiliza el script `format_verilogs.py` contenido en el repositorio adjunto para realizar los preprocesamientos correspondientes.

- **SIL**

  En esta etapa se realiza una simulación en lazo cerrado utilizando la descripción final de hardware obtenida en la etapa anterior. Para esto se utiliza Vitis Model Composer [41], una herramienta de AMD que permite simular bloques de RTL utilizando el entorno de Simulink.

  Para utilizar los bloques de RTL en el entorno Simulink, es necesario un archivo de configuración para cada bloque (*System Generator file*). Vitis Model Composer genera un *template*

de estos archivos de configuración para cada bloque de RTL, el cual debe ser modificado. En este flujo, este proceso es automatizado mediante un script de Python que utiliza los *template* y los bloques de RTL respectivos para generar los archivos de configuración finales.

Con esto ya es posible realizar la simulación, donde cada paso de simulación corresponde a un ciclo de reloj de la FPGA, lo que implica que en general las simulaciones SIL toman un tiempo considerable. Si los resultados coinciden con lo obtenido a nivel de MIL, los mismos archivos utilizados en esta etapa se utilizan para las etapas posteriores en la MicroLabBox.

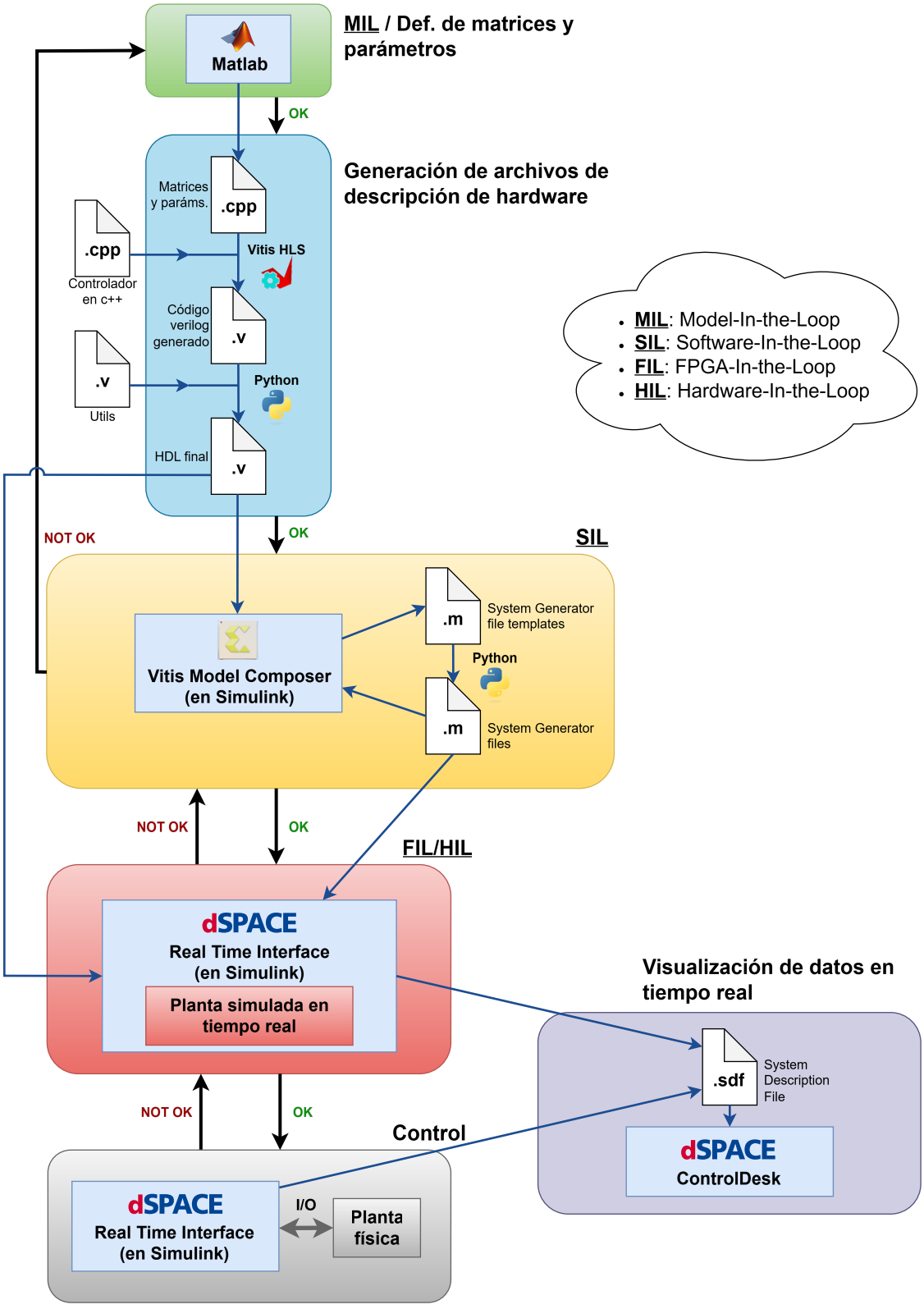


Fig. 3.2: Flujo de diseño para MPC implícito.

3.2.2. Flujo de diseño para MPC explícito con redes neuronales

La Figura 3.3 muestra el flujo completo para MPC explícito con redes neuronales, y cada etapa es explicada en detalle a continuación.

- **MIL(implícito)/Generación de conjunto de datos**

  En esta etapa, en primer lugar debe elegirse un método implícito de MPC, con el cual se obtiene el conjunto de datos que se utilizará para el entrenamiento, validación y prueba de la red neuronal (este método implícito también puede utilizarse para obtener una solución explícita y luego obtener los datos). El método elegido debe validarse nivel de MIL de la misma manera que se explicó anteriormente en el flujo para MPC implícito, eligiendo los parámetros correspondientes antes de generar el conjunto de datos. El proceso de generación de datos puede ser mediante simulaciones en lazo abierto o lazo cerrado, lo cual fue discutido en la Sección 2.3.3.

- **Entrenamiento de red neuronal**

  En esta etapa se utiliza el conjunto de datos obtenidos en la etapa previa para entrenar la red neuronal que aproximará la ley de control utilizando los métodos descritos en la Sección 2.3.3. Este conjunto de datos es normalizado utilizando de preferencia *Min-Max Scaling* (2.3.9), y posteriormente dividido en conjuntos de entrenamiento, de validación y de prueba. Se consideran redes neuronales profundas de tipo *feedforward*, completamente conectadas y con una cantidad constante de neuronas por capa oculta.

  Además, se propone utilizar exclusivamente ReLU como función de activación para obtener una cota para el número de regiones afines que puede representar la red utilizando la ecuación (2.3.8). Con esto se obtiene una heurística para la elección de los hiperparámetros L y M correspondiendo al número de capas ocultas y número de neuronas por capa respectivamente. Para obtener una aproximación del número de regiones que se necesita representar, puede resolverse de antemano el problema utilizando algún método de MPC explícito. Este procedimiento otorga un punto de partida para explorar las elecciones de los hiperparámetros, considerando el error de aproximación, la latencia (que depende principalmente de L) y el uso de recursos físicos que se utilizarán en el dispositivo. Otro aspecto importante que debe considerarse es la precisión numérica que se utilizará para los parámetros de la red, como se explica en la Sección 2.3.4.

  Para estos fines, en este flujo se utiliza Keras, y en particular QKeras [25] que permite diseñar redes neuronales con representaciones de punto fijo para los parámetros de la red y realizar un entrenamiento cuantizado de la red neuronal, es decir, considerar durante la fase de entrenamiento el error de aproximación inducido por el uso de una precisión reducida. Debe considerarse que una implementación con precisión más alta utilizará más recursos y puede resultar en una latencia mayor, mientras que utilizar una precisión más baja aumenta el error de aproximación de la red.

  En cuanto a la función de pérdida, se propone el uso de MSE y un término de regularización que permite disminuir el *overfitting* y disminuir el rango dinámico, aumentando la compatibilidad con precisiones reducidas. La función de pérdida considerada se muestra en la expresión (2.3.11). Para la elección del número de épocas, se utiliza la estrategia de finalizar el entrenamiento cuando la función de pérdida no disminuye consecutivamente durante un número predefinido de épocas, denominado paciencia. Además, se define un número máximo de épocas que permite finalizar el entrenamiento de manera anticipada para limitar el tiempo de entrenamiento. Al final de este proceso, se elige el número de épocas que minimiza la función de pérdida.

Una vez diseñada y entrenada la red, se exportan los parámetros resultantes en formato H5 para ser utilizados en las etapas posteriores. Además, se genera una *golden reference* que corresponde a un conjunto de datos que contiene las salidas de la red entrenada para distintas combinaciones de entradas. Esta *golden reference* se utiliza más adelante para realizar una cosimulación para el RTL generado.

- **MIL (explícito)**

  Una vez obtenidos los parámetros de la red, ya es posible realizar una simulación en lazo cerrado a nivel de MIL para validar que la aproximación de la ley de control es adecuada. Se utiliza QKeras para facilitar la evaluación de la red cuantizada en caso de utilizar representación en punto fijo. Si el desempeño observado no es adecuado, debe volverse a una etapa anterior, ya sea reentrenando la red con distintos hiperparámetros y/o precisión numérica, o generando nuevamente un conjunto de datos más representativo. En caso de éxito, se procede a generar la descripción de hardware utilizando los parámetros de la red.

- **Generación de archivos de descripción hardware**

  En esta etapa, los parámetros de la red ya entrenada y validada a nivel de MIL se utilizan para generar una descripción de hardware funcionalmente equivalente. Para esto se utiliza la herramienta HLS4ML [29], la cual a partir de los parámetros de la red, genera código C/C++ con directivas de optimización para ser procesado por una herramienta de HLS. En particular en este flujo se utiliza Vitis HLS.

  Los parámetros principales que HLS4ML permite ajustar son el nivel de cuantización por capa de la red, la estrategia de optimización a nivel de descripción de HLS y el factor de reutilización. El nivel de cuantización debe coincidir con la implementación en QKeras, la estrategia de optimización puede corresponder a *Latency* o *Resources*, y el factor de reutilización permite elegir cuántas veces se utiliza un mismo multiplicador por capa de la red.

  Una capa oculta de M neuronas de una red completamente conectada contiene M^2 multiplicaciones, por lo que un factor de reutilización de uno (*full unroll*) corresponde a utilizar M^2 multiplicadores simultáneamente en un solo ciclo de reloj, maximizando la velocidad de cómputo a expensas de un mayor uso de recursos. El máximo factor de reutilización sería M^2, que correspondería a utilizar un solo multiplicador para efectuar todas las multiplicaciones de la capa, minimizando el uso de recursos a expensas de una latencia mayor. Para un uso más balanceado de recursos, puede elegirse un factor de reutilización intermedio.

  Una vez generada la descripción en HLS, el RTL final se genera utilizando el mismo procedimiento que en el caso de MPC implícito explicado anteriormente.

- **SIL/Cosimulación**

  En esta etapa se realiza una simulación en lazo cerrado utilizando la descripción final de hardware obtenida en la etapa anterior, de la misma manera que en el caso de MPC implícito explicado anteriormente.

  Además, mediante Vitis Model Composer se efectúa una cosimulación del RTL, utilizando la *golden reference* generada en la etapa de entrenamiento de la red. Esta cosimulación consiste en comparar las salidas de la red implementada en Verilog con las salidas de la red implementada en QKeras, para una lista de entradas generadas aleatoriamente. La *golden reference* contiene las entradas generadas y sus respectivas salidas.

  En caso de que la simulación a nivel de SIL sea consistente con la simulación a nivel de MIL y además la cosimulación sea exitosa, los mismos archivos utilizados en esta etapa se utilizan para las etapas posteriores en la MicroLabBox.

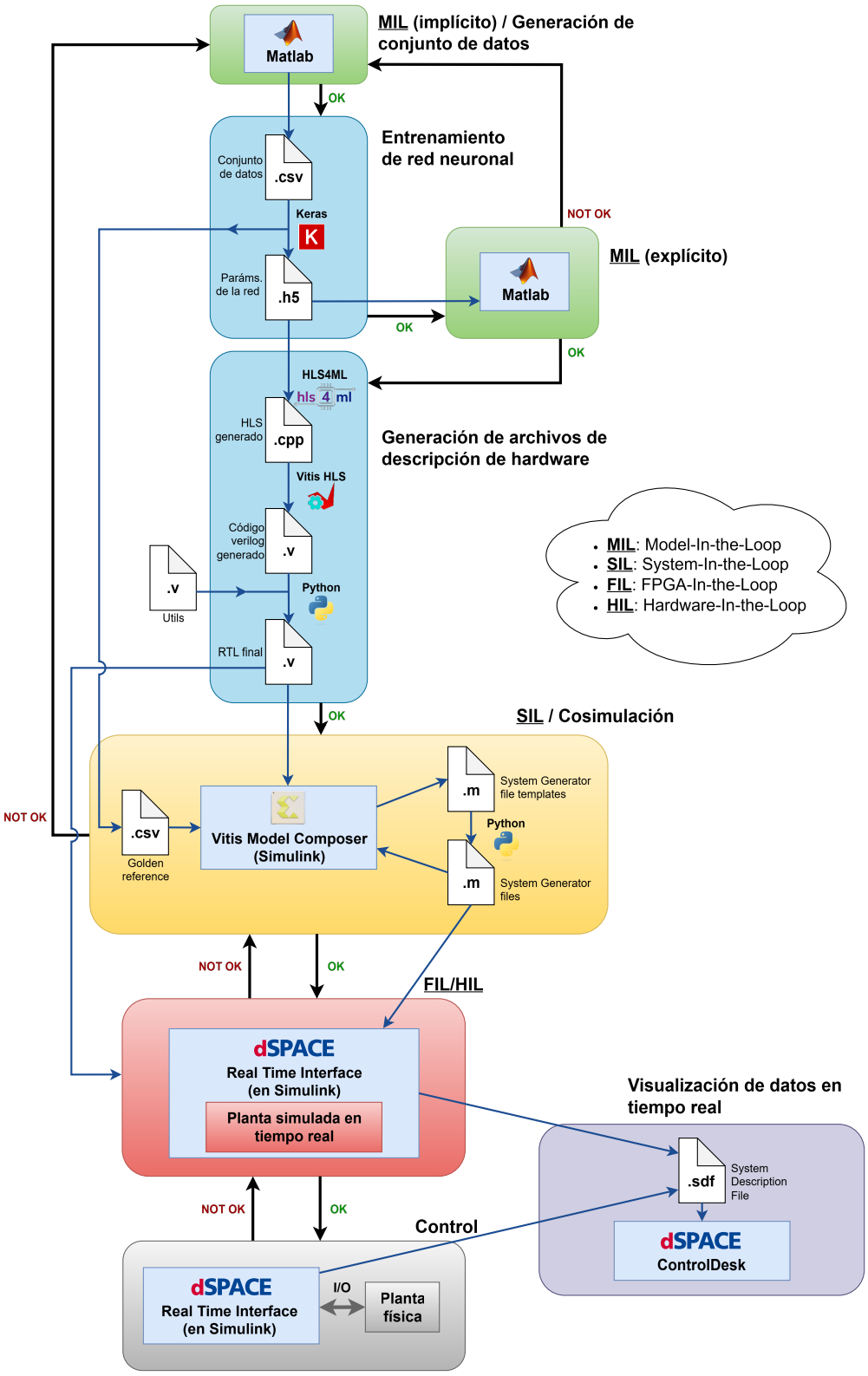


Fig. 3.3: Flujo de diseño para MPC explícito con redes neuronales.

3.2.3. Uso de MicroLabBox en el flujo de diseño

En esta sección se explican los pasos del flujo que utilizan la FPGA, considerando el uso de la MicroLabBox de dSPACE (ver Sección 2.4.2). La integración de la MicroLabBox con el entorno Simulink se hace a través del software Real-Time Interface [36], y la visualización en tiempo real de las señales se hace a través del software ControlDesk [42], ambos de dSPACE. Gracias a estas herramientas, es posible implementar simulaciones en tiempo real con bajo costo, lo que permite aproximarse directamente a escenarios que combinan características tanto de FIL como de HIL.

- **FIL**
  En esta etapa, el dispositivo de control se programa con el mismo código utilizado en la etapa de SIL, y se verifica su equivalencia funcional con el modelo de simulación. Tradicionalmente, FIL busca comprobar el comportamiento bit a bit del código sintetizado en la FPGA, comparándolo con los resultados obtenidos en Simulink o MATLAB. Sin embargo, en el flujo de esta tesis, esta etapa se considera solo de manera conceptual, ya que el uso de la MicroLabBox permite ejecutar directamente simulaciones en tiempo real donde la frontera entre FIL y HIL se vuelve difusa.

- **HIL**
  En esta etapa se utiliza la mayoría del hardware correspondiente a la implementación final, a excepción de la planta física. En el caso de la MicroLabBox, la FPGA ejecuta el algoritmo de control mientras que el procesador integrado simula la planta en tiempo real. De esta forma, se logra un lazo cerrado donde intervienen tanto el hardware de control como la dinámica de la planta simulada, manteniendo un comportamiento en tiempo real. Además, la MicroLabBox permite exponer físicamente las entradas y salidas analógicas, lo que facilita verificar mediante un osciloscopio que las señales corresponden a lo esperado.

  Este escenario constituye un híbrido entre FIL y HIL, pues aunque se prueban las salidas físicas que serán utilizadas en la implementación final, estas no se utilizan para comunicar el controlador con la planta simulada (en su lugar se utilizan los buses internos de la MicroLabBox), pero el procesador y la FPGA si están conectados físicamente entre ellos de la misma manera que en la implementación física. Por motivos de claridad en esta tesis denominaremos a este escenario como HIL.

  Para ello, la programación y configuraciones de la FPGA y del procesador de la MicroLabBox realizadas en el software Real-Time Interface se guardan en un archivo denominado *System Description File*, que contiene la configuración completa del dispositivo para implementar el lazo de control y puede ejecutarse directamente desde ControlDesk. En esta etapa se verifica que el controlador implementado en la FPGA tiene la capacidad de ejecutar el algoritmo en tiempo real con el desempeño esperado y que las salidas en los periféricos físicos es la esperada.. ControlDesk también permite registrar las señales observadas en tiempo real, facilitando el diagnóstico en caso de errores.

- **Control**
  Finalmente, el controlador se conecta a la planta física, completando la implementación del lazo de control. Cualquier ajuste o modificación que se requiera debería pasar nuevamente por las etapas anteriores. El control puede mantenerse en ejecución mediante la MicroLabBox, o considerarse su implementación en un dispositivo embebido más eficiente como paso siguiente.

Capítulo 4

VALIDACIÓN DEL FLUJO DE DISEÑO CON SERVOMOTOR

En este capítulo se reporta una validación experimental preliminar de los flujos de diseño propuestos en el Capítulo 3 para la implementación de lazos MPC implícitos y explícitos basados en redes neuronales. Estos flujos se aplican al control de un servomotor Quanser [43], cuya simplicidad permite verificar la correctitud funcional del controlador, la coherencia numérica entre etapas y la viabilidad de implementación en hardware reconfigurable. Si bien el control del servomotor no exige tiempos de cómputo que justifiquen plenamente la aceleración mediante FPGA, constituye un entorno adecuado para validar sistemáticamente el flujo completo antes de abordar aplicaciones de mayor complejidad. Además, se cuenta con una implementación previa de MPC implícito mediante ADMM en el procesador de la MicroLabBox [44], la cual se utiliza como línea base para comparar desempeño computacional y comportamiento en lazo cerrado.

El capítulo presenta los resultados considerando distintas precisiones en punto fijo y siguiendo las etapas de *Model-in-the-Loop*, *Software-in-the-Loop*, *Hardware-in-the-Loop* y la implementación física, permitiendo evaluar de manera estructurada el impacto de las decisiones de diseño sobre el desempeño final del sistema.

4.1. Planteamiento MPC para servomotor

Para el servomotor, los estados que se considerarán son la velocidad x_1 (o v) en [rad/s] y la posición angular x_2 en radianes. De esta manera, las matrices utilizadas para el modelo en tiempo discreto para un período de muestreo h son:

$$A = \begin{bmatrix} e^{-ah} & 0 \\ \frac{1-e^{-ah}}{a} & 1 \end{bmatrix},\ B = \begin{bmatrix} \frac{K}{a} \cdot \left(1 - e^{-ah}\right) \\ \frac{K}{a} \cdot \left(\frac{e^{-ah}}{a} - \frac{1}{a} + h\right) \end{bmatrix},$$
$$C = [0\ \ 1]\,,$$

donde experimentalmente se determinó que $a = 28.8582$ y $K = 45.0051$ [44]. Para esta planta se cuenta solo con un sensor de posición, y además la entrada tiene una zona muerta de aproximadamente $0.3[V]$, es decir, que voltajes bajo este umbral resultan en una actuación nula. Esto conlleva a problemas de seguimiento en estado estacionario, por lo que se utiliza un observador aumentado con acción integral, como el planteado en la expresión (2.1.15). Utilizando las expresiones (2.1.32) y (2.1.35) con las matrices consideradas para el servotomor, se obtiene que la actuación a aplicar al

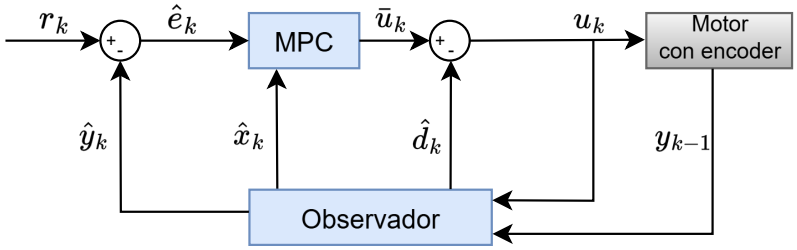


Fig. 4.1: Diagrama de bloques del lazo de control

controlador puede calcularse como $u_k = \bar{u}_k - \hat{d}_k$, es decir, como la diferencia entre la actuación calculada directamente con MPC y la perturbación estimada por el observador. La Figura 4.1 muestra un diagrama de bloques del lazo de control, utilizando la misma notación que en la Sección 2.1.1.

