

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



“IMPLEMENTACIÓN DE CONTROL PREDICTIVO EXPLÍCITO USANDO REDES NEURONALES”

JOEL RIVERA FERNÁNDEZ

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO
CIVIL ELECTRÓNICO**

PROFESOR GUIA: JUAN CARLOS AGÜERO.

PROFESOR CORREFERENTE 1: GONZALO CARVAJAL.

PROFESOR CORREFERENTE 2: ÁNGEL LEONEL CEDEÑO.

Agradecimientos

En primer lugar, quiero expresar mi más profundo agradecimiento a mis padres, quienes han sido mi mayor apoyo durante todo este proceso. A ustedes, mamá y papá, por su amor incondicional, por enseñarme el valor del esfuerzo y por estar siempre a mi lado a pesar de la distancia, brindándome ánimo en los momentos difíciles. Este logro no habría sido posible sin su guía y su confianza en mí.

A las personas que conocí gracias a la universidad, en especial a mis amigos más cercanos, ya que gracias a su compañía este proceso se hizo más ameno. Gracias por estar ahí para apoyarme.

A los profesores que me guiaron durante mi estadía en la universidad, por su orientación y por compartir su conocimiento conmigo. Su dedicación y paciencia han sido fundamentales para completar este trabajo, y me siento afortunado de haber tenido la oportunidad de aprender de ustedes y desarrollar un amor por mi carrera.

Por último agradecer al Centro Avanzado de Ingeniería Eléctrica y Electrónica (AC3E) proyecto Basal ANID AFB240002, al proyecto Fondecyt ANID 1211630 del profesor Juan C. Agüero y al proyecto Fondecyt ANID 3240181 del profesor Ángel Cedeño.

A todos ustedes, gracias.

Resumen

Esta memoria presenta la implementación de un Control Predictivo Basado en Modelo (MPC) explícito utilizando redes neuronales, con el objetivo de mejorar el control de sistemas dinámicos complejos, particularmente un motor de corriente continua (DC). El MPC es una técnica avanzada que predice el comportamiento futuro de un sistema y ajusta las acciones de control en consecuencia. Sin embargo, su aplicación en tiempo real puede ser computacionalmente costosa debido a la necesidad de resolver un problema de optimización en cada ciclo de control.

El uso de redes neuronales permite aproximar la ley de control de un MPC explícito de manera más eficiente. Para lograrlo, se diseñó una red neuronal con tres capas ocultas y funciones de activación ReLU, entrenada con datos generados por simulaciones del MPC en Simulink. Esta red neuronal puede replicar el comportamiento del MPC al clasificar el estado del sistema en regiones predefinidas y aplicar la acción de control correspondiente.

Los resultados de las simulaciones mostraron que la red neuronal puede aproximar con éxito el control del MPC, reduciendo significativamente el tiempo de cómputo, lo que es especialmente útil en aplicaciones de control en tiempo real. Sin embargo, se observaron algunas limitaciones, como la dificultad de manejar restricciones de control estrictas con la misma precisión que el MPC tradicional.

Futuros trabajos pueden enfocarse en mejorar la robustez del controlador basado en redes neuronales e investigar su implementación en sistemas físicos.

Palabras clave: Control predictivo basado en modelo, MPC, redes neuronales, Simulink, motor de corriente continua.

Abstract

This project presents the implementation of an explicit Model Predictive Control (MPC) using neural networks, with the objective of improving the control of complex dynamic systems, particularly a DC motor. MPC is an advanced technique that predicts the future behavior of a system and adjusts control actions accordingly. However, its real-time application can be computationally expensive due to the need to solve an optimization problem at each control cycle.

The use of neural networks allows for a more efficient approximation of the control law of an explicit MPC. To achieve this, a neural network with three hidden layers and ReLU activation functions was designed and trained with data generated from MPC simulations in Simulink. This neural network can replicate the behavior of MPC by classifying the system's state into predefined regions and applying the corresponding control action.

The simulation results showed that the neural network successfully approximates the MPC control, significantly reducing computation time, which is particularly useful in real-time control applications. However, some limitations were observed, such as the difficulty of handling strict control constraints with the same precision as traditional MPC.

Future work may focus on improving the robustness of the neural network-based controller and investigating its implementation in physical systems.

Keywords: Model predictive control, MPC, neural networks, Simulink, DC motor.