Se consideran referencias de 10 y 350 grados para la posición x_2, además de una restricción de velocidad tal que $x_1 \in [-2, 2]$ y una restricción en la actuación (en Volts) tal que $u \in [-10, 10]$. Todas las simulaciones se realizan en Matlab y se considera un período de muestreo de control de 12[ms] y un período de muestreo de la planta de 1[ms].

Además, en los gráficos se usan las siguientes abreviaciones para las leyendas:

1. Referencia [rad]: `ref`.
2. Posición [rad]: `pos`.
3. Posición estimada [rad]: `pos_est`.
4. Velocidad [rad/s]: `vel`.
5. Velocidad estimada [rad/s]: `vel_est`.
6. Señal de control [V]: `u`.
7. Perturbación estimada [V]: `pert`.

4.2. Diseño, simulación e implementación de lazos MPC implícitos

4.2.1. Simulaciones Model-in-the-Loop para MPC implícito

Para la implementación del lazo de control implícito, se consideran los parámetros que se muestran en la Tabla 4.1. La Figura 4.2 muestra los resultados de la simulación MIL, y con esta elección de parámetros se observa que se logra un control correcto al utilizar precisión de punto flotante, es decir que se cumple la restricción de velocidad y la referencia se sigue con un error en estado estacionario que es despreciable para efectos prácticos. En los gráficos se muestran los estados del motor, junto con la referencia y la actuación, y también se muestran por separado los estimados estimados, incluyendo a la perturbación estimada. Se observa que la velocidad se satura correctamente en el

Tabla 4.1: Parámetros ADMM para implementación implícita

h [ms]	N	iters.	ρ	α	*warm-start*
12	4	20	0.7526	0.8	Sí

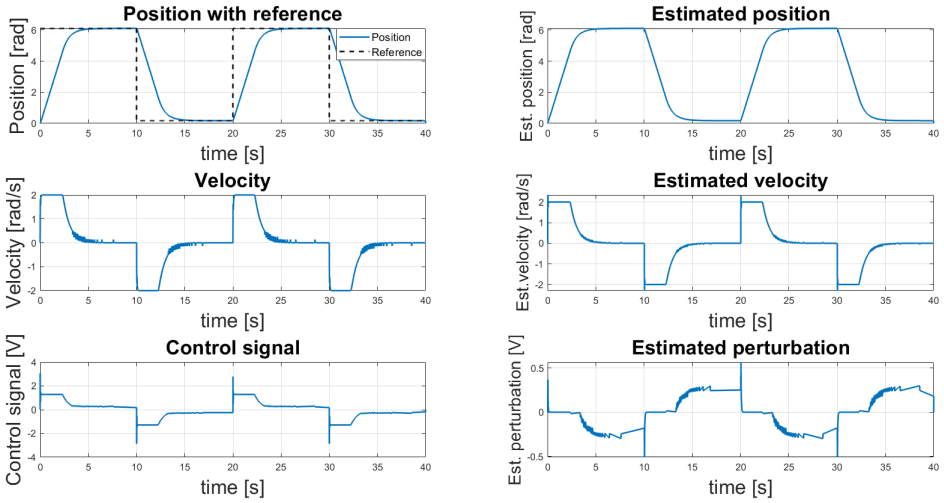


Fig. 4.2: Simulación MIL de MPC implícito para servomotor utilizando punto flotante. En la columna izquierda de muestran los estados y la señal de control, y en la columna derecha se muestran las señales estimadas.

valor de la restricción, y la Figura 4.3 muestra un acercamiento en las zonas de saturación donde se evidencia el cumplimiento de las restricciones. En la Figura 4.2 también se observa que los estados se estiman correctamente y la perturbación es estimada acorde a la zona muerta de 0.3[V]. Se evidencia además la efectividad del observador con acción integral para mitigar el efecto de la zona muerta, ya que gracias a la estimación de la perturbación, la actuación aumenta hasta alcanzar los 0.3[V], lo que provoca una velocidad no nula por un breve instante de tiempo y acerca cada vez más la posición a la referencia. Notar que la estimación de la velocidad tiene un *overshoot* que no cumple con la restricción por un breve instante, sin embargo esto no sucede con la velocidad de la planta.

Para la implementación en hardware, el uso de aritmética en punto flotante implica una alta demanda de recursos físicos. Por esta razón, suele ser preferible emplear aritmética en punto fijo. En este trabajo se realizan simulaciones en Matlab empleando precisión de punto fijo mediante `fimath` [45], utilizando la opción de `Round` como método de redondeo, saturación en el caso de *overflow*, los argumentos de `Productmode` y `SumMode` se configuran como `SpecifyPrecision`, y los argumentos de `ProductWordLength` y `SumWordLength` se configuran con el mismo largo de bits que los operandos (de esta manera el resultado siempre tiene la misma precisión que los operandos). Por simplicidad se adopta la misma precisión tanto para el controlador como para el estimador de estados. Se comienza con una configuración de 32 bits de ancho total ($W = 32$) y 8 bits asignados a la parte entera con signo ($Q = 8$), la cual calza con el largo de bits de los buses interno de la MicroLabBox, simplificando su implementación. Con esta configuración no se producen *overflows* y el desempeño obtenido es muy similar al de la versión en punto flotante, sirviendo como punto de comparación con las precisiones inferiores en las etapas posteriores. La Figura 4.4a presenta los

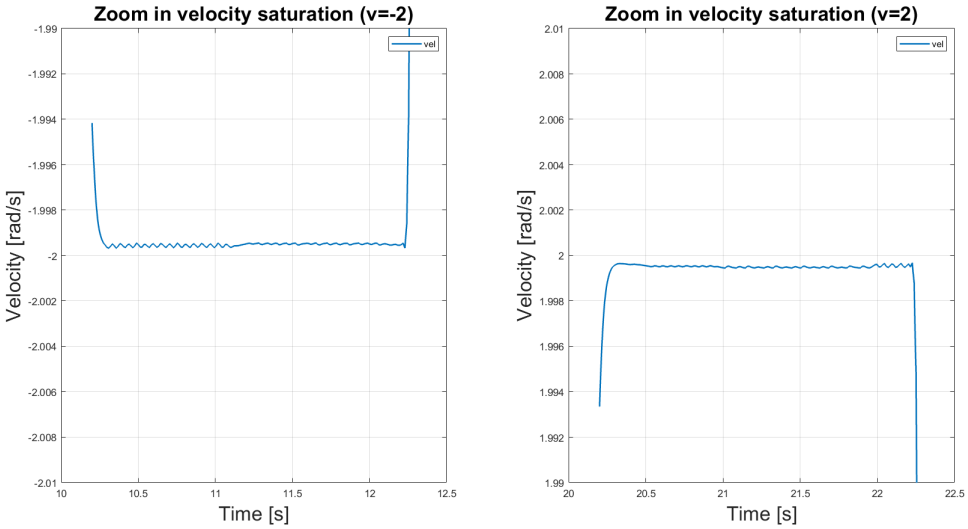
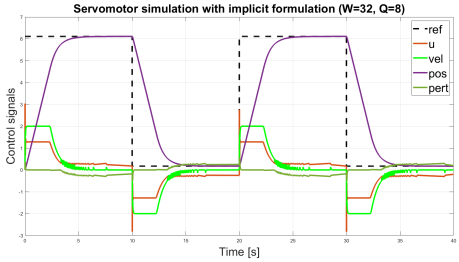


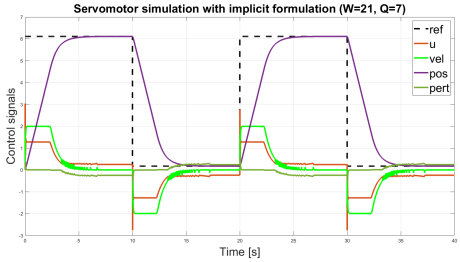
Fig. 4.3: Zoom en zonas de saturación de velocidad.

resultados del control.

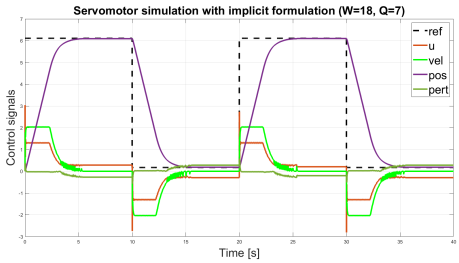
Con el objetivo de reducir el uso de recursos en la implementación física, es posible emplear representaciones de menor precisión, asumiendo una degradación en el desempeño. Tras realizar pruebas, se determina que el valor mínimo para evitar *overflows* para la parte entera con signo es de 7 bits ($Q = 7$). Para el ancho total, se evalúan las configuraciones $W \in \{21, 18, 16\}$. La Figura 4.4b muestra los resultados obtenidos con $W = 21$, donde se observa un desempeño comparable al de punto flotante. Con $W = 18$, mostrado en la Figura 4.4c, el desempeño comienza a deteriorarse más, incumpliendo levemente la restricción de velocidad en saturación. Finalmente, con $W = 16$, cuyos resultados se presentan en la Figura 4.4d, el deterioro en el desempeño es notorio, alcanzando magnitudes de velocidad en saturación excesivamente menores a las restricciones y un leve error en estado estacionario. La Figura 4.5 muestra una comparación entre las actuaciones, posiciones y velocidades de los casos $W = 32$ y $W = 16$, y se observa que el caso $W = 16$ muestra una actuación y velocidad de menor magnitud en saturación que el caso $W = 32$, y un leve error en estado estacionario de la posición. Una menor velocidad implica una menor pendiente en la posición y por lo tanto también un asentamiento más lento en la referencia.



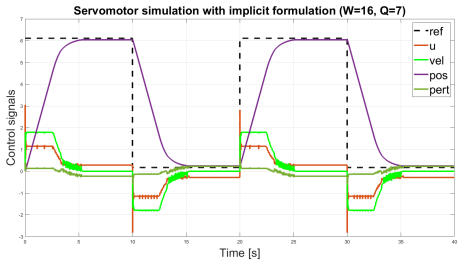
(a) Simulación MIL de servomotor con formulación implícita con punto fijo ($W = 32$, $Q = 8$).



(b) Simulación MIL de MPC implícito para servomotor utilizando punto fijo ($W = 21$, $Q = 7$).



(c) Simulación MIL de MPC implícito para servomotor utilizando punto fijo ($W = 18$, $Q = 7$).



(d) Simulación MIL de MPC implícito para servomotor utilizando punto fijo ($W = 16$, $Q = 7$).

Fig. 4.4: Comparación de simulaciones MIL de MPC implícito para servomotor con distintas configuraciones de punto fijo. Unidades: `ref` y `pos` en rad, `u` y `pert` en volts y `vel` en rad/s.

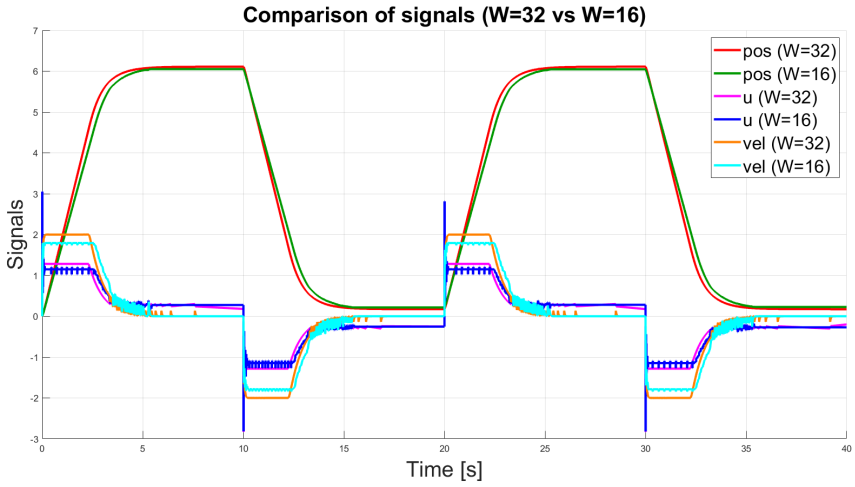


Fig. 4.5: Comparación de posición, actuación y velocidad para simulaciones HIL de MPC implícito ($W = 32$ y $W = 16$). Unidades: `pos` en rad, `u` en volts y `vel` en rad/s.

4.2.2. Simulaciones Software-in-the-Loop para MPC implícito

Obtención de descripciones de hardware

Ya obtenidos los parámetros a nivel de simulación, se procede a generar las descripciones de hardware de los controladores implícitos a implementar. Para esto se utiliza Vitis HLS, utilizando las matrices obtenidas a nivel de MIL para escribir el código C++. Se generan por separado las descripciones de los estimadores y los controladores, y el uso de recursos estimados a nivel de *HLS Synthesis* se muestran en las Tablas 4.2 y 4.3 respectivamente. Se observa que la mayoría de los recursos físicos y latencia son utilizados para el controlador, en comparación con el estimador. Además, se observa que los recursos físicos aumentan significativamente con el ancho de bits utilizado, pero la latencia no sigue un patrón evidente. Debe considerarse que los reportes a nivel de *HLS Synthesis* son estimaciones gruesas y suelen sobreestimar con respecto al reporte de implementación.

Simulación SIL

Una vez obtenidas las descripciones en Verilog, estas deben ser preprocesadas para poder simularse a nivel de SIL con Vitis Model Composer, y también para utilizarse en las etapas posteriores. Para esto se utiliza un script de Python que está en el repositorio adjunto a este trabajo. También deben modificarse las *System Generator Files* que genera Vitis Model Composer. La Figura 4.6 muestra el diagrama utilizado en Vitis Model Composer (siguiendo la misma estructura general de la Figura 4.1), y en esta se incluye un generador de pulsos periódicos que da inicio a los ciclos de control, la planta simulada como función de Matlab, el estimador de estados y el controlador implícito.

Vitis Model Composer funciona en el entorno Simulink, y cada paso de simulación corresponde a un ciclo de reloj de la FPGA simulada. En la práctica, la FPGA opera a una frecuencia de 100[MHz], por lo que para simular 40 segundos se requerirían 4 mil millones de pasos de simulación. Con el objetivo de disminuir el número de ciclos necesarios, se considera que la FPGA simulada opera a una frecuencia más baja. Para el caso $W = 21$, el lazo completo toma 1780 ciclos de reloj de la FPGA, y considerando que el período de muestreo de control es de $h = 12$[ms], puede considerarse que cada paso de simulación (o ciclo simulado de la FPGA) representa un paso temporal de $T_{fpga} = \frac{h}{1780} \approx 6.74$ [ms]. Este tiempo también se usa para discretizar a la planta simulada, ya que sus estados se actualizan en cada paso de simulación. Finalmente, para lograr $40[s]$ de simulación se requerirían $T = \frac{40[s]}{T_{fpga}} \approx 5.93 \cdot 10^6$ pasos de simulación.

La Figura 4.7 muestra la simulación SIL para $W = 21$, y la Figura 4.8 muestra una comparación de estas señales con lo obtenido a nivel de MIL. Se observa que el comportamiento a nivel de SIL es consistente con lo observado a nivel de MIL, con diferencias despreciables para efectos prácticos. En particular, en las simulaciones SIL no se produce el *overshoot* de la velocidad estimada y no se producen cambios abruptos en la perturbación estimada en los cambios de referencia.

Tabla 4.2: Recursos y latencias de estimadores reportados a nivel de *HLS Synthesis*.

W	Q	LUTs	FFs	DSPs	Latencia [μs]*
32	8	2748	1946	15	0.3
21	7	2171	1000	5	0.3
18	7	2104	1064	5	0.31
16	7	2010	974	5	0.31

*Con reloj de frecuencia 100[MHz].

Tabla 4.3: Recursos y latencias de controladores implícitos reportados a nivel de *HLS Synthesis*.

W	Q	LUTs	FFs	DSPs	Latencia [μs]*
32	8	41120	10730	136	16.83
21	7	31165	4998	61	17.66
18	7	29649	4242	42	17.44
16	7	27846	3927	33	16.24

*Con reloj de frecuencia 100[MHz].

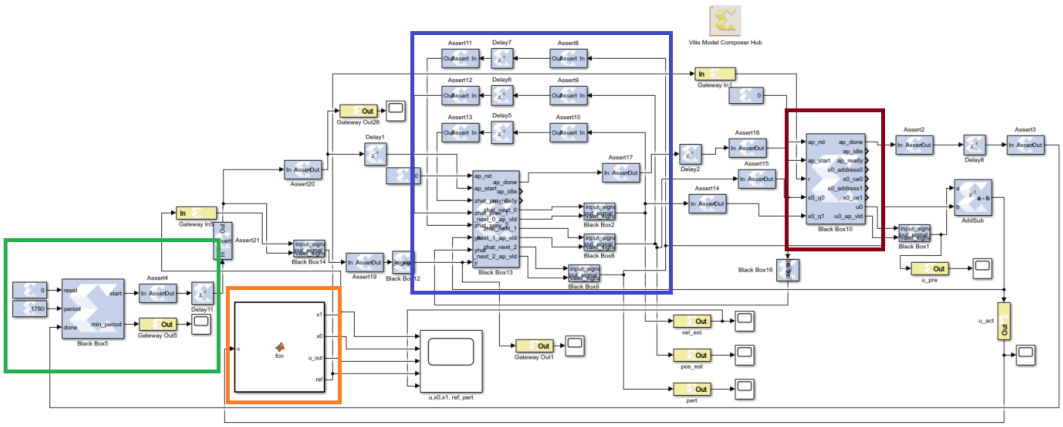


Fig. 4.6: Diagrama de simulación SIL implícita en Vitis Model Composer. El cuadrado verde contiene el generador de pulsos periódicos, el naranjo contiene la planta simulada como función de Matlab, el azul contiene el estimador de estados y el rojo contiene al controlador implícito.

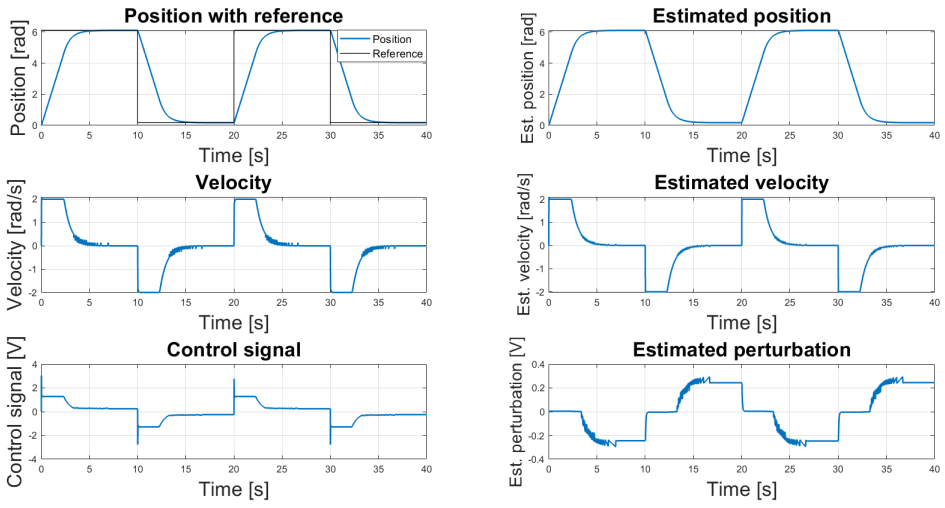


Fig. 4.7: Simulación SIL para MPC implícito ($W = 21, Q = 7$). En la columna izquierda de muestran los estados y la señal de control, y en la columna derecha se muestran las señales estimadas.

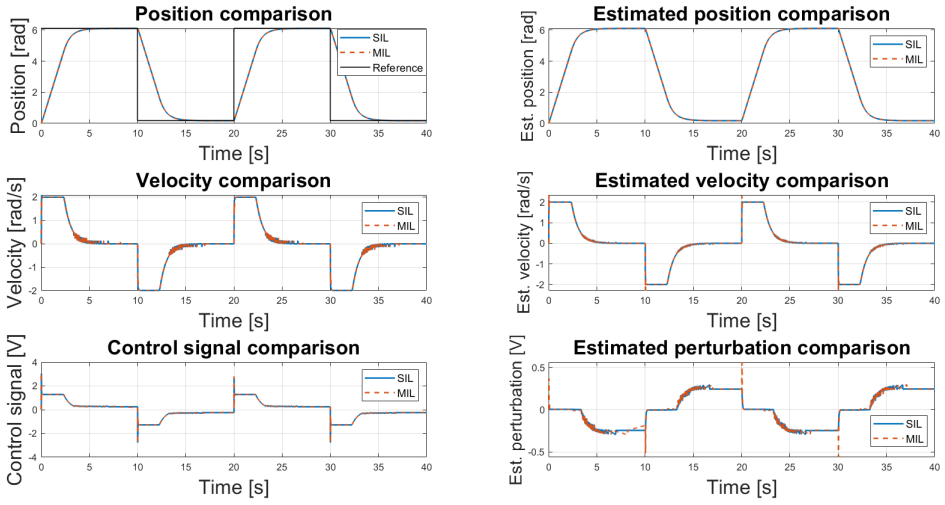


Fig. 4.8: Comparación de simulaciones MIL y SIL para MPC implícito ($W = 21, Q = 7$). En la columna izquierda de muestran los estados y la señal de control, y en la columna derecha se muestran las señales estimadas.

4.2.3. Simulaciones Hardware-in-the-Loop para MPC implícito

Para las simulaciones HIL (ver Sección 3.2.3 para los detalles de la definición del término HIL en este contexto), se utiliza la MicroLabBox distribuyendo las tareas entre el procesador y la FPGA como muestra la Figura 4.9.

La Figura 4.10 muestra el diagrama de bloques utilizado para configurar la FPGA. En esta se observan el controlador, el estimador de estados y un módulo que genera pulsos períodicos cada 12[ms] para dar inicio a los ciclos de control. El resto de bloques corresponden a `FPGA Setup` (para editar opciones de la configuración de la FPGA), conversión entre tipos de señales y bloques de comunicación con el procesador.