Glosario

- **MPC (Model Predictive Control):** Control Predictivo Basado en Modelo. Técnica de control que utiliza un modelo del sistema para predecir su comportamiento futuro y optimizar las acciones de control en función de estas predicciones.
- **RNA (Redes Neuronales Artificiales):** Algoritmos inspirados en el funcionamiento del cerebro humano que aprenden a partir de grandes cantidades de datos, utilizados para el modelado y control de sistemas no lineales.
- **ReLU (Rectified Linear Unit):** Función de activación utilizada en redes neuronales, que genera una salida igual a 0 si la entrada es negativa, y una salida igual a la entrada cuando esta es positiva.
- **DC (Direct Current):** Corriente continua. En el contexto de motores eléctricos, se refiere a motores que son alimentados con corriente continua.
- **Simulink:** Entorno de simulación gráfica integrado con MATLAB, utilizado para modelar, simular y analizar sistemas dinámicos.
- **MATLAB:** Entorno de cálculo numérico y programación utilizado ampliamente en ingeniería para simulaciones, análisis de datos y diseño de controladores.
- **Luenberger Observer (Observador de Luenberger):** Un observador que utiliza un modelo del sistema y las salidas disponibles para estimar las variables de estado no medidas de un sistema.
- **Filtro de Kalman:** Algoritmo que estima el estado de un sistema dinámico en presencia de ruido en las entradas y mediciones, minimizando el error cuadrático medio de la estimación.
- **PID (Proporcional-Integral-Derivativo):** Controlador clásico que ajusta la acción de control basada en la proporción, la integral y la derivada del error de control.

- **Red Neuronal Profunda (Deep Neural Network):** Red neuronal con múltiples capas ocultas que permite modelar y aprender representaciones más complejas de datos.
- **Modelo en Espacio de Estado (State-Space Model):** Representación matemática de un sistema dinámico en términos de un conjunto de variables de estado y sus relaciones mediante ecuaciones diferenciales o diferencias.
- **Programación Cuadrática (Quadratic Programming):** Técnica de optimización utilizada en MPC, que implica la minimización de una función objetivo cuadrática sujeta a restricciones lineales.

Índice general

1.. <i>Introducción</i>	3
1.1. Estado del Arte	4
1.2. Objetivos específicos	6
1.3. Trabajos Relacionados	7
2.. <i>Control Predictivo Basado en Modelo (MPC)</i>	9
2.1. Control Predictivo con Acción Integral	10
2.2. Restricciones	14
2.3. Control Predictivo Basado en Modelos (MPC) Explícito	15
3.. <i>Observadores</i>	18
3.1. Modelado del Sistema	18
3.2. Observador de Luenberger	19
3.2.1. Estabilidad del Observador	20
3.3. Observadores de Orden Completo y de Orden Reducido	20
3.3.1. Observadores de Orden Completo	20
3.3.2. Observadores de Orden Reducido	20
3.4. Observadores de Kalman	20
3.5. Aplicaciones de los Observadores	21
4.. <i>Redes Neuronales</i>	22
5.. <i>Planta de interés</i>	26
5.1. Motor DC	26

5.2. Modelo dinámico del Motor DC	26
6.. <i>Observador Modelo extendido</i>	28
7.. <i>Implementación del Control Predictivo basado en Modelo</i>	29
7.1. Matlab/Simulink	30
7.1.1. Librerías importantes en esta memoria	31
7.1.2. Bloques de Simulink útiles para esta memoria	32
7.2. Implementación del MPC en Simulink	35
7.2.1. Extracción de datos de simulación	44
8.. <i>Implementación de la Red Neuronal</i>	47
8.1. Diseño de arquitectura Red Neuronal	47
8.2. Entrenamiento Red Neuronal	48
8.3. Implementación en Simulink	51
9.. <i>Resultados de simulación</i>	55
9.1. Resultados MPC	55
9.2. Demostración de acción integral MPC	60
9.3. Resultados Red Neuronal	68
10.. <i>Conclusiones</i>	72
10.1. Conclusiones Generales	72
10.2. Limitaciones	73
10.3. Trabajos Futuros	74