La Figura 4.11 muestra el diagrama de bloques utilizado para programar el procesador de la MicroLabBox. Sus tareas principales incluyen la simulación de la planta, la generación de la referencia y la comunicación con la FPGA. Estas tareas se realizan cada 1[ms], aunque la actuación solo se actualiza cada 12[ms] por la FPGA. La Figura 4.12 muestra cómo se visualizan las señales en tiempo real utilizando ControlDesk. Posteriormente estas pueden exportarse para ser analizadas y graficadas.

La MicroLabBox tiene dos *frameworks* para programar la FPGA:

- `DS1202 FPGA I/O Type 1`: no deja disponibilidad de los pines I/O de la FPGA para ser utilizados en el procesador, pero permite utilizar casi todos los recursos de la FPGA para la implementación. Además, esta la implementación tiene una demora considerablemente menor en comparación al *framework* `DS1202 FPGA I/O Type 1 (Flexible I/O)`.
- `DS1202 FPGA I/O Type 1 (Flexible I/O)`: permite acceder a los pines I/O de la FPGA desde el procesador, pero a cambio tiene una alta utilización base de recursos, dejando menor disponibilidad de la FPGA para la implementación (ver Tabla 4.4 como ejemplo) y además demorando un tiempo considerablemente mayor en realizar la implementación en comparación al *framework* `DS1202 FPGA I/O Type 1`. En particular, el *blockset* `RTI Electric Motor` (EMC) es un conjunto de bloques disponibles para el procesador que permiten utilizar parte de la FPGA para aplicaciones con motores, y estos bloques solo están disponibles al utilizar este *framework*, ya que utiliza pines I/O de la FPGA. Dado que para la implementación física

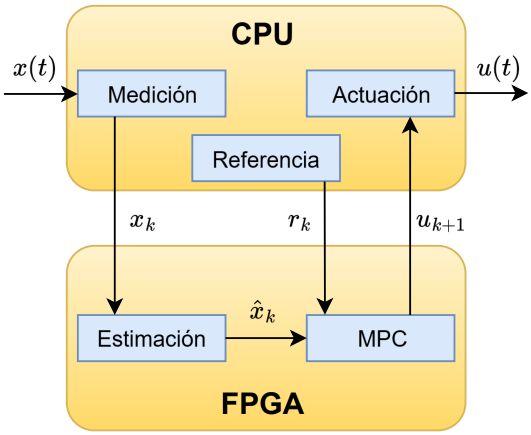


Fig. 4.9: Tareas realizadas en la MicroLabBox.

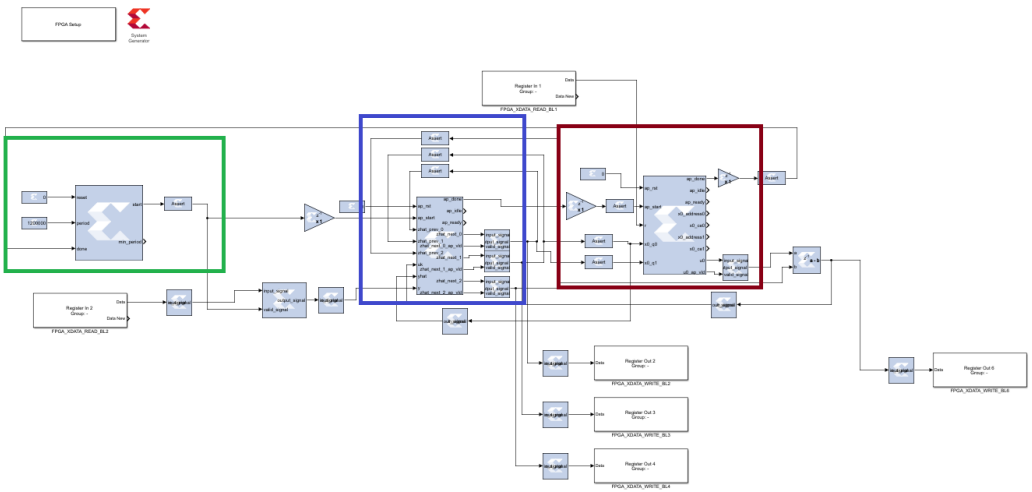


Fig. 4.10: Implementación implícita en FPGA. El cuadro rojo contiene el controlador, el azul el estimador de estados y el verde un módulo que genera pulsos periódicos cada 12[ms].

Tabla 4.4: Recursos utilizados para implementaciones implícitas para distintos niveles de cuantización y para distintos *frameworks* de la FPGA.

FW		`DS1202 FPGA I/O Type 1`			`DS1202 FPGA I/O Type 1 (Flexible I/O)`		
W	**Q**	**LUTs**	**FFs**	**DSPs**	**LUTs**	**FFs**	**DSPs**
32	8	23336	14708	150	136111	109551	292
21	7	16706	12764	90	129420	107607	232
18	7	15569	12093	71	128272	106938	213
16	7	14960	11608	60	127635	106453	202
Disp.		**203800**	**407600**	**840**	**203800**	**407600**	**840**

se necesita un *encoder* para medir la posición del motor, esta debe realizarse utilizando este *framework*.

Para las simulaciones HIL no se requiere el uso del *encoder* del motor, ya que la planta es simulada. Esto permite utilizar el *framework* `DS1202 FPGA I/O Type 1` de la FPGA, dejando disponibilidad casi completa de los recursos físicos. La Tabla 4.4 muestra los recursos utilizados para cada implementación según el nivel de cuantización.

Las simulaciones HIL realizadas se muestran en las Figuras 4.13a, 4.13b, 4.13c y 4.13d. A diferencia de las otras simulaciones, en estos gráficos la actuación que se muestra es la actuación efectiva después de la zona muerta. Los casos $W = 32$ y $W = 21$ coinciden con lo observado a nivel de MIL, mostrando un comportamiento correcto. El caso $W = 18$ muestra mayor ruido en la velocidad estimada y la actuación. Para el caso $W = 16$ se observa una velocidad en saturación excesivamente menor que la restricción y un leve error en estado estacionario, tal como se observó a nivel de MIL. La Figura 4.14 muestra una comparación entre las actuaciones, posiciones y velocidades de los casos $W = 32$ y $W = 16$ (trasladando temporalmente las señales para que coincidan las referencias), y se observa que al igual que a nivel de MIL, el caso $W = 16$ muestra una actuación y velocidad de menor magnitud en saturación que las demás implementaciones, y un leve error en estado estacionario de

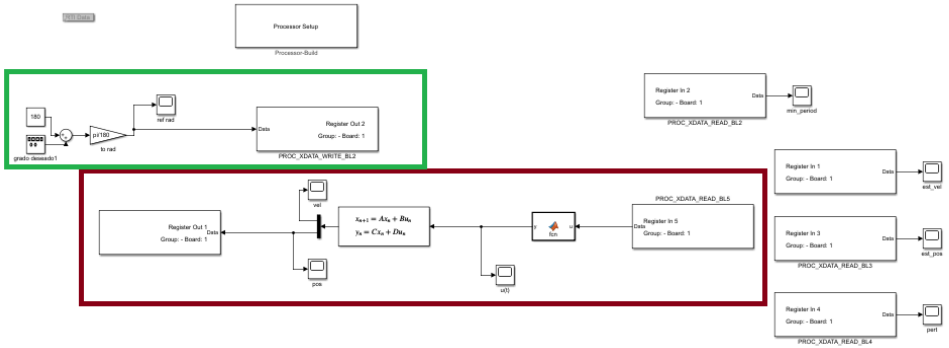


Fig. 4.11: Tareas del procesador para simulaciones HIL. El cuadro rojo contiene la planta simulada y el verde la generación de la referencia.

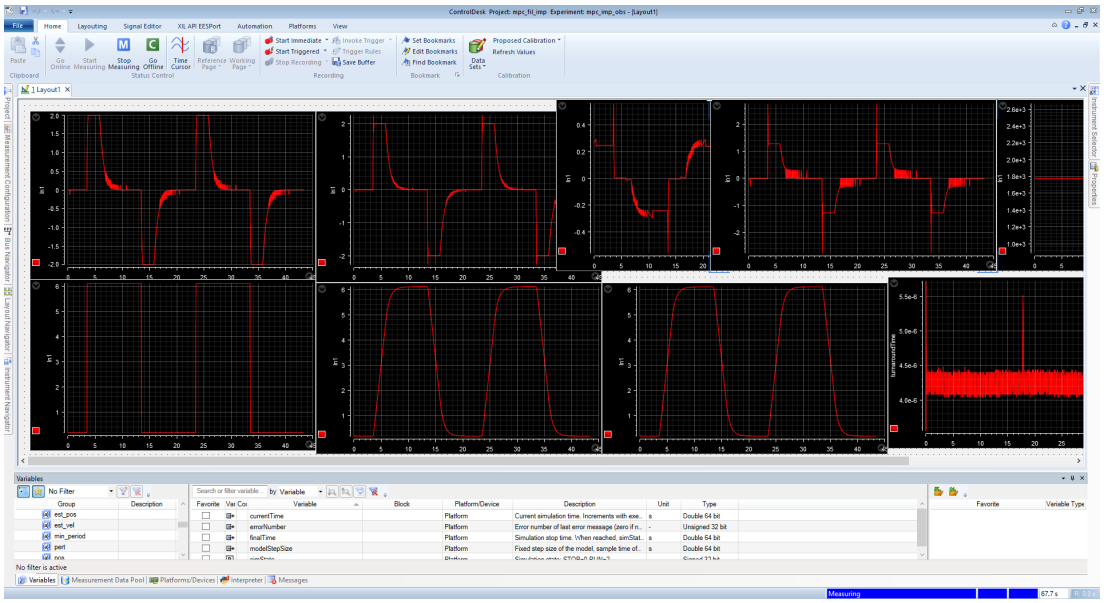
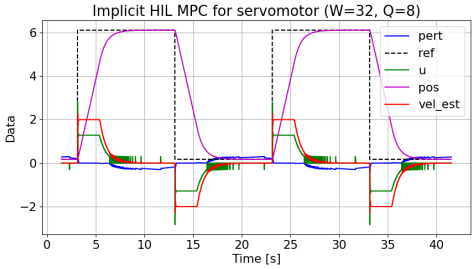
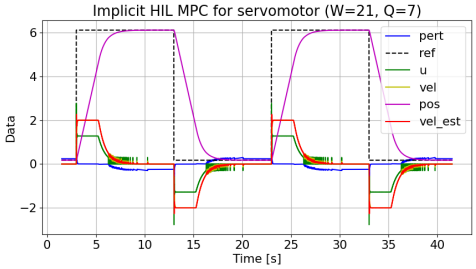


Fig. 4.12: Ejemplo de visualización de señales en tiempo real en ControlDesk.

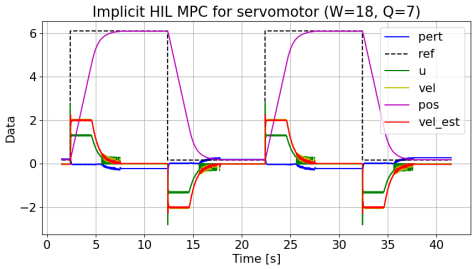
la posición. Una menor velocidad implica una menor pendiente en la posición y por lo tanto también un asentamiento más lento en la referencia.



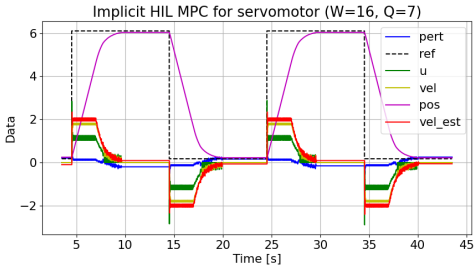
(a) Simulación HIL de MPC implícito ($W = 32, Q = 8$).



(b) Simulación HIL de MPC implícito ($W = 21, Q = 7$).



(c) Simulación HIL de MPC implícito ($W = 18, Q = 7$).



(d) Simulación HIL de MPC implícito ($W = 16, Q = 7$).

Fig. 4.13: Comparación de simulaciones HIL de MPC implícito con distintas configuraciones de punto fijo. Unidades: `ref` y `pos` en rad, `u` y `pert` en volts y `vel` y `vel_est` en rad/s.

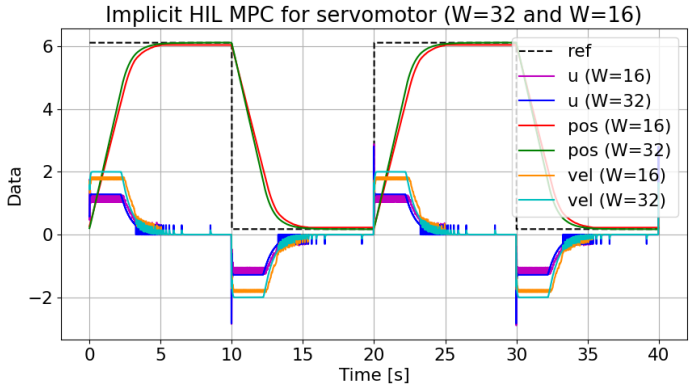


Fig. 4.14: Comparación de posición, actuación y velocidad para simulaciones HIL de MPC implícito ($W = 32$ y $W = 16$). Unidades: `ref` y `pos` en rad, `u` en volts y `vel` en rad/s.

4.2.4. Implementaciones implícitas en FPGA

Para la implementación física de los lazos de control se utiliza el setup experimental que muestra la Figura 4.15. Este incluye la MicroLabBox, la cual se conecta a un amplificador cuya salida es la señal que influye físicamente en el servomotor. Se muestra además el computador y monitor que

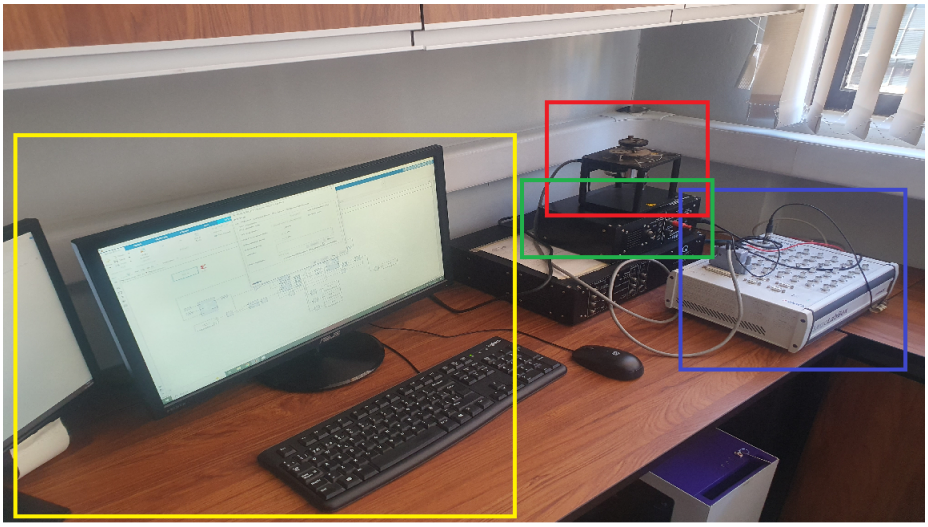


Fig. 4.15: Setup experimental del servomotor. El cuadrado azul contiene a la MicroLabBox, el rojo al servomotor, el verde el amplificador y el amarillo la interfaz para el control.

funciona como interfaz gráfica para implementar los lazos de control.

La implementación en la FPGA es la misma que se utilizó a nivel de HIL, que se muestra en la Figura 4.10. La programación del procesador se muestra en la Figura 4.16 y es distinta que la utilizada a nivel de HIL, ya que no se requiere la simulación de la planta, y además ahora es necesario utilizar un conversor digital-analógico para enviar la actuación y el *encoder* para leer la posición angular del servomotor. Este *encoder* (específicamente del *blockset* `RTI Electric Motor` (EMC)) se utiliza desde el procesador de la MicroLabBox, pero utiliza parte de la FPGA disponible. Esto último hace necesario utilizar el *framework* `DS1202 FPGA I/O Type 1 (Flexible I/O)` de la FPGA (ver Sección 4.2.3 para una descripción de los *frameworks* disponibles), aumentando considerablemente el uso base de recursos físicos y el tiempo de compilación de las implementaciones. Esto sucede porque para utilizar partes de la FPGA desde el procesador la herramienta obliga a programar la FPGA de manera que queden todos sus pines no utilizados disponibles desde el procesador.

La Figura 4.9 muestra las tareas realizadas por el procesador y la FPGA. Al igual que en las simulaciones HIL, el procesador realiza sus tareas cada $1[ms]$ y la FPGA actualiza la actuación cada $12[ms]$. Aunque tanto para el procesador como para la FPGA estas tareas están el orden de los microsegundos, este tiempo de muestreo se elige porque es adecuado para el motor. Tiempos de muestro bajo $1[ms]$ resultaron en errores asociados al observador, ya que la planta no reacciona tan rápido y se generan errores en las estimaciones de estados.

Las implementaciones para los cuatro niveles de cuantización se muestran en la Figura 4.17. Los resultados son similares a lo observado a nivel de HIL, pero en la implementación no es posible visualizar la velocidad de la planta. En general las velocidades estimadas y la actuación presentan mayor ruido. La Figura 4.18 muestra una comparación entre las actuaciones y posiciones de los casos $W = 32$ y $W = 16$ (trasladando temporalmente las señales para que calcen las referencias), y se observa que al igual que a nivel de HIL, el caso $W = 16$ muestra una actuación de menor magnitud en saturación que las demás implementaciones y un leve error en estado estacionario. Aunque no es posible visualizar la velocidad de la planta, para el caso el caso $W = 16$ se observa una menor pendiente en la posición y por lo tanto también un asentamiento más lento en la referencia.

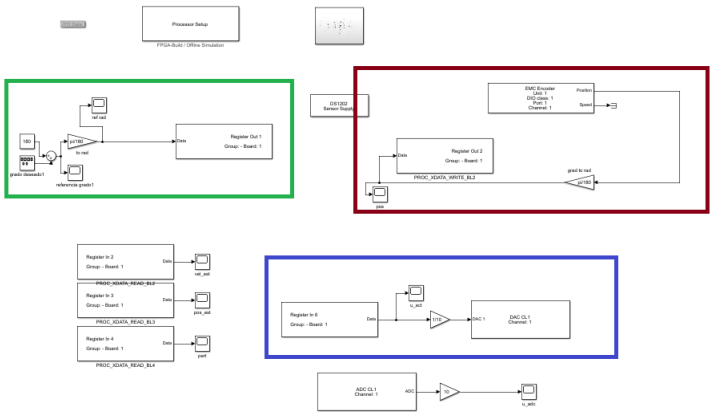
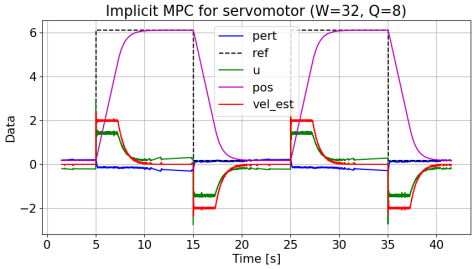
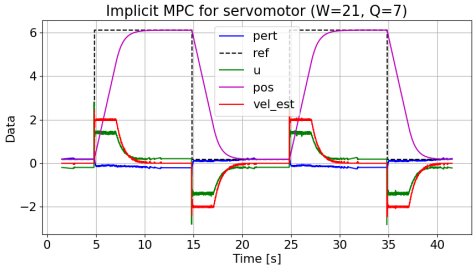


Fig. 4.16: Tareas del procesador para implementaciones físicas implícitas y explícitas. El cuadro rojo contiene el bloque del *encoder*, el verde la generación de la referencia y el azul el envío de la actuación por el conversor digital-analógico.

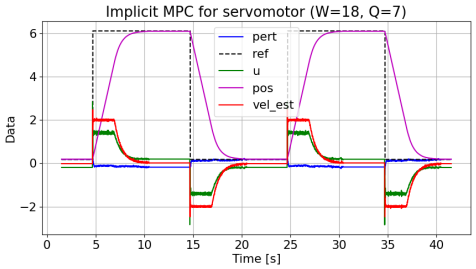
El uso de recursos se reporta en la Tabla 4.4, en las columnas correspondientes al *framework* `DS1202 FPGA I/O Type 1 (Flexible I/O)`. En comparación al *framework* utilizado a nivel de HIL, se observa un uso base bastante más elevado de todos los recursos reportados. La diferencia de utilización entre implementaciones es similar para ambos *frameworks*.



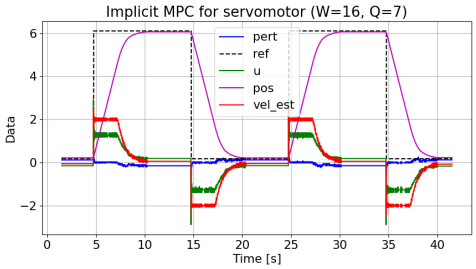
(a) Implementación física de MPC implícito con FPGA ($W = 32, Q = 8$).



(b) Implementación física de MPC implícito con FPGA ($W = 21, Q = 7$).



(c) Implementación física de MPC implícito con FPGA ($W = 18, Q = 7$).



(d) Implementación física de MPC implícito con FPGA ($W = 16, Q = 7$).

Fig. 4.17: Comparación de implementaciones físicas de MPC implícito con FPGA para distintas configuraciones de punto fijo. Unidades: `ref` y `pos` en rad, `u` y `pert` en volts, y `vel` y `vel_est` en rad/s.

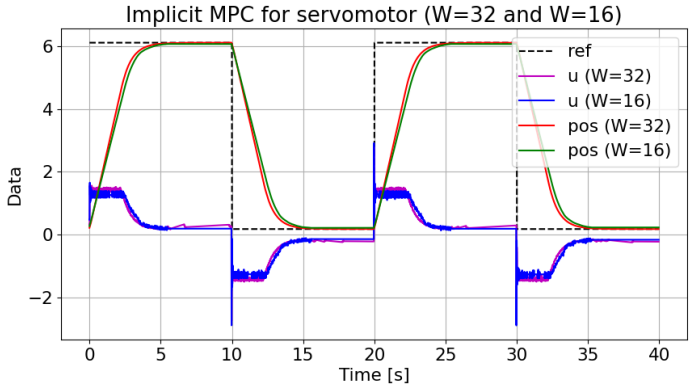


Fig. 4.18: Comparación de actuación y posición para implementaciones de MPC implícito ($W = 32$ y $W = 16$). Unidades: `ref` y `pos` en rad y `u` en volts.

Tabla 4.5: Parámetros de ADMM para datasets

h [ms]	N	iters.	ρ	α	*warm-start*
12	4	250	0.7526	0.8	No

Tabla 4.6: Hiperparámetros utilizados

Hiperparámetro	Flotante	Punto fijo
Optimizador	Adam	Adam
Learning rate	0.005	0.001
Batch size	16	16
Regularización	L1, ganancia 10^{-5}	L1, ganancia 10^{-5}
Épocas	paciencia 20	paciencia 3

4.3. Diseño, simulación e implementación de lazos MPC explícitos con DNNs

4.3.1. Simulaciones Model-in-the-Loop para MPC explícito

Entradas y salidas de la DNN

Utilizando la estructura explicada en la Sección 2.3.2 para las DNNs, se considera como entrada el vector $x_r = [x_1, x_2, r]^{\mathrm{T}} \in \mathbb{R}^3$, que corresponde a la concatenación de los estados del motor y la referencia, y como salida se considera la actuación óptima estimada $\hat{u} \in \mathbb{R}$.

Datasets e hiperparámetros

Para la generación de los datasest de entrenamiento de las redes neuronales se utiliza la formulación implícita con los parámetros que se muestran en la Tabla 4.5, con simulaciones en lazo abierto. Notar que no se utiliza *warm-start*, por lo que en comparación con la implementación implícita, se requieren más iteraciones para lograr que se cumplan los requisitos de control adecuadamente. Las redes se entrenan con Keras para punto flotante y QKeras para punto fijo, utilizando datasets que se dividen en 75 % para entrenamiento, 10 % para validación y 15 % para test. La Tabla 4.6 muestra los hiperparámetros utilizados.

Se crean dos datasets (A y B), generando aleatoriamente (con distribución uniforme) posiciones y referencias en el rango $[0, 2\pi]$ y velocidades en el rango $[-10, 10]$, normalizados con mín-máx. Además, una fracción del dataset B es generada cerca de los niveles de saturación de velocidad, en márgenes de $\pm 0.1[rad/s]$. Para cada dataset se entrenan 24 redes neuronales, combinando los parámetros $L \in \{2, 3, 4, 5\}$ y $M \in \{5, 6, 7, 8, 9, 10\}$. La Tabla 4.7 muestra la información relevante de cada dataset.

La Figura 4.19 y la Figura 4.20 muestran los RMSE menores a 0.05 obtenidos para las redes entrenadas con el dataset A y B respectivamente. Los mejores parámetros no siempre son los que minimizan el RMSE, por ejemplo la red $L = 5, M = 9$ tiene un RMSE ligeramente menor a la red $L = 3, M = 9$, pero esta última tiene un mejor desempeño en simulación en torno a las zonas de saturación, además de ser de menor dimensión.

Las Figuras 4.21 y 4.22 muestran el comportamiento en la zona de saturación $v = 2$ y $v = -2$ respectivamente, considerando las redes con los mejores parámetros de ambos datasets, indicados en

Tabla 4.7: Datasets de entrenamiento

Dataset	Número de datos		Mejores paráms.		
	Total	En sat.	L	M	RMSE (10^{-3})
A	120k	0	4	9	1.625
B	120k	20k	3	9	1.569

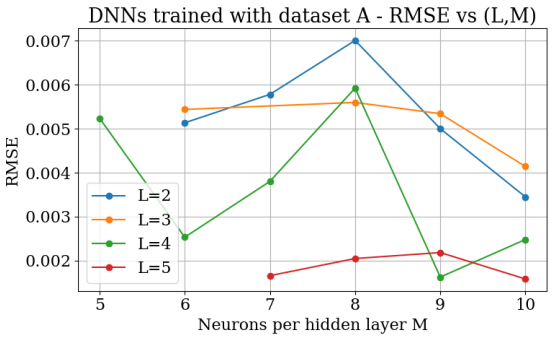


Fig. 4.19: RMSEs menores a 0.05 de redes entrenadas con el dataset A.

la Tabla 4.7. Se observa que la red entrenada con el dataset B tiene un mejor comportamiento en saturación, y en simulación se observa un control similar a la versión implícita.

Pruning

Dado que posteriormente las redes se describirán en hardware, un mayor número de parámetros resultará en un mayor uso de recursos físicos. Para mitigar este problema puede utilizarse *pruning*. La Figura 4.23 y muestran la distribución de la magnitud de los pesos para la red $L = 3, M = 9$ al utilizar regularización L1 y L2 respectivamente. Se observa que la regularización L1 resulta en una mayor cantidad de pesos que son exactamente o prácticamente cero, y también en algunos pesos de mayor mayor magnitud en comparación con la regularización L2. El *pruning* se efectúa manualmente a todos los pesos con magnitud menor a 0.001 se logra una *sparsity* de 80.09 % con regularización L1

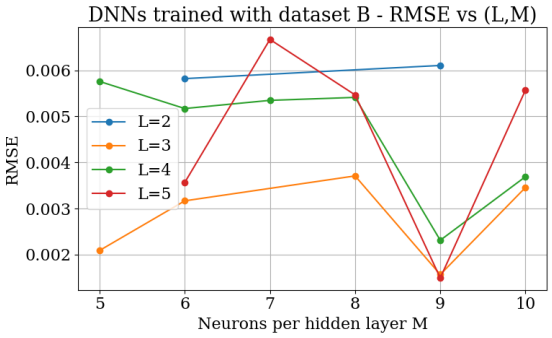


Fig. 4.20: RMSEs menores a 0.05 de redes entrenadas con el dataset B.

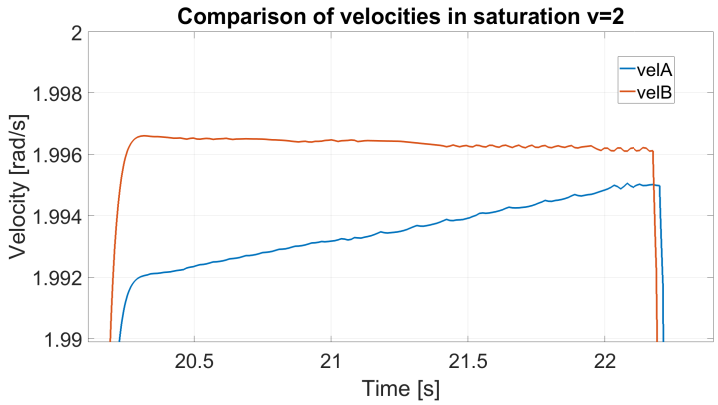


Fig. 4.21: Comparación de saturación de velocidad $v = 2$ para las DNNs entrenadas con los datasets A y B.

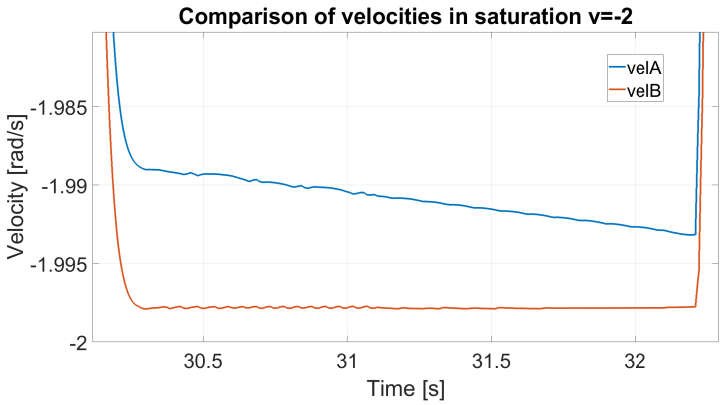


Fig. 4.22: Comparación de saturación de velocidad $v = -2$ para las DNNs entrenadas con los datasets A y B.

y de 67.26 % con regularización L2. La Figura 4.25 muestra una comparación del error cuadrático de la red con y sin *pruning* utilizando una *golden reference* aleatoria de 10 mil datos del dataset, observándose un error prácticamente igual y obteniendo un RMSE de 0.00312 y 0.00306 para la red con y sin *pruning* respectivamente.

Para las simulaciones y posteriores implementaciones se utilizarán las redes con *pruning* y con las primeras dos cuantizaciones utilizadas para MPC implícito, es decir $W = 32, Q = 8$ y $W = 21, Q = 7$. Las Figuras 4.26a y 4.26b muestran los resultados respectivamente, mostrando desempeños correctos y similares a MPC implícito. Se observa un cambio más abrupto en la actuación cuando hay cambio de referencia, en comparación con implícito.

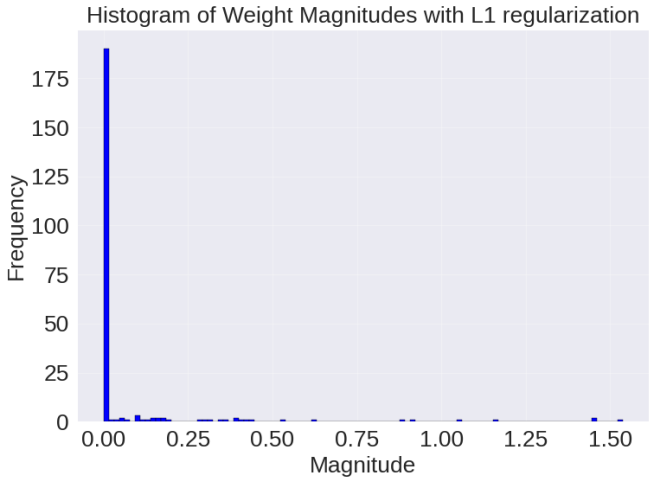


Fig. 4.23: Magnitud de pesos para entrenamiento con regularización L1.

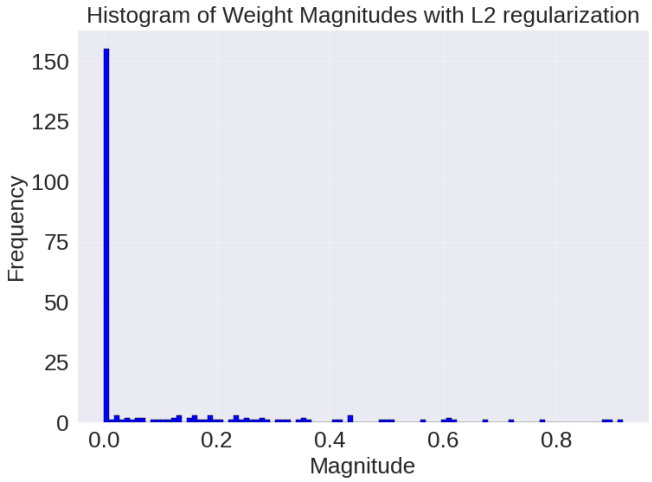


Fig. 4.24: Magnitud de pesos para entrenamiento con regularización L2.

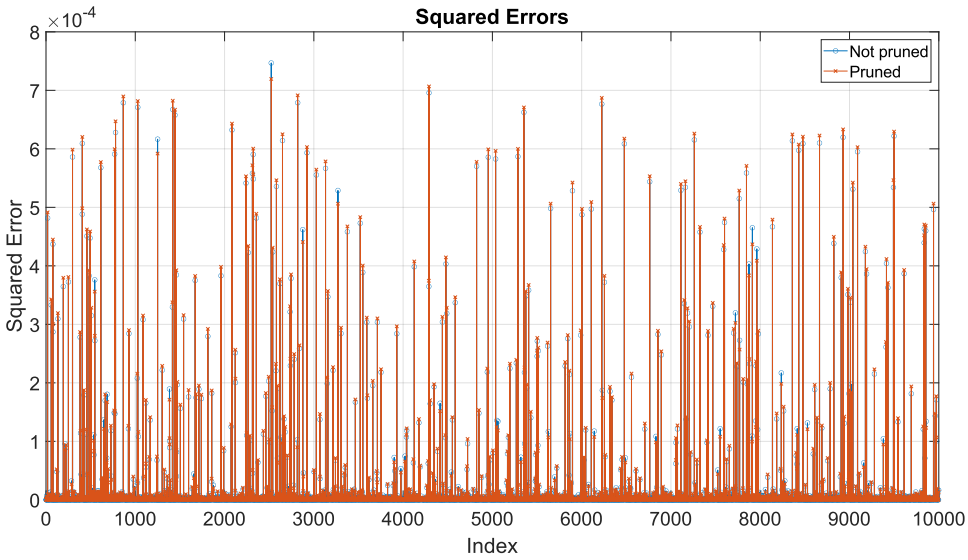
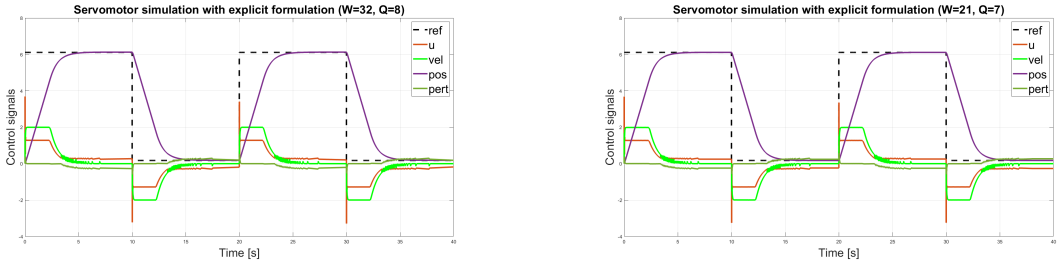


Fig. 4.25: Comparación de errores cuadráticos de red $L = 3, M = 9$ con y sin *pruning*.



(a) Simulación MIL de MPC explícito con redes para servomotor utilizando punto fijo ($W = 32$, $Q = 8$).

(b) Simulación MIL de MPC explícito con redes para servomotor utilizando punto fijo ($W = 21$, $Q = 7$).

Fig. 4.26: Comparación de simulaciones MIL de MPC explícito con redes para servomotor con diferentes configuraciones de punto fijo. Unidades: `ref` y `pos` en rad, `u` y `pert` en volts y `vel` y `vel_est` en rad/s.

4.3.2. Simulaciones Software-in-the-Loop para MPC explícito

Obtención de descripción de hardware

Para obtener la descripción de hardware en Verilog de las redes neuronales se utiliza HLS4ML, como se explicó en la Sección 3.2.2. Se utiliza la estrategia *Latency*, con un factor de reutilización unitario ya que en base a experimentos previos se estima que los recursos disponibles serán suficientes. Para las configuraciones de punto fijo de HLS4ML se utilizan las mismas que en Matlab para las simulaciones MIL, que corresponden a saturación en caso de *overflow* y opción de redondeo `Round`. La Tabla 4.8 muestra los recursos y latencias reportados a nivel de *HLS Synthesis* por HLS4ML, donde preliminarmente ya se aprecia una disminución drástica de los recursos físicos y de la latencia como consecuencia de realizar *pruning*. En comparación con las implementaciones implícitas, se observa una utilización menor de recursos y una reducción de la latencia en aproximadamente dos órdenes de magnitud.

Cosimulación

Para verificar que la descripción de hardware representa correctamente a la red neuronal, se efectúa una cosimulación utilizando Vitis Model Composer (como se explicó en la Sección 3.2.2), utilizando una *golden reference* de 10 mil datos. La Figura 4.27 muestra el diagrama de la cosimulación en Vitis Model Composer.

La Figura 4.28 muestra los resultados de la cosimulación de la red neuronal con precisión de punto fijo $W = 21, Q = 7$. En el gráfico se muestran las diferencias obtenidas entre la salida de la red y la salida de su descripción en Verilog para los datos de la *golden reference*. Se observa una diferencia de magnitud 2^{-14} para 14 puntos de los 10 mil puntos simulados. Esto corresponde a una diferencia en el bit menos significativo, ya que la representación elegida tiene 14 bits en su parte fraccionaria. Este error es esperable y no es significativo para la aplicación.

Tabla 4.8: Reporte de HLS4ML de recursos y latencias de controladores explícitos a nivel de *HLS Synthesis*.

W	Q	Pruned	LUTs	FFs	DSPs	Latencia [μs]*
32	8	No	89784	20393	322	0.25
32	8	Sí	14054	4827	64	0.19
21	7	Sí	8860	1773	21	0.17

*Con reloj de frecuencia 100[MHz].

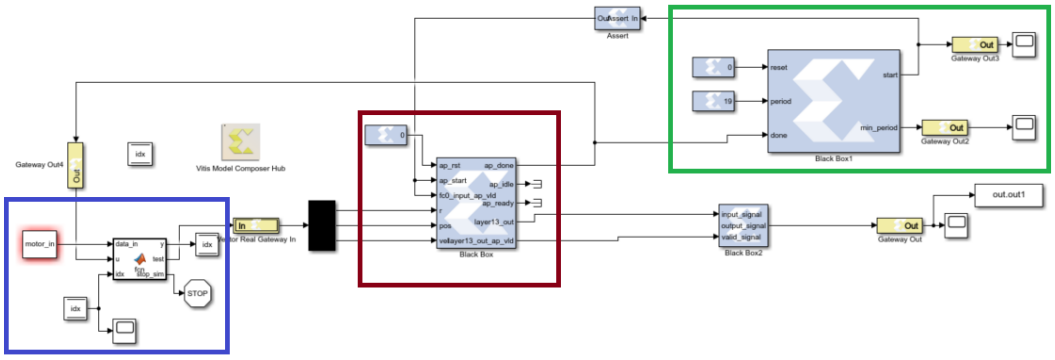


Fig. 4.27: Cosimulación en Vitis Model Composer. El cuadrado azul contiene la *golden reference*, el rojo contiene la red neuronal descrita en Verilog y el verde contiene un generador de pulsos periódicos.

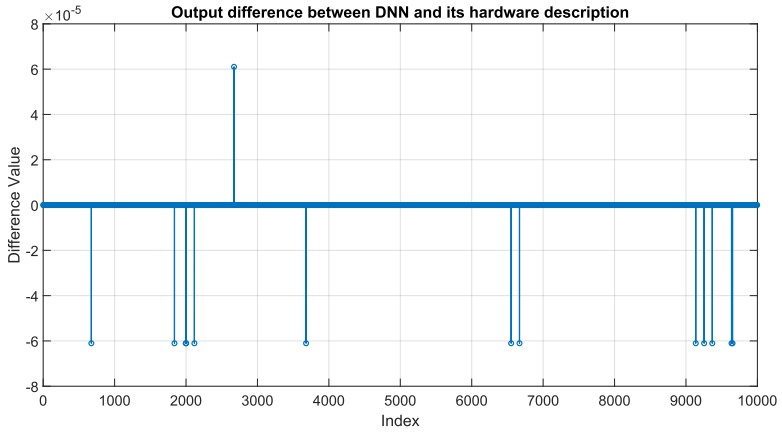


Fig. 4.28: Diferencia entre las salidas de la DNN y su descripción de hardware ($W = 21, Q = 7$).

4.3.3. Simulaciones Hardware-in-the-Loop para MPC explícito

Las simulaciones HIL para explícito (ver Sección 3.2.3 para los detalles de la definición del término HIL en este contexto) se realizan de manera similar a las implícitas, las cuales se mostraron en la Sección 4.2.3. La diferencia es que en la FPGA se reemplaza el controlador implícito por la red neuronal, además de necesitar normalización de las entradas y desnormalización de la salida. La Figura 4.29 muestra la implementación en FPGA.

En la implementación en FPGA hubo problemas de *timing*, lo que se solucionó con un *down-sampling* (factor 2) solamente de la red. Para cuantificar el ahorro de recursos al utilizar *pruning*, se realizó también la implementación de 32 bits sin *pruning*. La Tabla 4.9, en las columnas del *framework* `DS1202 FPGA I/O Type 1` (ver Sección 4.2.3 para una descripción de los *frameworks* disponibles), se muestran los recursos físicos utilizados a nivel de HIL. Al utilizar *pruning* (del 80.09 %) los recursos físicos disminuyen aproximadamente a la mitad y caen por debajo a la utilización de las implementaciones implícitas, que se muestran en la Tabla 4.4. Se observa además que el uso de LUTs y DSPs para la red sin *pruning* es menor que lo estimado a nivel de síntesis (ver Tabla 4.8).

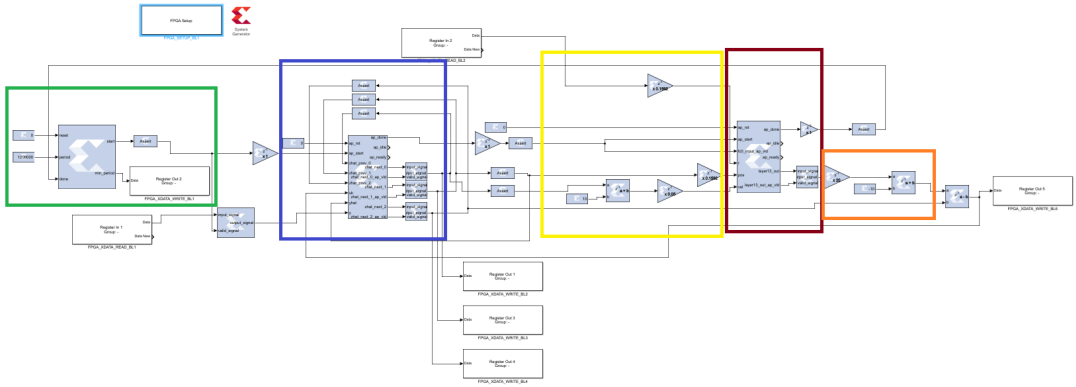
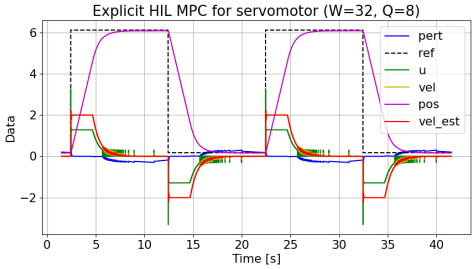


Fig. 4.29: Implementación explícita en FPGA. El cuadro rojo contiene el controlador (red neuronal), el azul el estimador de estados y el verde un módulo que genera pulsos periódicos cada 12[ms]. Además, el cuadro amarillo contiene la normalización de las entradas y el naranjo la desnormalización de la salida.