1. INTRODUCCIÓN

Durante la última década, el avance en el desarrollo y aplicación de Redes Neuronales Artificiales (RNA) como herramienta para el modelado y control de procesos ha demostrado ser una alternativa más que eficaz en comparación a métodos tradicionales. En particular, su uso en Control Predictivo se ha vuelto un punto clave de investigación en la última década debido a que este tipo de control fue relegado a procesos lentos en sus inicios debido a los largos tiempos necesarios para solucionar los problemas de optimización a los que se enfrentaba. El Control Predictivo se basa en que se intenta anticipar el comportamiento futuro de un sistema y actuar en consecuencia. Este proceso de predecir y actuar entre cada toma de muestras del Control Predictivo resulta siendo muy exigente a nivel de hardware y poco eficiente al usar técnicas tradicionales, sin embargo, el implementar un Control Predictivo en conjunción con las Redes Neuronales puede aportar mejoras considerables en predicción y eficiencia.

En este contexto, este trabajo se enfocará en la implementación de un Controlador Predictivo basado en Modelo (MPC) explícito basado en Redes Neuronales. El objetivo es diseñar y aplicar un controlador que permita mejorar el rendimiento de sistemas dinámicos complejos al utilizar la potencia de las Redes Neuronales para modelar y predecir el comportamiento de un sistema. Se espera que esta implementación proporcione una base sólida para futuros trabajos de investigación de temas similares.

1.1. Estado del Arte

Las Redes Neuronales Artificiales (RNA o NN por sus siglas en inglés) son algoritmos que intentan imitar y simular el cerebro humano con el objetivo de “aprender” en base a grandes cantidades de datos. Mientras que una red neuronal biológica cuenta con axiones, dendritas, conexiones sinápticas y las mismas neuronas, una RNA está conformada por agrupaciones de “neuronas artificiales”, donde cada una realiza cálculos muy simples.

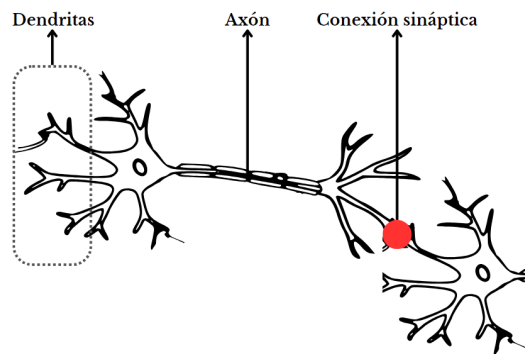


Fig. 1.1: Componentes de una neurona biológica

Una RNA normalmente está compuesta por tres partes: una capa de entrada, encargada de recoger la información; una o más capas ocultas, encargadas del procesamiento; y una capa de salida encargada de entregar el resultado. Cabe destacar que si la RNA cuenta con solo una capa oculta es conocida como una Red Neuronal Llana o poco profunda, mientras que si tiene dos o más pasa a ser conocida como una “Red Neuronal Profunda” (“Deep Neural Network” en inglés) [1].

Las neuronas en estas RNA operan recogiendo señales de entrada, las cuales pueden ser datos numéricos o salidas de otras neuronas. Estas señales son multiplicadas por su “peso” asociado, para sumarse y ser procesadas mediante una señal de activación, la cual está encargada de determinar si la señal es transmitida o no (es decir, si la neurona se activa o no).

En este proyecto se pondrá especial atención al uso de las funciones de activación

ReLU (Rectified Linear Unit), las cuales son principalmente utilizadas en las capas ocultas de la RNA, dado que estas permiten introducir no linealidad a los modelos de las redes, pero evitando los problemas causados por otras funciones de activación utilizadas como lo serían la tangente hiperbólica y sigmoide. Uno de estos problemas serían el desvanecimiento del gradiente, el cual se presenta cuando se entrenan redes neuronales con métodos de aprendizaje basados en el gradiente con retropropagación, y se traduce en que los pesos en las capas iniciales apenas se actualizarán durante el proceso de entrenamiento, por lo que esto derivará en un aprendizaje lento o incluso un posible estancamiento del proceso de optimización.

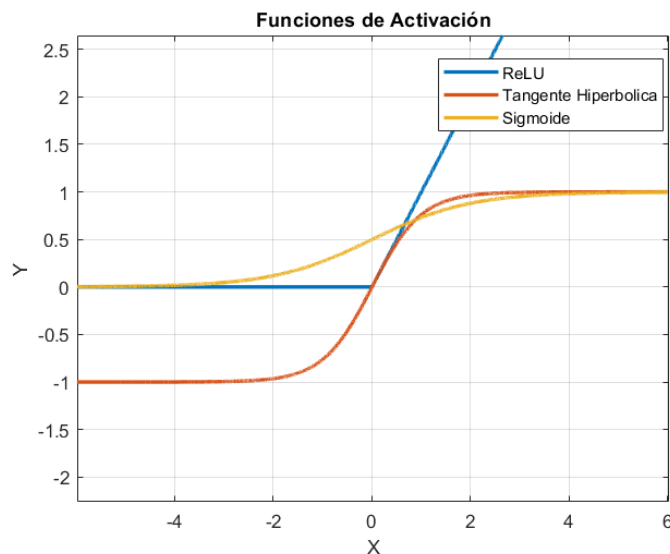


Fig. 1.2: Funciones de activación

Por otro lado, el Control Predictivo basado en Modelo es una rama del control automático que se centra en predecir el comportamiento futuro de un sistema basándose en el estado actual de éste y las posibles acciones de control. Las predicciones luego son utilizadas para seleccionar la mejor acción de control que minimice una función de coste determinada en cierto horizonte de predicción.

Los puntos más importantes a la hora de utilizar Control Predictivo basado en modelo son:

- **Modelo del Sistema:** Es necesario un modelo del sistema, el cual puede ser obtenido mediante las ecuaciones físicas que rigen a este, o usando técnicas de aprendizaje automático.
- **Predicciones:** Al usar el modelo del sistema, se hacen predicciones en un horizonte definido previamente. Mientras más amplio sea este, más exigente a nivel de recursos será el MPC.
- **Optimización:** Basándose en las predicciones, se calcula la acción de control que satisfaga un problema de optimización definido.

El proceso constituido por los puntos antes comentados son repetidos constantemente para lograr el control eficaz del sistema.

Ahora, al combinar el Control Predictivo con las Redes Neuronales se obtienen varias ventajas. Por un lado, las RNA han demostrado ser eficaces para modelar sistemas no lineales complejos que son difíciles o incluso imposibles de modelar con ecuaciones físicas. Por otro lado, el Control Predictivo maneja restricciones que pueden ser impuestas al estado del sistema o a las acciones de control que pueden ser tomadas, por lo que aplicando ambos elementos antes mencionados (RNA y MPC) se puede lograr un mejor rendimiento y una mayor cantidad de aplicaciones en comparación a métodos de control convencionales.

1.2. Objetivos específicos

Durante el desarrollo de esta memoria se espera cumplir los siguientes objetivos principales:

- Implementar un control predictivo basado en modelo con acción integral en MATLAB/Simulink.
- Validar la implementación del MPC mediante una simulación en Simulink.

- Diseñar, entrenar e implementar una red neuronal prealimentada con función de activación ReLU con el objetivo de lograr un control similar al logrado por el MPC.
- Comparar el control logrado por el MPC y la Red Neuronal.

1.3. Trabajos Relacionados

La implementación de un Control Predictivo Explícito utilizando redes neuronales ha sido un tópico de alto interés durante la última década debido a que gracias al avance en el campo del aprendizaje profundo se ha abierto la posibilidad de usar estas técnicas de aprendizaje para modelar sistemas complejos, permitiendo así el diseñar sistemas de control más avanzados y eficaces.

Dentro de los estudios que se destacan por el uso que se le puede dar en este proyecto se encuentra *Efficient Representation and Approximation of Model Predictive Control Laws via Deep Learning*[1], donde se muestra que una red neuronal con funciones de activación ReLU puede representar exactamente la función afín por partes resultante de la formulación de un MPC de sistemas lineales e invariantes en el tiempo. Asimismo, en este documento se plantean una forma de calcular los límites teóricos para la cantidad mínima de capas y neuronas por capa de una red neuronal, de forma que esta pueda representar exactamente cierta ley de un MPC. Lo anterior permite aproximar eficientemente las leyes de un control predictivo. Finalmente, se sugieren distintas alternativas para corregir o cuantificar los errores de aproximación y se muestran ejemplos de aplicaciones de lo presentado en sistemas controlados simples.

En [2] se presenta un método para calcular la ley de un MPC explícito utilizando el conocimiento del sistema para entrenar una red neuronal. Además explica que la complejidad computacional de un MPC explícito aumenta rápidamente al aumentar el número de restricciones y el método que se utiliza en MPC rápidos se centran en acelerar el proceso de optimización en línea.