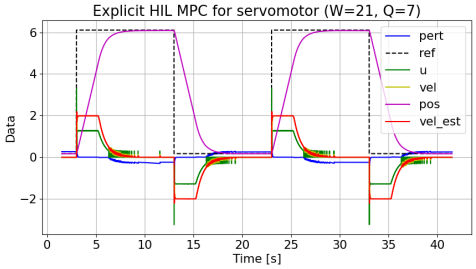
Tabla 4.9: Recursos utilizados para implementaciones explícitas para distintos niveles de cuantización y para distintos *frameworks* de la FPGA.

FW			`DS1202 FPGA I/O Type 1`			`DS1202 FPGA I/O Type 1 (Flexible I/O)`		
W	**Q**	**Pruned**	**LUTs**	**FFs**	**DSPs**	**LUTs**	**FFs**	**DSPs**
32	8	No	26025	20457	238	138773	115304	380
32	8	Sí	14426	11825	102	127134	106672	244
21	7	Sí	11423	10561	50	124119	105404	192
	Disp.		**203800**	**407600**	**840**	**203800**	**407600**	**840**

Las Figuras 4.30a y 4.30b muestran los resultados de las simulaciones HIL para $W = 32$ y $W = 21$ respectivamente. Se observa un control correcto, con un *overshoot* de la actuación mayor al observado en implícito, tal cómo se observó a nivel de MIL. A diferencia de las otras simulaciones, en estos gráficos la actuación que se muestra es la actuación efectiva después de la zona muerta.



(a) Simulación HIL de MPC explícito ($W = 32, Q = 8$).



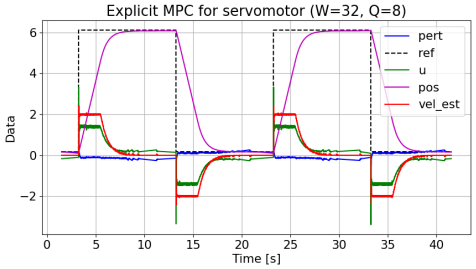
(b) Simulación HIL de MPC explícito ($W = 21, Q = 7$).

Fig. 4.30: Comparación de simulaciones HIL de MPC explícito con distintas configuraciones de punto fijo. Unidades: `ref` y `pos` en rad, `u` y `pert` en volts y `vel` y `vel_est` en rad/s.

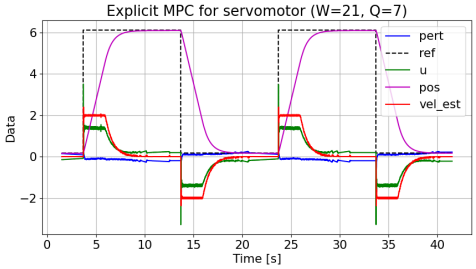
4.3.4. Implementaciones explícitas en FPGA

Las implementaciones explícitas se realizan de manera similar a las implícitas, las cuales se mostraron en la Sección 4.2.4. La diferencia es la implementación de la FPGA, que es como se explicó en la sección de simulaciones HIL explícitas, y se muestra en la Figura 4.29. El uso de recursos se muestra en la Tabla 4.9, en las columnas del *framework* `DS1202 FPGA I/O Type 1 (Flexible I/O)` (ver Sección 4.2.3 para una descripción de los *frameworks* disponibles), observándose un patrón igual al de las implementaciones HIL, pero con el uso base de recursos extra impuestos por el *framework* utilizado, necesario para usar el *encoder*.

Las Figuras 4.31a y 4.31b muestran las implementaciones para $W = 32$ y $W = 21$, y se observa un comportamiento consistente con lo observado a nivel de HIL.



(a) Implementación física de MPC explícito con FPGA ($W = 32, Q = 8$).



(b) Implementación física de MPC explícito con FPGA ($W = 21, Q = 7$).

Fig. 4.31: Comparación de implementaciones físicas de MPC explícito con FPGA para distintas configuraciones de punto fijo. Unidades: `ref` y `pos` en rad, `u` y `pert` en volts y `vel` y `vel_est` en rad/s.

4.4. Caracterización de latencia de los lazos de control

Para caracterizar los lazos de control deben considerarse por separado las tareas del procesador y de la FPGA, cuyas tares se muestran en la Figura 4.9. El procesador comienza con la medición o sensado (*input*), luego actualiza la referencia y finalmente actualiza el conversor digital-analógico con la actuación disponible (*output*). La FPGA realiza la estimación de estado y el cálculo de la actuación (MPC), y se consideran particularmente las implementaciones de 21 bits para el análisis temporal.

Para la medición de tiempos del procesador se utiliza la variable `turnaroundTime` (TAT) de ControlDesk, la cual indica el tiempo que demoran en ejecutarse todas las tareas asignadas en el ciclo de ejecución de la MicroLabBox. Se comienza por dejar solo los elementos esenciales del lazo de control, y luego hacer *builds* con cada tarea por separado, midiendo los tiempos de ejecución cada $1[ms]$, durante 1 minuto de ejecución, es decir 60 mil muestras de tiempo por tarea. Además, debe considerarse que la MicroLabBox realiza tareas base, además de las tareas consideradas. Para medir este tiempo base se removieron todos los elementos y se dejó el procesador sin tareas.

La Figura 4.32 muestra los tiempos de ejecución base medidos (TAT_{base}). Las figuras 4.33, 4.34 y 4.35 muestran los tiempos medidos de sensado (TAT_{input}), referencia (TAT_{ref}) y actuación (TAT_{output}) respectivamente. Además, la Figura 4.36 muestra los tiempos de ejecución del lazo completo (TAT_{loop}). Todos estos tiempos presentan un *jitter* asociado, inherente al procesador.

Para estimar el tiempo promedio de cada tarea, puede restarse el tiempo base promedio medido ($\overline{TAT}_{base}$) al promedio resultante al ejecutar solo la tarea de interés. Por ejemplo, el tiempo promedio de sensado puede estimarse como $\bar{t}_{input} = \overline{TAT}_{input} - \overline{TAT}_{base}$. Para el tiempo base se considera que $\bar{t}_{base} = \overline{TAT}_{base}$. La Figura 4.37 muestra los tiempos promedios estimados, los tiempos promedios medidos y los tiempos máximos medidos para cada tarea. Se observa que la diferencia entre los promedios medidos y los máximos medidos es aproximadamente igual para cada tarea, por lo que el *jitter* podría deberse principalmente a las tareas base.

Con estos tiempos puede calcularse el tiempo promedio estimado del lazo completo en el procesador como:

$$\bar{t}_{loop} = \bar{t}_{base} + \bar{t}_{input} + \bar{t}_{ref} + \bar{t}_{output} \tag{4.4.1}$$

La Figura 4.38 muestra el tiempo máximo del lazo medido del procesador, el tiempo promedio estimado ($\bar{t}_{loop}$) y el tiempo promedio medido ($\overline{TAT}_{loop}$), junto con los tiempos de ejecución de la FPGA para las implementaciones implícitas y explícitas de 21 bits. Se observa que el tiempo de lazo estimado usando las tareas por separado presenta una leve sobreestimación con respecto al medido. Los tiempos de la FPGA no presentan *jitter* al ser ejecutados en un número fijo de ciclos de reloj.

Dado que el procesador y la FPGA operan en paralelo, si se quisiera acelerar el lazo de control tendría que acortarse el tiempo de ejecución del dispositivo que presente una mayor latencia en sus tareas, es decir, el cuello de botella. Para la implementación implícita el cuello de botella es el cálculo de la FPGA, pero para la explícita son las tareas del procesador.

La Figura 4.39 muestra los tiempos promedios de todas las tareas, tanto para el procesador como para la FPGA. Notar que la latencia del cálculo de MPC para implícito es un poco menor a la estimada a nivel de *HLS Synthesis*, que se muestra en la Tabla 4.3, y para explícito es aproximadamente el doble que lo estimado en *HLS Synthesis* (como muestra la Tabla 4.8), dado que en la implementación explícita fue necesario realizar un *downsampling*. La latencia del estimador resultó ser menor a la estimada a nivel de *HLS Synthesis*, la cual se muestra en la Tabla 4.2.

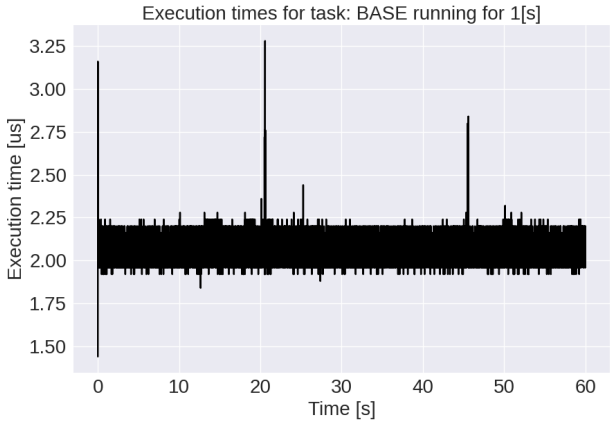


Fig. 4.32: Tiempo de ejecución base, considerando 1000 muestras por segundo.

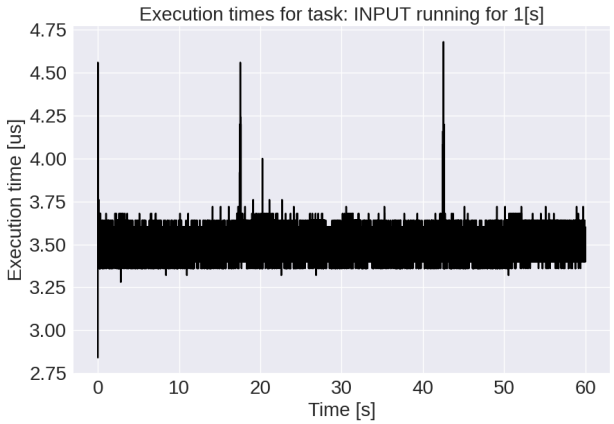


Fig. 4.33: Tiempo de ejecución de la medición (*input*), considerando 1000 muestras por segundo.

Fig. 4.34: Tiempo de ejecución de la referencia, considerando 1000 muestras por segundo.

Fig. 4.35: Tiempo de ejecución de la actuación (*output*), considerando 1000 muestras por segundo.

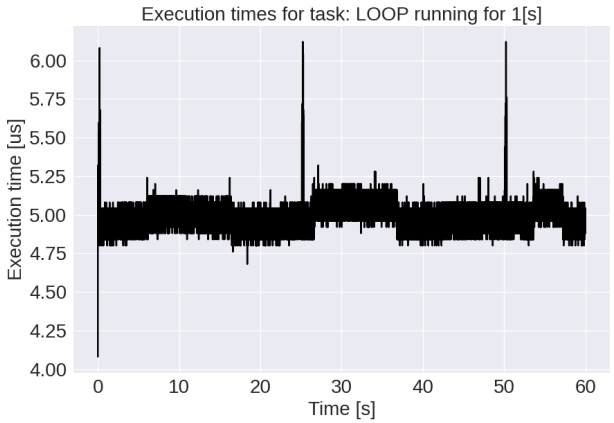


Fig. 4.36: Tiempo de ejecución del lazo de control completo, considerando 1000 muestras por segundo.

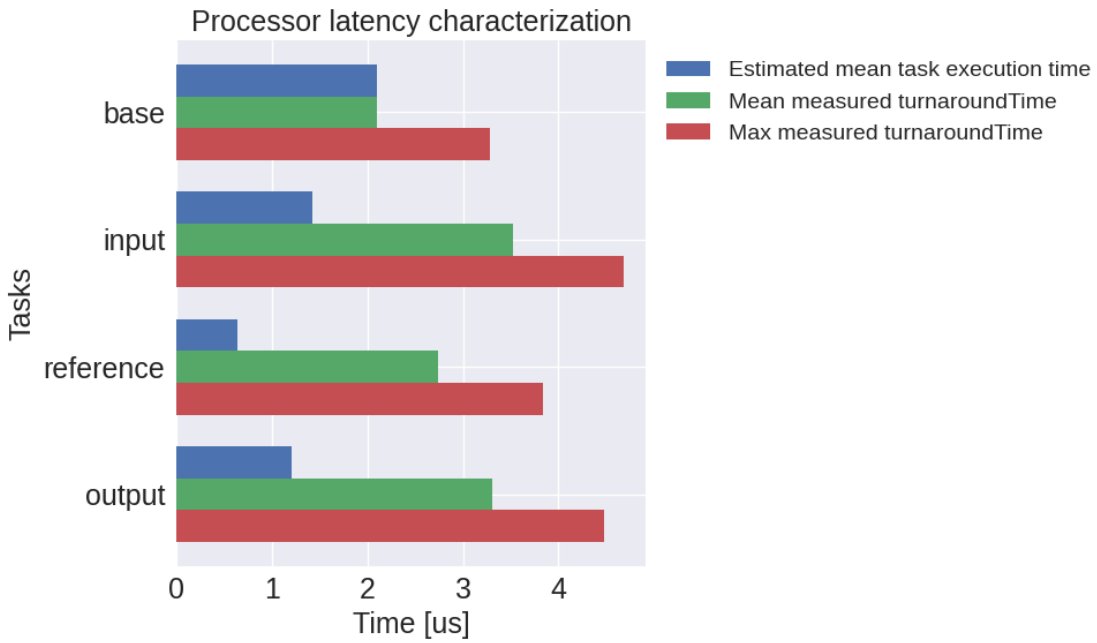


Fig. 4.37: Tiempos promedios estimados, tiempos promedios medidos y tiempos máximos medidos para cada tarea del procesador.

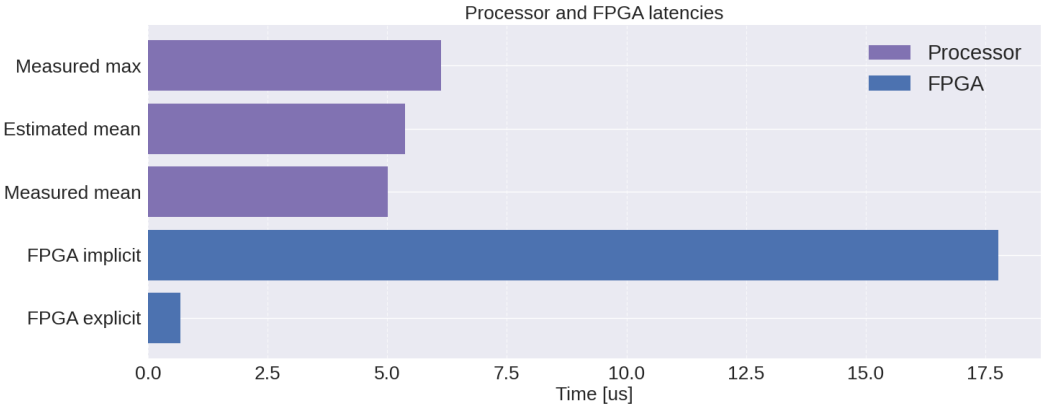


Fig. 4.38: Latencias del procesador y de la FPGA ($W = 21$).

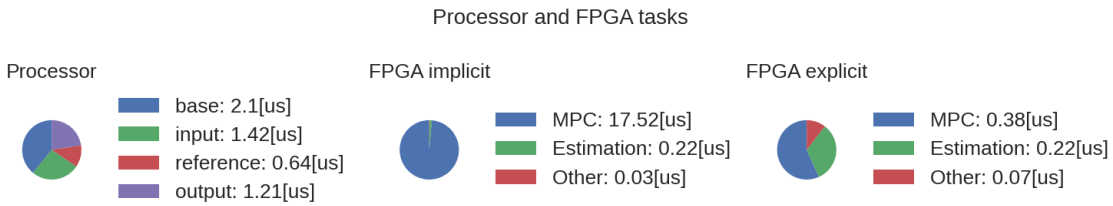


Fig. 4.39: Tiempos de todas las tareas del procesador y de la FPGA ($W = 21$).

Capítulo 5

APLICACIÓN DEL FLUJO DE DISEÑO A UNIDAD DER

En este capítulo se aplica el flujo de diseño propuesto en el Capítulo 3 para lazos MPC explícitos con redes neuronales en FPGA para controlar una unidad de Recursos Energéticos Distribuidos (DER, por sus siglas en inglés). Las implementaciones realizadas en este capítulo están basadas en un trabajo previo [46] en el cual se integran lazos MPC explícitos con una MicroLabBox, utilizando redes neuronales en procesador. En las implementaciones de esta tesis solo se reemplazan las redes neuronales y se implementan en FPGA, por lo que para más detalles véase el trabajo original [46].

Los DER son sistemas de generación o almacenamiento de energía eléctrica a pequeña o mediana escala, que se ubican cerca de los puntos de consumo y operan de manera descentralizada respecto a las plantas de generación tradicionales. Dentro de esta categoría se incluyen tecnologías como paneles fotovoltaicos, turbinas eólicas de pequeña escala, microturbinas, generadores a diésel o gas, y sistemas de almacenamiento como baterías. A diferencia de la generación centralizada, los DER permiten gestionar la energía de manera más flexible y localizada, adaptándose mejor a las necesidades específicas de comunidades, industrias o microrredes.

Las aplicaciones de los DER abarcan un espectro amplio. En modo *grid-forming*, pueden actuar como fuentes de tensión independientes que abastecen directamente cargas locales. En *grid-feeding*, se conectan a la red inyectando o absorbiendo potencia activa, aunque sin aportar a la estabilidad global del sistema. Finalmente, en el modo *grid-supporting*, contribuyen a la regulación de voltaje y frecuencia, fortaleciendo la estabilidad de la red mediante el suministro coordinado de potencia activa y reactiva [47–51]. Este último rol es especialmente relevante en la integración de energías renovables, ya que permite compensar su variabilidad y mejorar la confiabilidad del sistema eléctrico.

La importancia de los DER radica en su capacidad de aumentar la resiliencia, eficiencia y sostenibilidad de las redes eléctricas modernas. Su integración posibilita la creación de *microgrids*, que pueden operar conectadas o de manera aislada respecto a la red principal, garantizando continuidad de suministro incluso frente a fallas. Además, permiten reducir pérdidas por transmisión, aprovechar fuentes renovables cercanas al consumo, y responder de manera más dinámica a fluctuaciones de la demanda. En el contexto de la transición energética, los DER constituyen una pieza clave para avanzar hacia sistemas eléctricos más limpios, descentralizados y flexibles [52, 53].

En particular, los DER con los que se trabaja en esta tesis están implementados mediante convertidores electrónicos de potencia y filtros LCL, lo que plantea desafíos de control asociados a resonancias y restricciones operacionales [48, 54]. Para abordar estos desafíos, se emplea un esquema jerárquico de control MPC en dos niveles: el nivel cero, encargado de regular la tensión de salida del DER, y el nivel primario, orientado a equilibrar la generación con la demanda, garantizando la

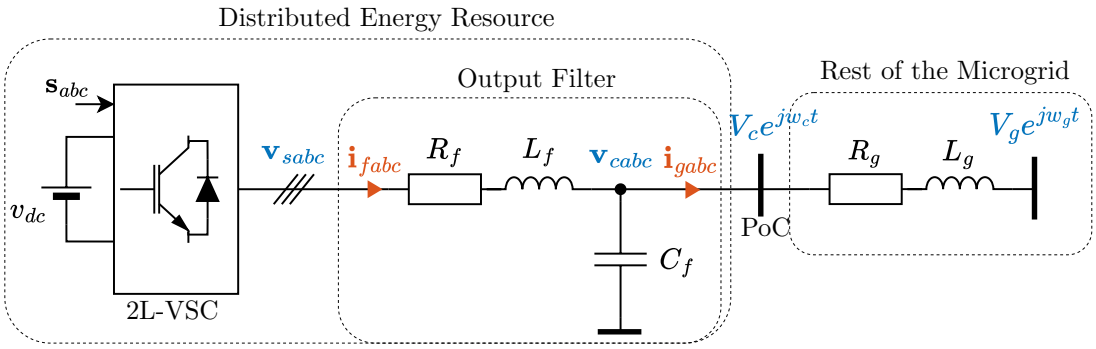


Fig. 5.1: Diagrama de una unidad DER conectada a una microrred [46].

estabilidad de voltaje y frecuencia en la microrred.

5.1. Planteamiento dinámico y modelos linealizados del DER

La Figura 5.1 ilustra el modelo de interacción entre una unidad DER y la microrred, que se basa en un convertidor de dos niveles (2L-VSC), un filtro de salida tipo LCL y la red representada como una fuente de tensión con impedancia en serie. La Tabla 5.1 muestra los valores de los parámetros del circuito del DER y los valores nominales de interés.