En general, los textos citados muestran distintos tipos de aproximación, evaluación y modelado de un MPC explícito, buscando que éste sea más rápido y eficiente. También se puede usar esta información a la hora de utilizar una red neuronal para implementar un MPC explícito.

2. CONTROL PREDICTIVO BASADO EN MODELO (MPC)

El control Predictivo basado en Modelo o MPC es una técnica de control avanzado que consiste en utilizar un modelo matemático y el valor de los estados de un sistema para predecir las futuras salidas de una planta en base a una secuencia de acciones que el MPC simula. El MPC selecciona la secuencia de acciones que mejor resuelve un problema de optimización que busca seguir de la mejor manera posible a la referencia sin romper las restricciones definidas para los valores que pueden tomar sus entradas y salidas. [3]

Los cuatro pilares del Control Predictivo basado en Modelo son:

- **Modelo del Sistema:** Modelo matemático de los estados del sistema en tiempo discreto. Se utiliza para estimar la dinámica del sistema.
- **Función Objetivo:** Es la función de costos del problema de optimización que resolverá el MPC para encontrar la mejor solución.
- **Restricciones:** Se relacionan directamente con la función objetivo, ya que al resolver el problema de optimización las soluciones deben cumplir con las restricciones para que se consideren válidas. Estas restricciones generalmente representan los valores mínimos y máximos que pueden tomar las entradas, salidas y/o estados del sistema.
- **Horizonte de predicción:** Define cuantas muestras a futuro predice el controlador.

El desempeño de este tipo de controlador está muy ligado a la exactitud del modelo

matemático del sistema [4]. Si el modelo del sistema no representa correctamente la dinámica de este, las predicciones del MPC no serán acertadas. Esto se traduce en que las acciones de control entregadas no serían realmente óptimas y podrían llevar a la inestabilidad del sistema.

2.1. Control Predictivo con Acción Integral

Los algoritmos estándar de Control Predictivo Basado en Modelos (MPC) típicamente no incluyen una acción integral debido a que no siempre es necesario o deseable para lograr el control óptimo. El MPC predice el comportamiento futuro del sistema y ajusta las entradas para minimizar el error, lo cual, en sistemas bien modelados y sin perturbaciones constantes, permite corregir el error sin necesidad de una acción integral explícita. El agregar una acción integral puede complicar el problema de optimización al añadir un estado adicional que acumule el error. En sistemas donde no hay errores persistentes o perturbaciones no modeladas, la acción integral no es óptima porque puede llevar a sobreajustes o problemas de estabilidad.[3]

Sin embargo, si se quiere tener acción integral en el controlador debido a que el sistema a controlar presenta offsets en espacio estacionario o perturbaciones constantes, hay métodos que permiten conseguirlo. Uno de estos métodos se describe en [5], donde al trabajar con modelos de desviación se logra remover perturbaciones constantes o de lenta variación de las ecuaciones de estado y salida.

Dado un proceso cuyo modelo en variable de estados es:

$$x_{k+1} = Ax_k + Bu_k + v_k \quad (2.1)$$

$$y_k = Cx_k + w_k \quad (2.2)$$

donde $x_k \in \mathbb{R}^n$ es el vector de estados, $u_k \in \mathbb{R}^r$ es el vector de entradas de control,

$y_k \in \mathbb{R}^m$ es el vector de salidas del sistema, y A , B y C son las matrices del sistema conocidas de las dimensiones apropiadas. Las perturbaciones v_k y w_k son desconocidas.

Se utilizará como base el controlador MPC sujeto a la siguiente función objetivo:

$$J_k = \sum_{i=1}^L ((y_{k+i} - r_{k+i})^T Q_i (y_{k+i} - r_{k+i}) + \Delta u_{k+i-1}^T P_i \Delta u_{k+i-1}), \quad (2.3)$$

donde k es el tiempo discreto presente, $\Delta u_k = u_k - u_{k-1}$ es la razón de cambio del control, r_k es la señal de referencia, Q_i y P_i son matrices simétricas semidefinidas positivas de las dimensiones apropiadas y L es el horizonte de predicción. Para horizontes de predicción finitos L , Q_L puede ser la solución de la ecuación de Riccati del problema para asegurar la estabilidad del lazo cerrado.

La función objetivo (2.3) puede ser escrita de forma más compacta como:

$$J_k = (y_{k+1|L} - r_{k+1|L})^T Q (y_{k+1|L} - r_{k+1|L}) + \Delta u_{k|L}^T P \Delta u_{k|L}, \quad (2.4)$$

donde $Q \in \mathbb{R}^{Lm \times Lm}$ es una matriz diagonal por bloques con $Q_i \forall i = 1, \dots, L$ en sus bloques diagonales. $P \in \mathbb{R}^{Lr \times Lr}$ se define de forma similar con $P_i \forall i = 1, \dots, L$ en sus bloques diagonales. La notación utilizada para definir los vectores extendidos $y_{k+1|L}$, $r_{k+1|L}$ y $\Delta u_{k|L}$ se define como: Dado i y j números enteros, el vector extendido $s_{i|j}$ queda definido por:

$$s_{j|i} \stackrel{def}{=} \begin{bmatrix} s_j \\ s_{j+1} \\ \vdots \\ s_{j+i-1} \end{bmatrix} \quad (2.5)$$

Para poder resolver el problema de control óptimo de MPC se necesita un modelo que sea independiente de las perturbaciones desconocidas. De la ecuación de estados

(2.1) se obtiene

$$\Delta x_{k+1} = A\Delta x_k + B\Delta u_k, \quad (2.6)$$

donde $\Delta x_k = x_k - x_{k-1}$. De la ecuación de salidas del sistema (2.2) se obtiene

$$y_k = y_{k-1} + C\Delta x_k. \quad (2.7)$$

Al extender las ecuaciones (2.6) con (2.7) se genera el modelo de espacio de estados

$$\begin{bmatrix} \Delta x_{k+1} \\ y_k \end{bmatrix} = \begin{bmatrix} A & 0_{n \times m} \\ C & I_{m \times m} \end{bmatrix} \begin{bmatrix} \Delta x_k \\ y_{k-1} \end{bmatrix} + \begin{bmatrix} B \\ 0_{m \times r} \end{bmatrix} \Delta u_k, \quad (2.8)$$

$$y_k = \begin{bmatrix} C & I_{m \times m} \end{bmatrix} \begin{bmatrix} \Delta x_k \\ y_{k-1} \end{bmatrix} \quad (2.9)$$

Si se toma como referencia $r_k \neq 0$, las ecuaciones (2.8) y (2.9) pueden ser reescritas como

$$\overbrace{\begin{bmatrix} \Delta x_{k+1} \\ y_k - r_k \end{bmatrix}}^{\tilde{x}_{k+1}} = \overbrace{\begin{bmatrix} A & 0_{n \times m} \\ C & I_{m \times m} \end{bmatrix}}^{\tilde{A}} \overbrace{\begin{bmatrix} \Delta x_k \\ y_{k-1} - r_{k-1} \end{bmatrix}}^{\tilde{x}_k} + \overbrace{\begin{bmatrix} B \\ 0_{m \times r} \end{bmatrix}}^{\tilde{B}} \Delta u_k, \quad (2.10)$$

$$\overbrace{y_k - r_k}^{\tilde{y}_k} = \overbrace{\begin{bmatrix} C & I_{m \times m} \end{bmatrix}}^{\tilde{C}} \overbrace{\begin{bmatrix} \Delta x_k \\ y_{k-1} - r_{k-1} \end{bmatrix}}^{\tilde{x}_k} \quad (2.11)$$

Por lo tanto, se tiene un modelo de espacio de estados estrictamente propio de la forma

$$\tilde{x}_{k+1} = \tilde{A}\tilde{x}_k + \tilde{B}\Delta u_k, \quad (2.12)$$

$$\tilde{y}_k = \tilde{C}\tilde{x}_k \quad (2.13)$$

con $\tilde{x}_k \in \mathbb{R}^{n_e}$, $\Delta u_k \in \mathbb{R}^{r_e}$ y $\tilde{y}_k \in \mathbb{R}^{m_e}$, y con $\tilde{A}$, $\tilde{B}$ y $\tilde{C}$ de dimensiones adecuadas.

Para que sea posible controlar este sistema es necesario tener los estados $\tilde{x}_k$, que pueden ser obtenidos mediante el uso de observadores de estado. La teoría de dichos

observadores serán descritos en el Capítulo 3, mientras que el desarrollo del que será utilizado en esta memoria será expuesto en el Capítulo 6.