Parámetros nominales

Tras aplicar las transformaciones de Clarke y Park (ver [46], Cap. 5) y expresando las variables en el marco rotante dq (alineado con la tensión en el PoC), el modelo de estado continuo queda:

Tabla 5.1: Parámetros del circuito del DER y valores nominales

Parámetro	Valor	Observación	Parámetro	Valor	Observación
R_f	$0.035\,\Omega$	Resistencia del filtro	R_g	$0.84\,\Omega$	Resistencia de cortocircuito
L_f	5 mH	Inductancia del filtro	L_g	3.3 mH	Inductancia de cortocircuito
C_f	$12\,\mu$F	Capacitancia del filtro	f_{sw}	5 kHz	Frecuencia de conmutación (portadora)
v_{dc}	300 V	Tensión del enlace DC	T_s	$200\,\mu$s	Período de muestreo
V_g	150 V	Tensión de red	$\{\omega_b, \omega_g\}$	$50 \cdot 2\pi$ rad/s	Frecuencia nominal y de red
v_b	173.20 V	Tensión nominal	i_b	6 A	Corriente nominal
$\check{h}_{vc}$	100 V	Tensión mínima	$\hat{h}_{vc}$	200 V	Tensión máxima
$\check{h}_{\omega_c}$	$49 \cdot 2\pi$ rad/s	Frecuencia mínima	$\hat{h}_{\omega_c}$	$51 \cdot 2\pi$ rad/s	Frecuencia máxima

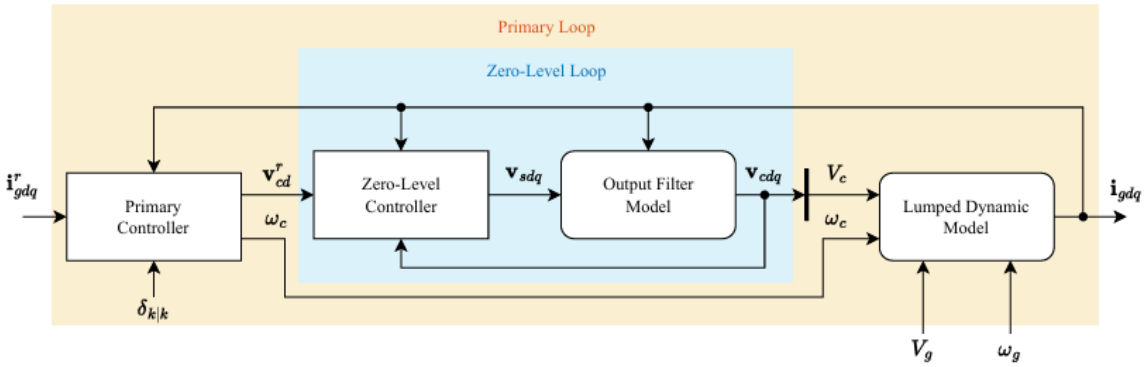


Fig. 5.2: Modelo en cascada para control en nivel cero y primario [46].

$$\dot{i}_{fdq} = -\frac{R_f}{L_f} i_{fdq} - j\omega_c i_{fdq} - \frac{1}{L_f} v_{cdq} + \frac{1}{L_f} v_{sdq}, \tag{5.1.1}$$

$$\dot{v}_{cdq} = -\frac{1}{C_f} i_{gdq} - j\omega_c v_{cdq} + \frac{1}{C_f} i_{fdq}, \tag{5.1.2}$$

$$\dot{i}_{gdq} = -\frac{R_g}{L_g} i_{gdq} - j\omega_c i_{gdq} + \frac{1}{L_g} v_{cdq} - \frac{1}{L_g} v_{gdq} e^{j\delta}, \tag{5.1.3}$$

$$\dot{\delta} = \omega_g - \omega_c. \tag{5.1.4}$$

Se linealiza alrededor del punto de operación (PO) definido por $\bar{i}_{fdq} = \mathbf{0}$, $\bar{i}_{gdq} = \mathbf{0}$, $\bar{v}_{cdq} = \bar{V}_c = v_b$, $\bar{\omega}_c = \omega_b$, $\bar{\delta} = 0$ (condición intermedia generación/carga), y se definen las variables de pequeña señal como $\tilde{x} = x - \bar{x}$.

La Figura 5.2 muestra a grandes rasgos cómo se interconectan en cascada los controladores en nivel cero y primario con los elementos de la planta.

Modelo del filtro de salida (nivel cero)

El modelo dinámico continuo del filtro en la salida del inversor es descrito por las ecuaciones (5.1.1) y (5.1.2), y se expresa en forma linealizada (con variables de pequeña señal) como:

$$\dot{\tilde{x}}_z = \mathbf{A}_z \tilde{x}_z + \mathbf{B}_z \tilde{u}_z + \mathbf{P}_z \tilde{d}^m_z, \tag{5.1.5}$$

$$\tilde{y}_z = \mathbf{C}_z \tilde{x}_z, \tag{5.1.6}$$

donde se consideran los siguientes vectores:

- estados $x_z = [\, i_{fd},\, i_{fq},\, v_{cd},\, v_{cq} \,]^{\mathrm{T}} \in \mathbb{R}^4$: corrientes del filtro y voltajes del capacitor del filtro,
- entradas $u_z = [\, v_{sd},\, v_{sq} \,]^{\mathrm{T}} \in \mathbb{R}^2$: voltajes aplicados por el 2L-VCS,
- salidas $y_z = [\, v_{cd},\, v_{cq} \,]^{\mathrm{T}} \in \mathbb{R}^2$: voltajes del capacitor del filtro que deben ser regulados,
- perturbaciones medibles $d^m_z = [\, i_{gd},\, i_{gq} \,]^{\mathrm{T}} \in \mathbb{R}^2$: corrientes de la red,

y las matrices:

$$\mathbf{A}_z = \begin{bmatrix} -\dfrac{R_f}{L_f} & \bar{\omega}_c & -\dfrac{1}{L_f} & 0 \\ -\bar{\omega}_c & -\dfrac{R_f}{L_f} & 0 & -\dfrac{1}{L_f} \\ \dfrac{1}{C_f} & 0 & 0 & \bar{\omega}_c \\ 0 & \dfrac{1}{C_f} & -\bar{\omega}_c & 0 \end{bmatrix}, \quad \mathbf{B}_z = \begin{bmatrix} \dfrac{1}{L_f} & 0 \\ 0 & \dfrac{1}{L_f} \\ 0 & 0 \\ 0 & 0 \end{bmatrix},$$

$$\mathbf{P}_z = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ -\dfrac{1}{C_f} & 0 \\ 0 & -\dfrac{1}{C_f} \end{bmatrix}, \qquad \mathbf{C}_z = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Modelo de acoplamiento con la red (nivel primario)

El modelo dinámico continuo de la interacción con la red es descrito por las ecuaciones (5.1.3) y (5.1.4), y se expresa en forma linealizada (con variables de pequeña señal) como:

$$\dot{\tilde{x}}_p = \mathbf{A}_p\, \tilde{x}_p + \mathbf{B}_p\, \tilde{u}_p + \mathbf{P}_p\, \tilde{d}_z^m, \tag{5.1.7}$$

$$\tilde{y}_p = \mathbf{C}_p\, \tilde{x}_p, \tag{5.1.8}$$

donde se consideran los siguientes vectores:

- estados $x_p = [\, i_{gd},\, i_{gq},\, \delta \,]^{\mathrm{T}} \in \mathbb{R}^3$: corrientes de la red y el ángulo de fase,
- entradas $u_p = [\, V_c,\, \omega_c \,]^{\mathrm{T}} \in \mathbb{R}^2$: amplitud y frecuencia del voltaje en la salida del filtro,
- salidas $y_p = [\, i_{gd},\, i_{gq} \,]^{\mathrm{T}} \in \mathbb{R}^2$: corrientes de la red que deben ser reguladas,
- perturbaciones medibles $d_p^m = [\, V_g,\, \omega_g \,]^{\mathrm{T}} \in \mathbb{R}^2$: amplitud y frecuencia del voltaje de la micro-rred,

y las siguientes matrices:

$$\mathbf{A}_p = \begin{bmatrix} -\dfrac{R_g}{L_g} & \bar{\omega}_c & \dfrac{V_g}{L_g}\sin\bar{\delta} \\ -\bar{\omega}_c & -\dfrac{R_g}{L_g} & -\dfrac{V_g}{L_g}\cos\bar{\delta} \\ 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{B}_p = \begin{bmatrix} \dfrac{1}{L_g} & \bar{i}_{gq} \\ 0 & -\bar{i}_{gd} \\ 0 & -1 \end{bmatrix}, \qquad \mathbf{P}_p = \begin{bmatrix} -\dfrac{1}{L_g}\cos\bar{\delta} & 0 \\ -\dfrac{1}{L_g}\sin\bar{\delta} & 0 \\ 0 & 1 \end{bmatrix},$$

$$\mathbf{C}_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$

las cuales se evalúan en el PO indicado ($\bar{i}_{gd} = \bar{i}_{gq} = 0,\ \bar{\delta} = 0$) y se obtienen las matrices simplificadas:

$$\mathbf{A}_p\big|_{\mathrm{PO}} = \begin{bmatrix} -\dfrac{R_g}{L_g} & \bar{\omega}_c & 0 \\ -\bar{\omega}_c & -\dfrac{R_g}{L_g} & -\dfrac{V_g}{L_g} \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{B}_p\big|_{\mathrm{PO}} = \begin{bmatrix} \dfrac{1}{L_g} & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix}, \quad \mathbf{P}_p\big|_{\mathrm{PO}} = \begin{bmatrix} -\dfrac{1}{L_g} & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

Relación discreta (ZOH)

Para implementación a tiempo discreto con periodo T_s:

$$\mathbf{A}_{\ell d} = e^{\mathbf{A}_\ell T_s}, \qquad \begin{bmatrix} \mathbf{B}_{\ell d} & \mathbf{P}_{\ell d} \end{bmatrix} = \int_0^{T_s} e^{\mathbf{A}_\ell \tau} \begin{bmatrix} \mathbf{B}_\ell & \mathbf{P}_\ell \end{bmatrix} d\tau,$$

con $\ell \in \{z, p\}$.

Las matrices mostradas en las secciones anteriores son las linealizadas continua, pero para la implementación de MPC se usan las formas discretas $A_{\ell d}, B_{\ell d}, P_{\ell d}$. Luego, el sistema dinámico discreto queda modelado por las ecuaciones:

$$x_{\ell,k+1} = \mathbf{A}_{\ell d} x_{\ell,k} + \mathbf{B}_{\ell d}\left(u_{\ell,k} + d^u_{\ell,k}\right) + \mathbf{P}_{\ell d} d^m_{\ell,k}, \tag{5.1.9}$$

$$y_{\ell,k} = \mathbf{C}_\ell x_{\ell,k}, \tag{5.1.10}$$

donde $d^u_{\ell,k}$ es el vector de perturbaciones no medibles de entradas.

Restricciones operativas

Restricción suave de corriente del filtro (aproximación poligonal de la circunferencia) Por seguridad se impone la restricción suave $\|i_{fdq}\|_2 \leq i_b$, la cual se aproxima por un dodecágono (ver Figura 5.3):

$$\mathbf{H}_{if} \begin{bmatrix} i_{fd} \\ i_{fq} \end{bmatrix} \leq h_{if},$$

donde (ver [46]):

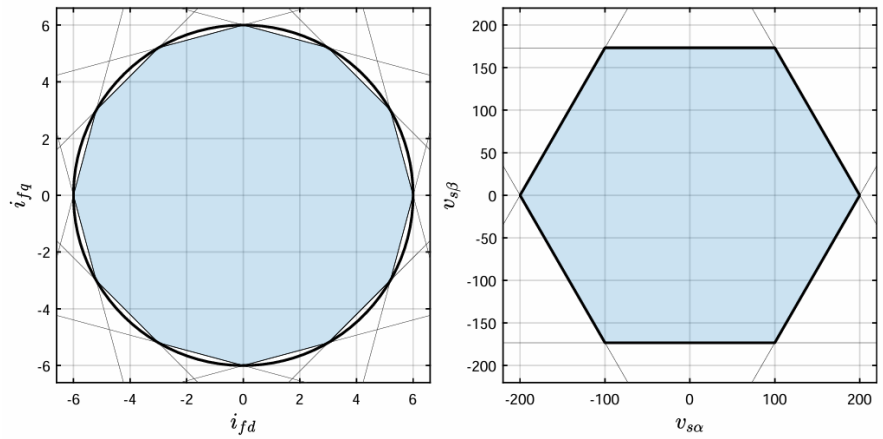


Fig. 5.3: Restricciones politópicas de i_{fdq} y $v_{s\alpha\beta}$ [46].

$$\mathbf{H}_{if} = \begin{bmatrix} 3.7321 & 1 \\ 1 & 1 \\ 0.2679 & 1 \\ -0.2679 & 1 \\ -1 & 1 \\ -3.7321 & 1 \\ -3.7321 & -1 \\ -1 & -1 \\ -0.2679 & -1 \\ 0.2679 & -1 \\ 1 & -1 \\ 3.7321 & -1 \end{bmatrix}, \qquad h_{if} = i_b \begin{bmatrix} 3.7321 \\ 1.3660 \\ 1 \\ 1 \\ 1.3660 \\ 3.7321 \\ 3.7321 \\ 1.3660 \\ 1 \\ 1 \\ 1.3660 \\ 3.7321 \end{bmatrix}.$$

Restricción dura de tensión del inversor (hexágono en $\alpha\beta$) El espacio de tensiones del inversor está limitado por un hexágono (ver Figura 5.3):

$$\mathbf{H}_{vs}\, \mathbf{T}_{dq}^{\alpha\beta}(\theta_c) \begin{bmatrix} v_{sd} \\ v_{sq} \end{bmatrix} \leq h_{vs},$$

donde (ver [46]):

$$\mathbf{H}_{vs} = \begin{bmatrix} 1.7321 & 1 \\ 0 & 1 \\ -1.7321 & 1 \\ -1.7321 & -1 \\ 0 & -1 \\ 1.7321 & -1 \end{bmatrix}, \qquad h_{vs} = \frac{2}{3} v_{dc} \begin{bmatrix} 1.7321 \\ 0.8660 \\ 1.7321 \\ 1.7321 \\ 0.8660 \\ 1.7321 \end{bmatrix},$$

y $\mathbf{T}_{dq}^{\alpha\beta}(\theta_c)$ es la matriz de rotación que transforma de coordenadas dq a $\alpha\beta$ (dependencia rotacional con θ_c).

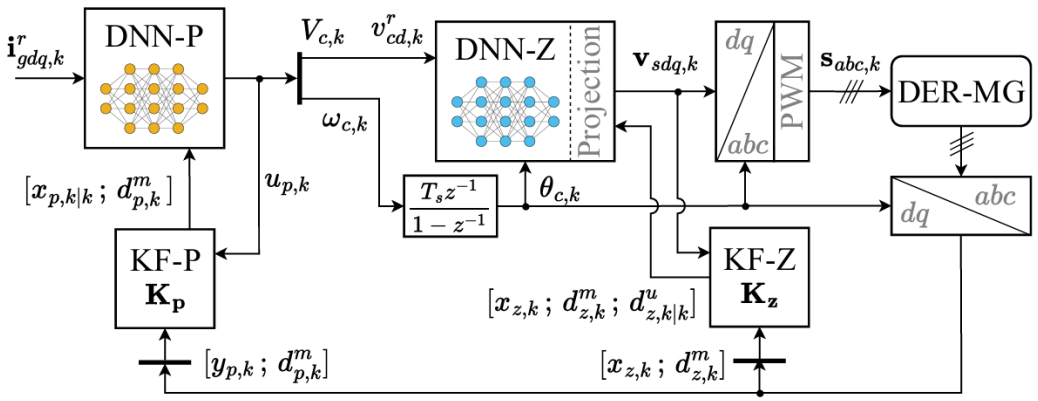


Fig. 5.4: Diagrama de bloques con DNNs y filtros de Kalman [46].

Restricción suave de amplitud y frecuencia del PoC (caja) Para la amplitud V_c y la frecuencia ω_c se usan restricciones tipo caja:

$$\mathbf{H}_{v\omega_c}\begin{bmatrix}V_c\\ \omega_c\end{bmatrix} \le h_{v\omega_c},$$

con

$$\mathbf{H}_{v\omega_c} = \begin{bmatrix}1 & 0\\ -1 & 0\\ 0 & 1\\ 0 & -1\end{bmatrix}, \qquad h_{v\omega_c} = \begin{bmatrix}\hat{h}_{vc}\\ -\check{h}_{vc}\\ \hat{h}_{\omega_c}\\ -\check{h}_{\omega_c}\end{bmatrix},$$

donde típicamente $\check{h}_{vc} = 100$ V, $\hat{h}_{vc} = 200$ V, $\check{h}_{\omega_c} = 49 \cdot 2\pi$, $\hat{h}_{\omega_c} = 51 \cdot 2\pi$.

Proyección de salida de DNN y filtros de Kalman

La Figura 5.4 muestra el diagrama de bloques del lazo completo, donde se consideran DNNs en cascada para aproximar las leyes de control en nivel cero y primaria. Dado que al utilizar redes neuronales no se garantiza el cumplimiento de restricciones, se utiliza una proyección en la salida de la red de control en nivel cero (DNN-Z) que garantice que la siguiente salida mantendrá a los estados en sus respectivos espacios factibles. Además se utilizan filtros de Kalman estacionarios para lidiar con las perturbaciones, como se propuso en la implementación original. Para los detalles de los filtros de Kalman y la formulación del problema de optimización asociado a la proyección ver [46], Cap. 7.

5.2. Simulaciones Model-in-the-Loop

Entradas y salidas de las DNNs

Utilizando la estructura explicada en la Sección 2.3.2 para las DNNs, se consideran como entradas los vectores $x_z \in \mathbb{R}^9$ y $x_p \in \mathbb{R}^7$ para el control en nivel cero y primario respectivamente, donde:

$$x_{r,z} = \left[x_z, d_z^m, v_{cd}^r, \cos\left(\theta_c\right), \sin\left(\theta_c\right)\right]^{\mathrm{T}}, \quad x_{r,p} = \left[x_p, d_p^m, i_{gdq}^r\right]^{\mathrm{T}}, \tag{5.2.1}$$

donde v_{cd}^r es la referencia paras el voltaje v_{cd} (la referencia v_{cq}^r no se considera en el vector ya que se fija en cero), e i_{gdq}^r es la referencia para las salidas del control primario, que corresponden a las corrientes de la red.

Como salidas se consideran las estimaciones de las actuaciones óptimas:

$$\hat{u}_z = \left[\hat{v}_{sd}, \hat{v}_{sq}\right]^{\mathrm{T}}, \quad \hat{u}_p = \left[\hat{V}_c, \hat{\omega}_c\right]^{\mathrm{T}}. \tag{5.2.2}$$

Conjunto de datos

Para la generación de datos, se resuelve la fomulación implícita de MPC en lazo cerrado para simulaciones de 0.4[s] utilizando Simulink y Simscape Electrical. se utiliza el solver *quadprog* con el algoritmo *active-set* en Matlab considerando un horizonte de $N = 3$. Se realizaron 4000 simulaciones en lazo cerrado, alternando entre episodios de control en nivel cero y primario. Dado que se utilizan 200[μs] entre cada instante de tiempo, en total hay 2000 puntos de datos por simulación. Esto resulta en un total de 8 millones de datos para cada conjunto de datos de datos (para DNN-Z y DNN-P). Estos datos se separan en un 80 % de entrenamiento, 10 % para validación y 10 % para test.

Para obtener conjuntos de datos representativos de la interacción DER-red y sus leyes de control implícitas, se consideraron las siguientes variaciones de parámetros (ver tabla 5.1):

- Los parámetros R_g y L_g se hacen variar en un rango de $\pm 15\,\%$ de sus valores nominales.
- Los valores de referencias se seleccionan aleatoriamente en los rangos $v_{cd}^r \in \left[\check{h}_{vc}, \hat{h}_{vc}\right]$, y ambas componentes de i_{gdq}^r se generan aleatoriamente en el rango $[0, 1.5i_b]$. Además, para todas las referencias se aplican cambios de tipo escalón en instantes aleatorios dentro de una misma simulación.
- Las condiciones iniciales $[x_{z,0}, x_{p,0}]^{\mathrm{T}}$ se seleccionan aleatoriamente calculando soluciones en estado estacionario factibles utilizando las ecuaciones (5.1.1), (5.1.2), (5.1.3) y (5.1.4).

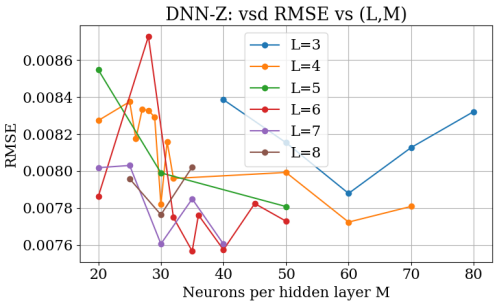
Hiperparámetros

Para los entrenamientos de las redes DNN-Z y DNN-P se consideran los mismos hiperparámetros de entrenamiento, los cuales se muestran en la Tabla 5.2. En las exploraciones no se lograron resultados favorables al utilizar regularización L1, por lo que se optó por utilizar regularización L2.

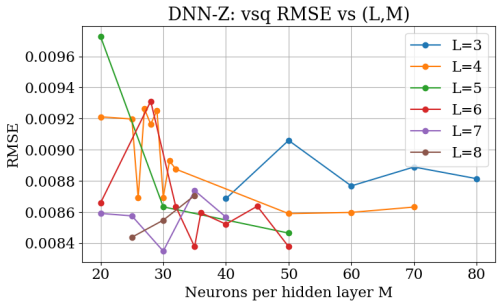
Para la exploración del número de capas ocultas L y el número de neuronas M por capa oculta, se realiza una exploración probando diversas configuraciones. Las Figuras 5.5 muestran las redes neuronales DNN-Z entrenadas con RMSEs menor a 0.05 que se obtuvieron en la exploración. De igual manera,5.5 muestra las redes DNN-P entrenadas y sus RMSEs. El barrido de parámetros se

Tabla 5.2: Hiperparámetros utilizados

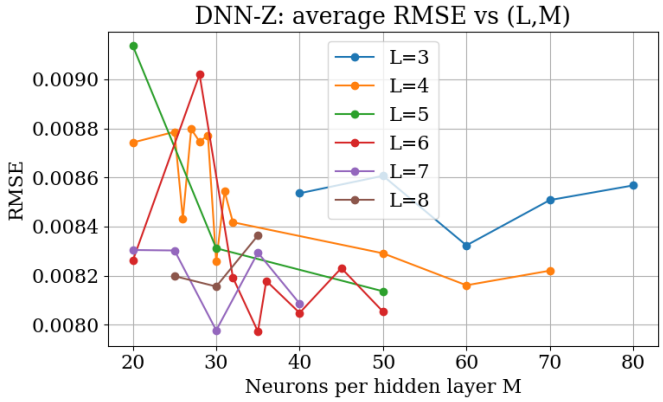
Hiperparámetro	**Flotante**	**Punto fijo**
Optimizador	Adam	Adam
Learning rate	0.001	0.00075
Batch size	32	32
Regularización	L2, ganancia 10^{-9}	L2, ganancia 10^{-9}
Épocas	paciencia 10	paciencia 3



(a) Primera salida de DNN-Z.