El desarrollo de esta memoria utiliza observadores porque se asume que los estados del sistema no son medibles directamente. Sin embargo, es posible utilizar directamente los estados del sistema (siempre y cuando estos puedan ser medidos directamente). Se demostrará el efecto que tiene el utilizar la medición directa de los estados en vez de usar un observador para estimarlos en el Capítulo 9, específicamente en la sección 9.2.

Los modelos de espacio de estado de las ecuaciones (2.12) y (2.13) se usan para definir un Modelo de Predicción de la forma

$$\tilde{y}_{k+1|L} = p_L + F_L \Delta u_{k|L}, \quad (2.14)$$

donde

$$p_L = O_L \tilde{A} \tilde{x}_k, \quad (2.15)$$

$$F_L = \begin{bmatrix} O_L \tilde{B} & H_L^d \end{bmatrix}, \quad (2.16)$$

con O_L como la matriz de observabilidad extendida del par de matrices $\tilde{A}$, $\tilde{C}$, definida como:

$$O_L = \begin{bmatrix} \tilde{C} \\ \tilde{C} \tilde{A} \\ \vdots \\ \tilde{C} \tilde{A}^{L-1} \end{bmatrix} \quad (2.17)$$

y H_L^d la matriz Toeplitz de las matrices de respuesta a impulso $\tilde{C} \tilde{A}^{i-1} \tilde{B}$ definida

como:

$$H_L^d = \begin{bmatrix} 0 & 0_{m_e \times r_e} & 0_{m_e \times r_e} & \dots & 0_{m_e \times r_e} \\ \tilde{C}\tilde{B} & 0 & 0_{m_e \times r_e} & \dots & 0_{m_e \times r_e} \\ \tilde{C}\tilde{A}\tilde{B} & \tilde{C}\tilde{B} & 0 & \dots & 0_{m_e \times r_e} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \tilde{C}\tilde{A}^{L-2}\tilde{B} & \tilde{C}\tilde{A}^{L-3}\tilde{B} & \tilde{C}\tilde{A}^{L-4}\tilde{B} & \dots & 0 \end{bmatrix} \quad (2.18)$$

La función objetivo (2.3) junto con el Modelo de Predicción de (2.14), pueden ser escritos como una función cuadrática en forma estándar, es decir,

$$J_k = \Delta u_{k|L}^T H \Delta u_{k|L} + 2f_k^T \Delta u_{k|L}, \quad (2.19)$$

donde

$$H = F_L^T Q F_L + P \quad (2.20)$$

$$f_k = F_L^T Q(p_L - r_{k+1|L}). \quad (2.21)$$

2.2. Restricciones

Se consideran restricciones para la amplitud

$$u_{min} \leq u_{k|L} \leq u_{max} \quad (2.22)$$

y para la razón de cambio de la entrada del sistema

$$\Delta u_{min} \leq \Delta u_{k|L} \leq \Delta u_{max}, \quad (2.23)$$

Dichas restricciones pueden ser escritas como una inecuación lineal de matrices

$$\mathcal{A} \Delta u_{k|L} \leq b_k, \quad (2.24)$$

donde

$$\mathcal{A} = \begin{bmatrix} I_{(rL \times rL)} \\ -I_{(rL \times rL)} \\ T \\ -T \end{bmatrix} \quad (2.25)$$

y

$$b_k = \begin{bmatrix} \Delta u_{max} \\ -\Delta u_{min} \\ u_{max} - cu_{k-1} \\ -u_{min} + cu_{k-1} \end{bmatrix} \quad (2.26)$$

Donde $T \in \mathbb{R}^{Lr \times Lr}$ y $c \in \mathbb{R}^{Lr \times r}$ están definidas como:

$$T = \begin{bmatrix} I_r & 0_r & \dots & 0_r \\ I_r & I_r & \dots & 0_r \\ \vdots & \vdots & \ddots & \vdots \\ I_r & I_r & \dots & I_r \end{bmatrix}, \quad c = \begin{bmatrix} I_r \\ I_r \\ \vdots \\ I_r \end{bmatrix} \quad (2.27)$$

Se busca resolver el problema de optimización resultante de minimizar la función objetivo (2.19) sujeta a las restricciones (2.24). Esto da como resultado la secuencia de acciones Δu_k que mejor resuelve el problema de optimización. La primera acción de control Δu_k es la que finalmente es entregada al sistema que se busca controlar.