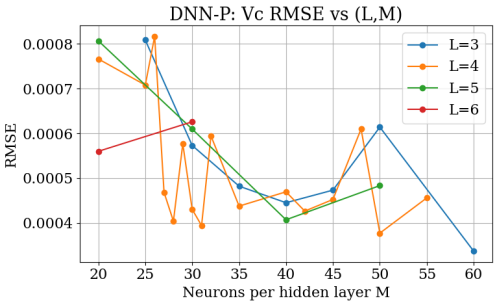
(b) Segunda salida de DNN-Z.



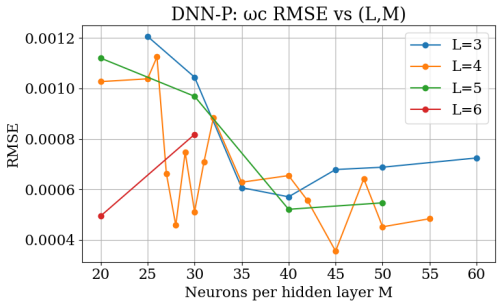
(c) Promedio de RMSEs de las salidas de DNNZ.

Fig. 5.5: RMSEs menores a 0.05 para las salidas de DNN-Z.

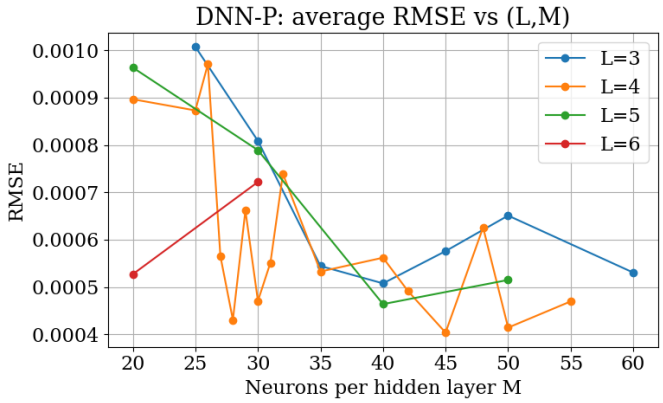
hizo de tal manera que $L \in \{3, 4, 5, 6, 7, 8\}$, y los valores de M se fueron eligiendo a medida que se observaban los RMSEs obtenidos. Considerando como criterios el RMSE, el número de parámetros y el desempeño en simulación, se eligieron los parámetros $L = 4$ y $M = 30$, tanto para DNN-Z como para DNN-P.



(a) Primera salida de DNN-P.



(b) Segunda salida de DNN-P.



(c) Promedio de RMSEs de las salidas de DNN-P.

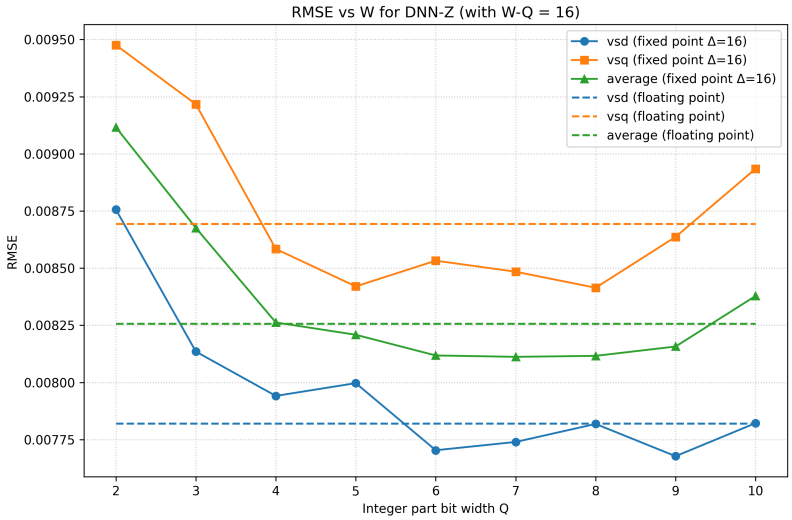
Fig. 5.6: RMSEs menores a 0.05 para las salidas de DNN-P.

Exploración de niveles de cuantización

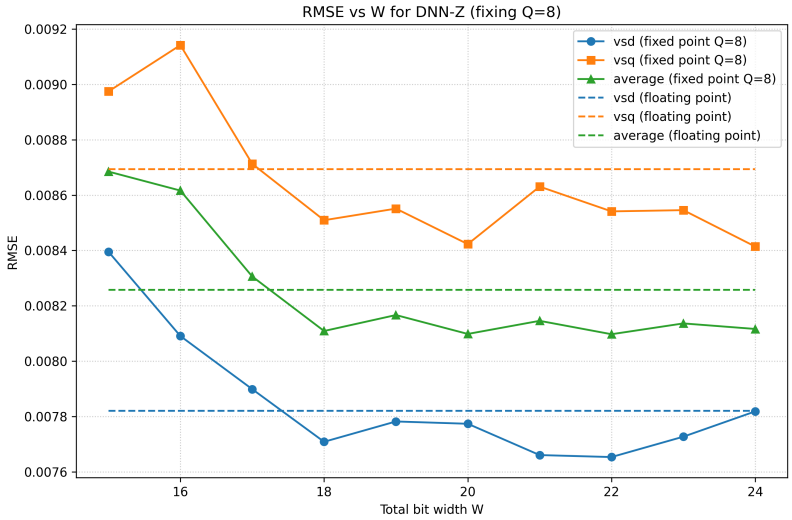
Ya obtenidas DNN-Z y DNN-P en punto flotante, se procede a elegir representaciones en punto fijo adecuadas para cada red. Esto consiste en elegir el número total de bits W y el número de bits para la parte entera con signo Q. Para obtener el valor de Q, se hace un barrido manteniendo un valor fijo de 16 bits para la parte fraccionaria. Una vez elegido el valor de Q, este se mantiene fijo y se hace un barrido de W.

Para DNN-Z, la Figura 5.7a muestra los RMSEs obtenidos al hacer el barrido de Q, y considerando tanto el RMSE como el desempeño en simulación, se elige $Q = 8$. La Figura 5.7b muestra los RMSEs obtenidos el barrido para W (manteniendo $Q = 8$ fijo), eligiendo $W = 22$.

Para DNN-P, la Figura 5.8a muestra los RMSEs obtenidos al hacer el barrido de Q, y considerando tanto el RMSE como el desempeño en simulación, se elige $Q = 4$. La Figura 5.8b muestra los RMSEs obtenidos el barrido para W (manteniendo $Q = 4$ fijo), eligiendo $W = 15$.

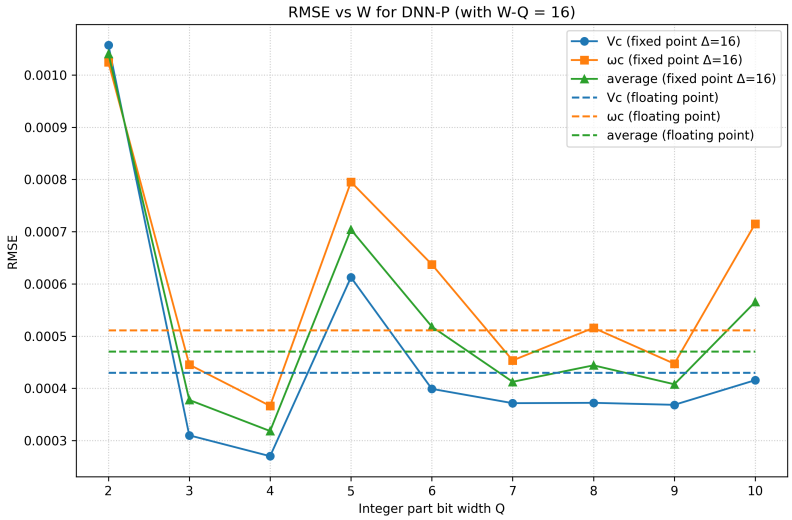


(a) RMSEs de las salidas de DNN-Z para distintas configuraciones de punto fijo con 16 bits para parte fraccionaria.

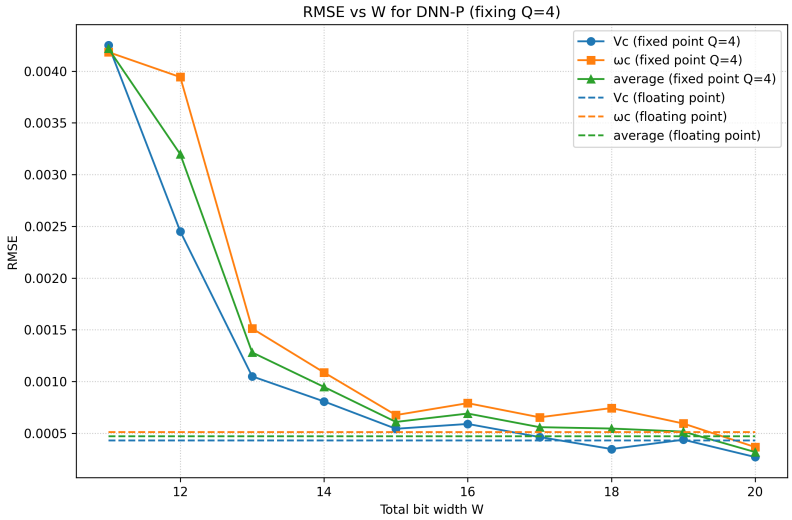


(b) RMSEs de las salidas de DNN-Z para distintas configuraciones de punto fijo con 8 bits para parte entera con signo.

Fig. 5.7: RMSEs de redes cuantizadas para DNN-Z.



(a) RMSEs de las salidas de DNN-P para distintas configuraciones de punto fijo con 16 bits para parte fraccionaria.



(b) RMSEs de las salidas de DNN-P para distintas configuraciones de punto fijo con 4 bits para parte entera con signo.

Fig. 5.8: RMSEs de redes cuantizadas para DNN-P.

Resultados de simulación

En esta sección se muestran los resultados de simulación, comparando los resultados obtenidos al utilizar una formulación implícita y la explícita con DNNs. Para la implícita se utiliza el mismo solver que se utilizó para generar los datos, es decir, *quadprog* con el algoritmo *active-set* en Matlab considerando horizontes de predicción $N = 3$. La DNNs utilizadas son las que se obtuvieron mediante las exploraciones de las secciones anteriores, pero además se le aplica *pruning* a todos los pesos menores a 0.001, y mediante simulaciones se verificó que el *pruning* tiene un efecto despreciable en el desempeño. Los parámetros de las redes utilizadas y los RMSEs previos al *pruning* se muestran en la Tabla 5.3.

La Figura 5.9 muestra las señales relevantes del control en nivel cero. Se observa un correcto seguimiento de v_{cd} con solver y con redes neuronales, excepto cuando se activa la restricción hexagonal del voltaje de entrada v_s, la cual se redujo un 92 % para forzar su activación. Las señales obtenidas utilizando redes aproximan correctamente la ley de control, observándose algunas diferencias en la zona activa de restricción y en el último transiente. En particular, en la zona de activación de restricción, las señales v_{cd} y v_{sd} obtenidas con redes no oscilan de la misma manera que las señales obtenidas con solver.

La Figura 5.10 muestra las corrientes de red y el ángulo de fase. Las corrientes siguen las referencias dentro de lo posible, saturándose cuando se activan las restricciones, y la comparación entre las señales con solver y redes neuronales es similar a lo observado en el control en nivel cero.

La Figura 5.11 muestra la magnitud y frecuencia del voltaje en la salida del filtro. Se observa que la frecuencia se mantiene dentro del rango considerado, y el resultado con solver es similar al de redes neuronales, excepto en la zona de activación de la restricción hexagonal de voltaje y en el transiente final.

La Figura 5.12a muestra las corrientes del filtro dentro de la zona de restricción circular suave, la cual fue aproximada con un dodecágono. La Figura 5.12b muestra los voltajes de la salida del inversor dentro de la zona de restricción dura hexagonal, la cual fue reducida a un 92 % para forzar su activación.

Tabla 5.3: Parámetros y desempeño de las redes neuronales cuantizadas entrenadas.

Red	**L**	**M**	***sparsity***	**W**	**Q**	**RMSE prom.**
DNN-Z	4	30	70.53 %	22	8	0.0081
DNN-P	4	30	71.18 %	15	4	0.00061

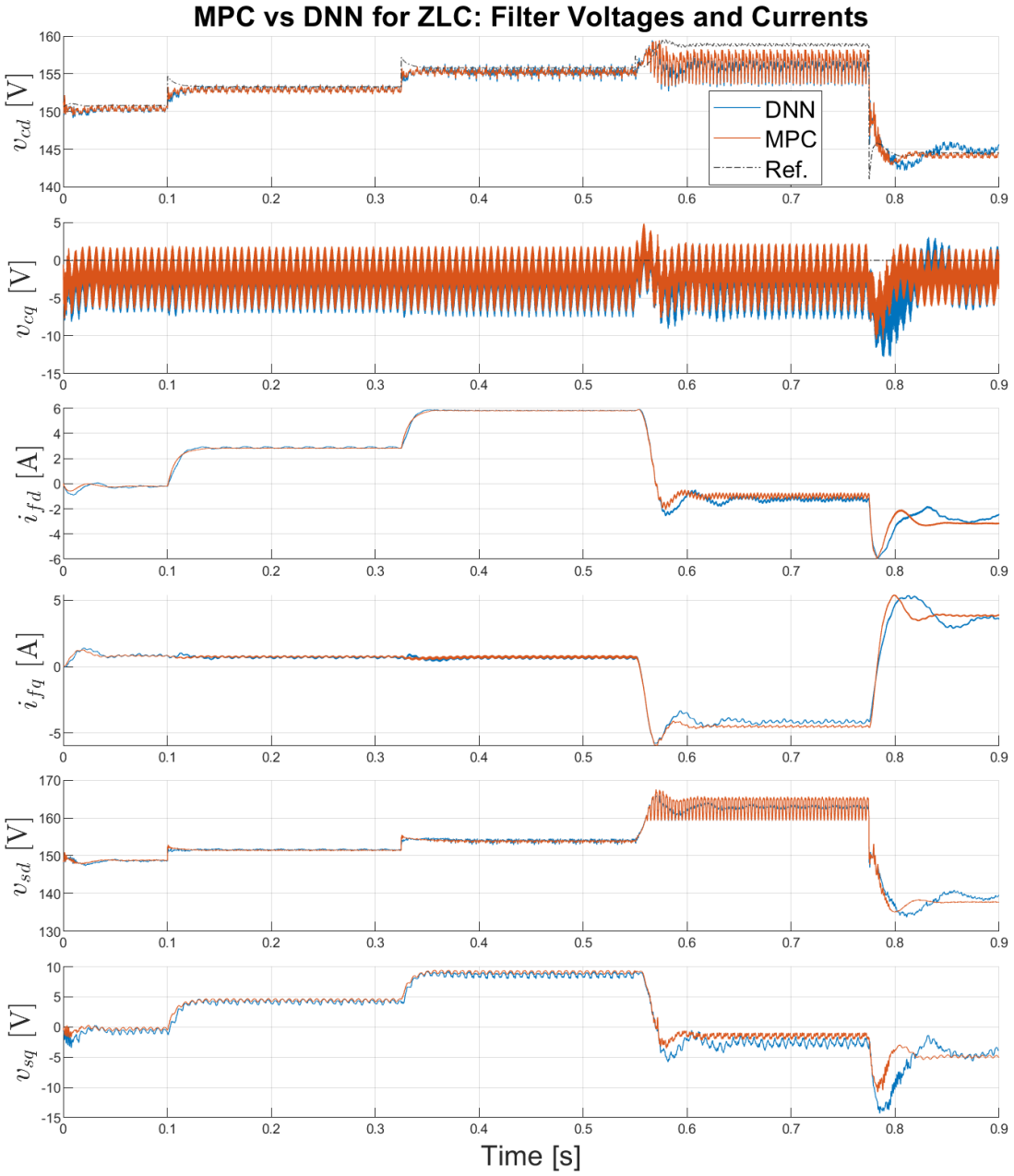


Fig. 5.9: Señales en el plano *dq* asociadas al control en nivel cero. Se incluyen voltajes del capacitor del filtro, corrientes del filtro y voltajes aplicados por el 2L-VCS.

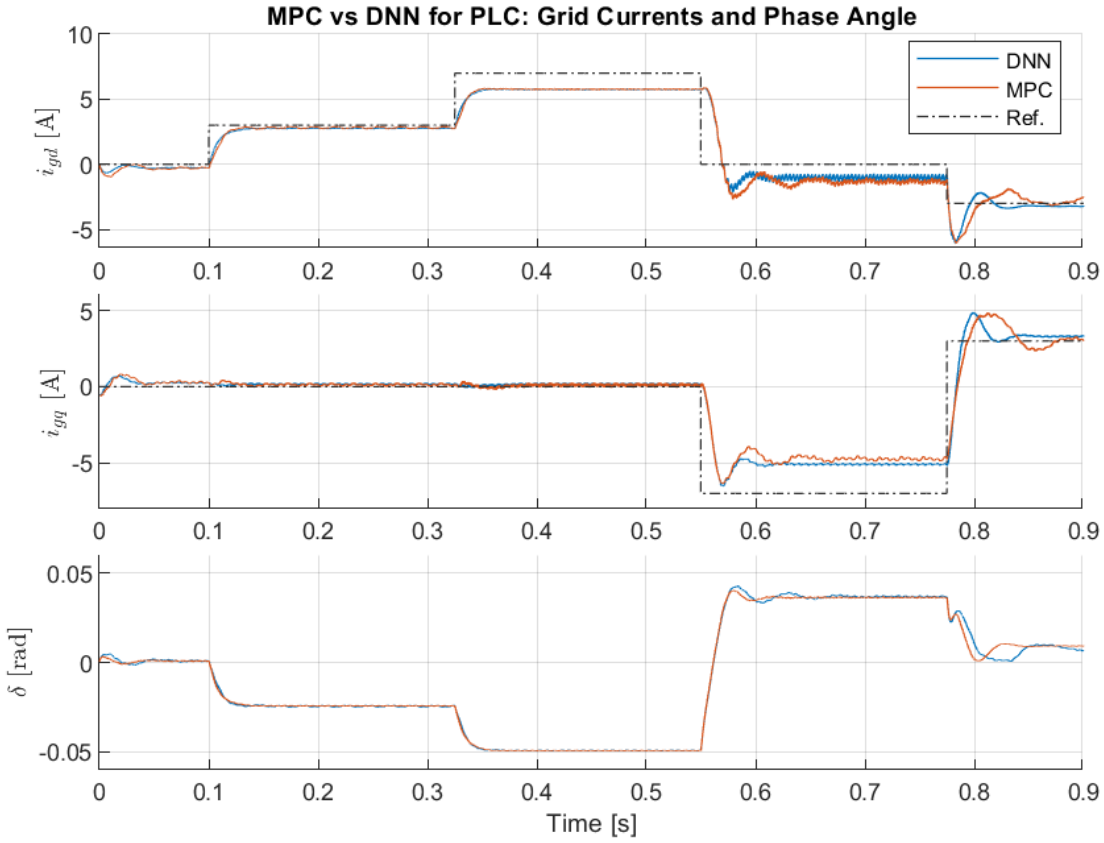


Fig. 5.10: Corrientes de la red en el plano dq y estimación del ángulo de fase.

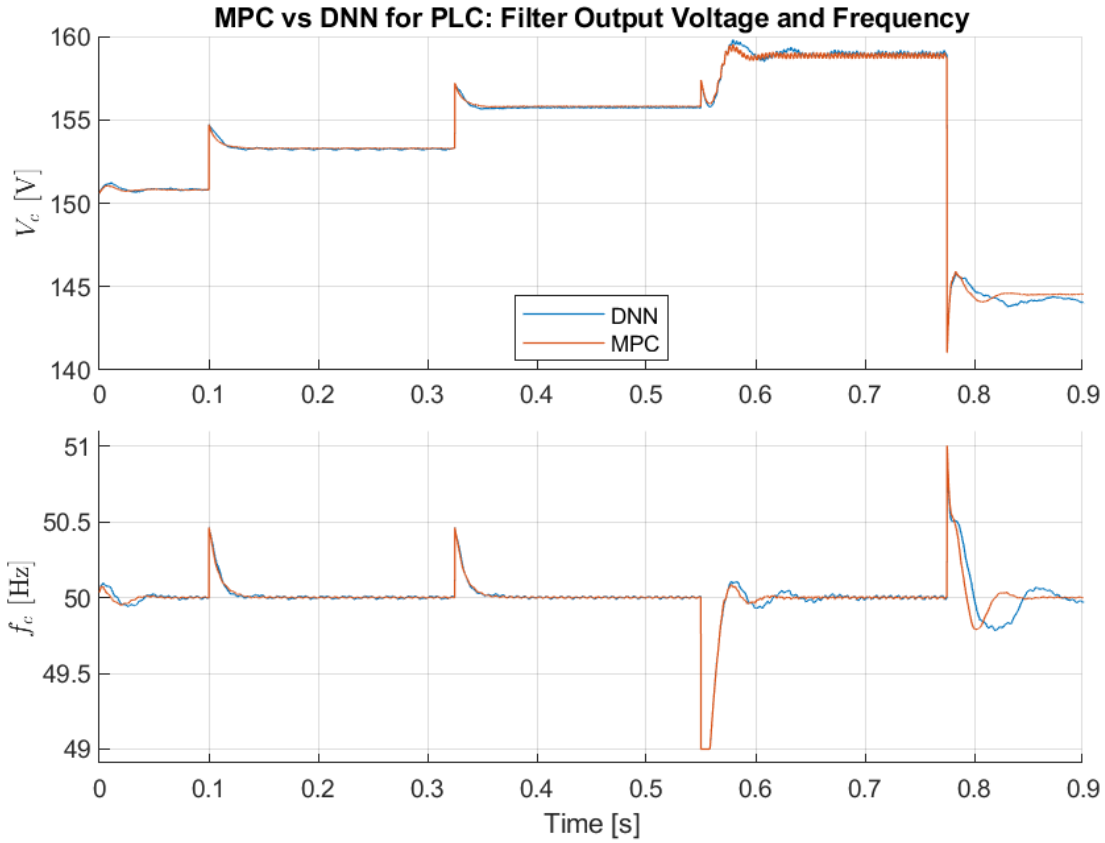
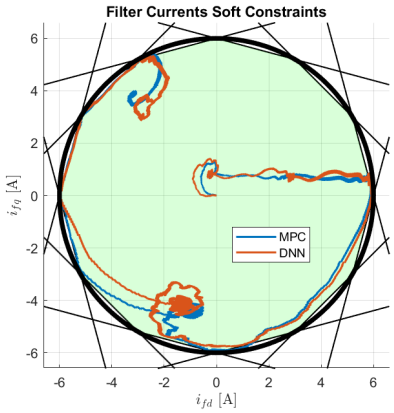
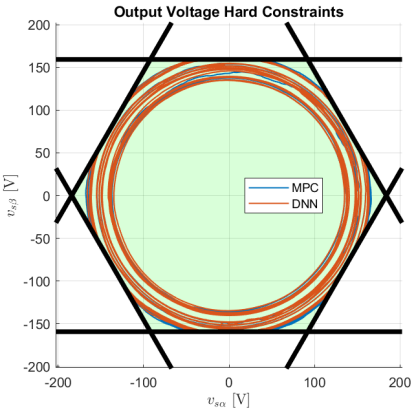


Fig. 5.11: Magnitud y frecuencia del voltaje en la salida del filtro.



(a) Corrientes del filtro en el plano dq, con restricción suave circular aproximada con un dodecágono.



(b) Voltajes de salida del inversor en el plano $\alpha\beta$, con restricción dura hexagonal reducida a un 92 %.

Fig. 5.12: Señales con restricciones politópicas

5.3. Simulaciones Software-in-the-Loop

Para obtener la descripción de hardware en Verilog de las redes neuronales se utiliza HLS4ML, como se explicó en la Sección 3.2.2. Se utiliza la estrategia *Resources*, con un factor de reutilización dos, ya que es el menor factor de reutilización que permite una solución con DSPs suficientes. Para las configuraciones de punto fijo de HLS4ML se utilizan las mismas que en Matlab para las simulaciones MIL, que corresponden a saturación en caso de *overflow* y opción de redondeo `Round`. La Tabla 5.4 muestra los recursos y latencias reportados a nivel de *HLS Synthesis* por HLS4ML, donde preliminarmente se estima una cantidad de recursos que excede la capacidad de la FPGA, pero es una sobreestimación.

Cosimulación

Para verificar que las descripciones de hardware representan correctamente a las redes neuronales, se efectúan cosimulaciones utilizando Vitis Model Composer (como se explicó en la Sección 3.2.2), utilizando *golden references* de 10 mil datos.

Las Figuras 5.13 y 5.14 muestran los resultados de la cosimulación de DNN-Z y DNN-P, con precisión de punto fijo $W = 22, Q = 8$ y $W = 15, Q = 4$ respectivamente. En los gráfico se muestran las diferencias obtenidas entre la salidas de las redes y la salidas de sus descripciones en Verilog para los datos de las *golden references*. Se observan diferencias de a lo más en el segundo bit menos significativo.

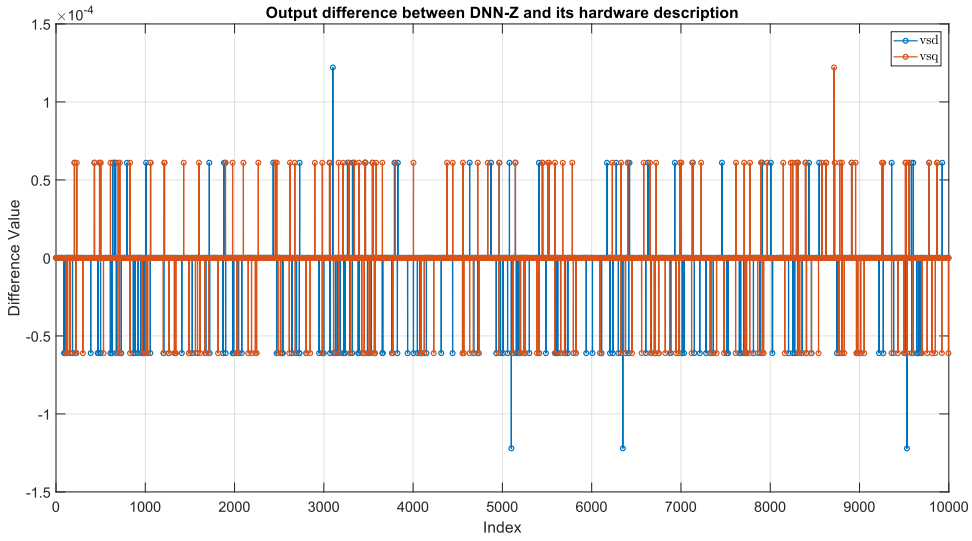


Fig. 5.13: Diferencia entre las salidas de DNN-Z y de su descripción en hardware ($W = 22, Q = 8$).

Tabla 5.4: Recursos y latencias de DNNs reportados a nivel de *HLS Synthesis*.

DNN	**LUTs**	**FFs**	**DSPs**	**Latencia [μs]***
DNN-Z	308238	56154	405	0.35
DNN-P	242765	28030	389	0.29

*Con reloj de frecuencia 100[MHz].

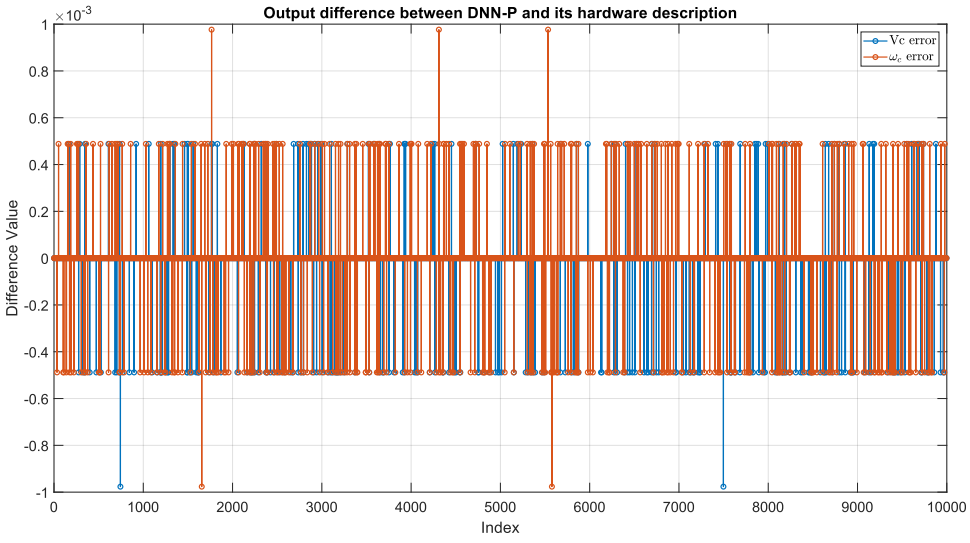


Fig. 5.14: Diferencia entre las salidas de DNN-P y de su descripción en hardware ($W = 15, Q = 4$).

Capítulo 6

CONCLUSIONES

En esta tesis se desarrollaron y validaron flujos sistemáticos de diseño para la implementación de lazos de control predictivo en FPGA, orientada a aplicaciones de control en tiempo real con restricciones de latencia y restricciones de operación. Dicho flujo abarca desde la formulación del problema de control y su discretización, hasta la generación de descripciones de hardware optimizadas, permitiendo una transición estructurada desde el diseño en alto nivel hacia su implementación en arquitecturas reconfigurables.

Los flujos propuestos fueron validados completamente en un motor DC, sobre el cual se desplegaron tanto lazos MPC implícitos como explícitos basados en redes neuronales profundas. En ambos casos, se logró una implementación funcional y estable, demostrando que las arquitecturas hardware desarrolladas permiten ejecutar el control predictivo dentro de márgenes temporales compatibles con sistemas de alta dinámica. Los resultados experimentales obtenidos en el motor DC confirman que la aceleración en FPGA reduce de manera significativa la latencia del lazo de control, sin afectar negativamente el desempeño dinámico del sistema.

Adicionalmente, el flujo para MPC explícito con redes neuronales fue aplicado a un segundo sistema de mayor complejidad, correspondiente a un sistema DER, para el cual se realizó una validación hasta la etapa de *Software-in-the-Loop*. Este estudio permitió verificar la escalabilidad conceptual de los procedimientos propuestos y su aplicabilidad a sistemas de mayor dimensión, aun cuando la implementación completa en hardware quedó fuera del alcance experimental de esta tesis. Los resultados en SIL muestran que el enfoque propuesto es consistente y transferible, estableciendo una base sólida para futuras implementaciones en hardware de MPC para este tipo de sistemas

Un aporte central de este trabajo corresponde al análisis detallado del uso de recursos de FPGA y la caracterización temporal de los lazos de control implementados. Este análisis permitió identificar los principales cuellos de botella computacionales y establecer comparaciones objetivas entre las distintas aproximaciones de control consideradas, evidenciando los compromisos existentes entre precisión, complejidad hardware y velocidad de ejecución. En particular, se observó que las formulaciones explícitas basadas en redes neuronales permiten alcanzar latencias notablemente reducidas, mientras que los enfoques implícitos ofrecen una mayor flexibilidad frente a cambios en el modelo y las restricciones.

En conjunto, los resultados de esta tesis demuestran que la combinación de control predictivo, aprendizaje automático y aceleración en FPGA constituye una solución viable y eficiente para mitigar el problema de la latencia en lazos de control avanzados. El flujo de diseño propuesto, junto con la validación experimental y el análisis de recursos y tiempos, aporta un marco práctico y reproducible que puede ser extendido a otras aplicaciones de control embebido de alta velocidad, tales como sistemas de potencia, robótica y aplicaciones industriales críticas. De esta forma, este trabajo contribuye a cerrar la brecha existente entre el diseño teórico de estrategias MPC y su implementación

efectiva en plataformas hardware con restricciones reales de tiempo y recursos.

Trabajo Futuro

- Completar el flujo de diseño propuesto para sistemas DER, avanzando desde la validación *Software-in-the-Loop* hacia implementaciones *Hardware-in-the-Loop* y pruebas experimentales.
- Extender los flujos propuestos hacia plataformas embebidas más especializadas, tales como sistemas en chip (SoC) con aceleradores dedicados o implementaciones a nivel de chip (ASIC).
- Explorar distintas arquitecturas de redes neuronales profundas, analizando su impacto en el compromiso entre precisión, robustez, latencia y uso de recursos en implementaciones en hardware de MPC explícito.
- Evaluar la factibilidad de implementar en FPGA algoritmos de control y aprendizaje de mayor complejidad computacional, tales como técnicas de aprendizaje reforzado, analizando tanto su desempeño dinámico como sus requerimientos temporales y de recursos.
- Profundizar en la automatización del flujo de diseño, incorporando herramientas que permitan la selección sistemática de precisiones fijas, grados de paralelismo y arquitecturas de cómputo en función de restricciones de latencia y recursos definidas por el usuario.

REFERENCIAS

[1] A. Alessio and A. Bemporad, *A Survey on Explicit Model Predictive Control*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 345–369.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[3] M. Jeong, M. Schoen, and J. Biela, "When FPGAs meet ADMM with high-level synthesis (HLS): A real-time implementation of long-horizon MPC for power electronic systems," in *2023 11th International Conference on Power Electronics and ECCE Asia (ICPE 2023 - ECCE Asia)*, 2023, pp. 1704–1711.

[4] T. Geyer, F. Torrisi, and M. Morari, "Optimal complexity reduction of polyhedral piecewise affine systems," *Automatica*, vol. 44, pp. 1728–1740, 07 2008.

[5] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, sep 2020. [Online]. Available: https://doi.org/10.1109%2Ftcyb.2020.2999556

[6] F. Dong, X. Li, K. You, and S. Song, "Standoff tracking using dnn-based MPC with implementation on FPGA," *IEEE Transactions on Control Systems Technology*, vol. 31, no. 5, pp. 1998–2010, 2023.

[7] A. Cortés, "Implementación y evaluación de aceleradores de cómputo para aplicaciones de control predictivo por modelo utilizando arquitecturas heterogéneas." Master's thesis, Universidad Técnica Federico Santa María, 2024.

[8] J. J. V. Cádiz, "ImplementaciÓn en fpga de redes neuronales artificiales aplicadas a lazos de control predictivo por modelo." Master's thesis, Universidad Técnica Federico Santa María, 2025.

[9] dSPACE. MicroLabBox. https://www.dspace.com/en/pub/home/products/hw/microlabbox.cfm. Accessed: 2025-24-03.

[10] P. J. Antsaklis and A. N. Michel, *Linear Systems*, 2nd ed. Boston: Birkhäuser, 2006.

[11] J. L. Jerez, E. C. Kerrigan, and G. A. Constantinides, "A sparse and condensed QP formulation for predictive control of LTI systems," *Automatica*, vol. 48, no. 5, pp. 999–1002, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109812001069

[12] T. V. Dang, K. Ling, and J. Maciejowski, "Embedded ADMM-based QP solver for MPC with polytopic constraints," in *2015 European Control Conference (ECC)*, 2015, pp. 3446–3451.

[13] J. Jerez, "Model predictive control for deeply pipelined field-programmable gate array implementation: algorithms and circuitry," *IET Control Theory Applications*, vol. 6, pp. 1029–1041(12), May 2012. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/iet-cta.2010.0441

[14] A. G. Wills, G. Knagge, and B. Ninness, "Fast linear model predictive control via custom integrated circuit architecture," *IEEE Transactions on Control Systems Technology*, vol. 20, no. 1, pp. 59–71, 2012.

[15] H. Peyrl, A. Zanarini, T. Besselmann, J. Liu, and M.-A. Boéchat, "Parallel implementations of the fast gradient method for high-speed MPC," *Control Engineering Practice*, vol. 33, pp. 22–34, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967066114002093

[16] S. Lucia, D. Navarro, Lucía, P. Zometa, and R. Findeisen, "Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 137–145, 2018.

[17] P. Tøndel, T. Johansen, and A. Bemporad, "Evaluation of piecewise affine control via binary search tree," *Automatica*, vol. 39, no. 5, pp. 945–950, 2003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109802003084

[18] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," 2014. [Online]. Available: https://arxiv.org/abs/1402.1869

[19] T. Serra, C. Tjandraatmadja, and S. Ramalingam, "Bounding and counting linear regions of deep neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1711.02114

[20] M. Abu-Ali, F. Berkel, M. Manderla, S. Reimann, R. Kennel, and M. Abdelrahem, "Deep learning-based long-horizon mpc: Robust, high performing, and computationally efficient control for pmsm drives," *IEEE Transactions on Power Electronics*, vol. 37, no. 10, pp. 12 486–12 501, 2022.

[21] Hecht-Nielsen, "Theory of the backpropagation neural network," in *International 1989 Joint Conference on Neural Networks*, 1989, pp. 593–605 vol.1.

[22] S. Ruder, "An overview of gradient descent optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1609.04747

[23] Y. Tian and Y. Zhang, "A comprehensive survey on regularization strategies in machine learning," *Information Fusion*, vol. 80, pp. 146–166, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S156625352100230X

[24] B. Rokh, A. Azarpeyvand, and A. Khanteymoori, "A comprehensive survey on model quantization for deep neural networks in image classification," *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 6, Nov. 2023. [Online]. Available: https://doi.org/10.1145/3623402

[25] Google. (2025) QKeras: Quantization extensions for Keras. https://github.com/google/qkeras. Accessed: 2025-03-17.

[26] P. Babu and P. Eswaran, "Reconfigurable FPGA architectures: A survey and applications," *Journal of The Institution of Engineers (India) Series B*, vol. 102, p. 143–156, 11 2020.

[27] AMD. Zynq 7000 SoCs. https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html. Accessed: 2025-24-03.

[28] E. D. Sozzo, D. Conficconi, A. Zeni, M. Salaris, D. Sciuto, and M. D. Santambrogio, "Pushing the level of abstraction of digital system design: A survey on how to program FPGAs," *ACM Comput. Surv.*, vol. 55, no. 5, dec 2022. [Online]. Available: https://doi.org/10.1145/3532989

[29] Fast Machine Learning Lab. HLS4ML. https://fastmachinelearning.org/hls4ml/. Accessed: 2025-24-03.

[30] Y. Zhang, M. Wang, B. Zhou, X. Hu, and D. Zhang, "Design and implementation of dSPACE using OpenSCENARIO to construct scenarios in vehicle simulation testing," in *2022 International Conference on Artificial Intelligence, Information Processing and Cloud Computing (AIIPCC)*, 2022, pp. 20–23.

[31] S. You, M. Greiff, R. Quirynen, S. Ran, Y. Wang, K. Berntorp, R. Dai, and S. Di Cairano, "Integral action nmpc for tight maneuvers of articulated vehicles," in *2023 American Control Conference (ACC)*, 2023, pp. 3155–3161.

[32] X. Mu, F. Cai, R. Zheng, D. Zhang, and D. Gu, "A predictive current control for aerospace servo motor," in *2021 3rd International Conference on Applied Machine Learning (ICAML)*, 2021, pp. 366–369.

[33] C. Xie, T. Xu, and R. Song, "A deep LSTM based semg-to-force model for a cable-driven rehabilitation robot," in *2022 International Conference on Advanced Robotics and Mechatronics (ICARM)*, 2022, pp. 660–665.

[34] A. Villalón, C. Muñoz, J. Muñoz, and M. Rivera, "A detailed dSPACE-based implementation of modulated model predictive control for AC microgrids," *Sensors*, vol. 23, no. 14, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/14/6288

[35] B. Bossoufi, M. Karim, M. Taoussi, H. A. Aroussi, M. Bouderbala, S. Motahhir, and M. Camara, "dSPACE-based implementation for observer backstepping power control of DFIG wind turbine," *IET Electric Power Applications*, vol. 14, pp. 2395–2403, 2020.

[36] dSPACE. Real-Time Interface. https://www.dspace.com/en/pub/home/products/sw/impsw/real-time-interface.cfm. Accessed: 2025-05-01.

[37] Francisco Abusleme. Implementation of MPC algorithms with hardware acceleration via FPGA using a dSPACE platform. https://github.com/vyrkka/mpc_fpga_dspace. Accessed: 2025-05-02.

[38] dSPACE. Model-Based Development. https://www.dspace.com/en/pub/home/applicationfields/ind-appl/researcheducation/model-based-development-.cfm. Accessed: 2025-17-04.

[39] Mathworks Support Team. What are MIL, SIL, PIL, and HIL, and how do they integrate with the Model-Based Design approach? https://www.mathworks.com/matlabcentral/answers/440277-what-are-mil-sil-pil-and-hil-and-how-do-they-integrate-with-the-model-based-design-approach. Accessed: 2025-17-04.

[40] Matlab. MATLAB Coder. https://www.mathworks.com/products/matlab-coder.html. Accessed: 2025-05-01.

[41] AMD. Vitis Model Composer. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-model-composer.html. Accessed: 2025-26-04.

[42] dSPACE. ControlDesk. https://www.dspace.com/en/pub/home/products/sw/experimentandvisualization/controldesk.cfm. Accessed: 2025-05-01.

[43] Quanser. (2023) Rotary servo base unit. Accessed: 2025-20-03. [Online]. Available: https://www.quanser.com/products/rotary-servo-base-unit/

[44] M. Azúa, "Implementación de un sistema de control predictivo usando el toolbox quarc," Master's thesis, Universidad Técnica Federico Santa María, 2022.

[45] Matlab. fimath. https://www.mathworks.com/help/fixedpoint/ref/embedded.fimath.html. Accessed: 2025-09-02.

[46] I. A. A. Brito, "Reinforcement learning-based primary control for der units integration in ac microgrids," Master's thesis, Universidad Técnica Federico Santa María, Valparaíso, Chile, May 2025, tesis de Magíster en Ciencias de la Ingeniería Electrónica.

[47] R. H. Lasseter, "Microgrids," *IEEE Power Engineering Society Winter Meeting*, vol. 1, pp. 305–308, 2002.

[48] M. Liserre, F. Blaabjerg, and S. Hansen, "Design and control of lcl-filter-based grid converters for power quality improvement," *IEEE Transactions on Industrial Electronics*, vol. 41, no. 5, pp. 1204–1211, 2005.

[49] J. M. Guerrero, J. C. Vasquez, J. Matas, M. Castilla, and L. G. de Vicuña, "Hierarchical control of droop-controlled ac and dc microgrids – a general approach toward standardization," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 1, pp. 158–172, 2011.

[50] P. Rodriguez, A. Luna, J. I. Candela, R. Mujal, R. Teodorescu, and F. Blaabjerg, "Multiresonant frequency-locked loop for grid synchronization of power converters under distorted grid conditions," in *IEEE Transactions on Industrial Electronics*, vol. 58, no. 1, 2011, pp. 127–138.

[51] N. Pogaku, M. Prodanovic, and T. C. Green, "Modeling, analysis and testing of autonomous operation of an inverter-based microgrid," *IEEE Transactions on Power Electronics*, vol. 22, no. 2, pp. 613–625, 2007.

[52] D. E. Olivares, A. Mehrizi-Sani, A. H. Etemadi, C. A. Cañizares, R. Iravani, M. Kazerani, A. H. Hajimiragha, O. Gomis-Bellmunt, M. Saeedifard, R. Palma-Behnke, G. A. Jiménez-Estévez, and N. D. Hatziargyriou, "Trends in microgrid control," *IEEE Transactions on Smart Grid*, vol. 5, no. 4, pp. 1905–1919, 2014.

[53] F. García-Torres and C. Bordons, "Optimal load sharing of distributed generators and storage devices in microgrids," *IEEE Transactions on Smart Grid*, vol. 7, no. 1, pp. 354–365, 2015.

[54] M. Cespedes and J. Sun, "Impedance modeling and analysis of grid-connected voltage-source converters," *IEEE Transactions on Power Electronics*, vol. 29, no. 3, pp. 1254–1261, 2014.