2.3. Control Predictivo Basado en Modelos (MPC) Explícito

Uno de los principales desafíos del MPC tradicional es la necesidad de resolver un problema de optimización en cada instante de muestreo, lo que puede resultar computacionalmente costoso, especialmente en sistemas de alta velocidad o con recursos limitados.

Para superar este inconveniente, el MPC explícito se propone como una alternativa viable, ya que permite precomputar la ley de control y almacenarla de manera que pue-

da ser evaluada en tiempo real sin necesidad de resolver problemas de optimización online. Esta técnica es particularmente útil para sistemas embebidos con limitaciones de procesamiento y memoria.

En el caso del MPC explícito, el problema de control predictivo se formula como un problema de programación cuadrática paramétrica, donde la solución es una función afín por partes (PWA, por sus siglas en inglés). Esta función se define en una partición del espacio de estados en regiones poligonales, donde cada región tiene asociada una ley de control afín. En otras palabras, la ley de control puede expresarse como:

$$u(x) = K_i x + g_i \quad \text{si } x \in R_i, \quad (2.28)$$

donde K_i y g_i son los parámetros afines correspondientes a la región R_i del espacio de estados. La evaluación de la ley de control explícita se reduce a identificar en qué región R_i se encuentra el estado actual del sistema y aplicar la correspondiente ley de control afín.

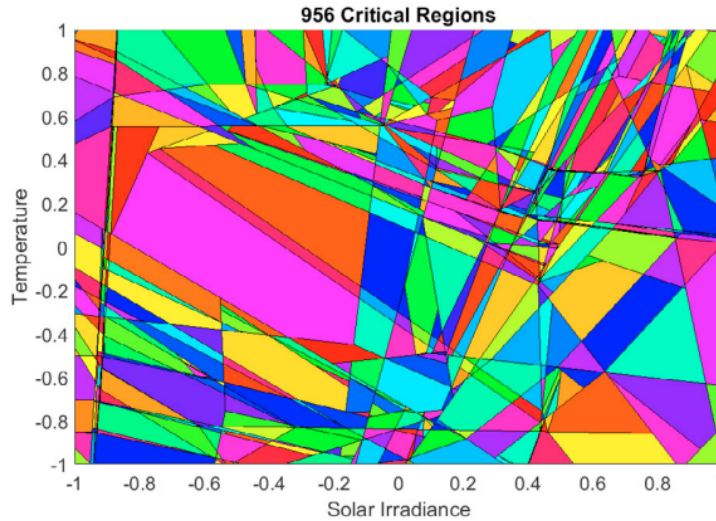


Fig. 2.1: Ejemplo de solución multiparamétrica de un MPC explícito. Fuente [6]

No obstante, el MPC explícito presenta una desventaja clave: el número de regio-

nes R_i crece exponencialmente con el horizonte de predicción y el número de restricciones del sistema. Este crecimiento exponencial puede conllevar grandes requerimientos de almacenamiento y complicar el proceso de ubicación del estado en su región correspondiente. Para mitigar este problema, se han propuesto diversas técnicas de simplificación, como la eliminación de regiones redundantes o la aproximación de la solución exacta mediante particiones simpliciales o redes neuronales.

A pesar de estas limitaciones, el MPC explícito sigue siendo una herramienta poderosa para sistemas donde la velocidad de ejecución es crítica y el problema de optimización debe resolverse en tiempos muy cortos. Además, el uso de métodos de aproximación permite reducir los requisitos de memoria, haciendo posible la implementación en sistemas embebidos con recursos limitados.

3. OBSERVADORES

En los sistemas de control automático, uno de los mayores retos es la obtención precisa y confiable de todas las variables de estado de un sistema. En muchas aplicaciones, no es posible medir directamente todas las variables debido a restricciones de costo, limitaciones técnicas de los sensores o accesibilidad física. En estos casos, es necesario diseñar un mecanismo que permita estimar esas variables no medidas. Este mecanismo se conoce como *observador*.

Un observador es un sistema dinámico que puede implementarse en una estructura de software o hardware, que se utiliza para reconstruir las variables de estado internas de un sistema utilizando las señales de entrada y salida disponibles, junto con un modelo matemático del sistema. El uso de observadores es común en sistemas de control en tiempo real, donde se necesita información confiable de las variables internas del sistema para optimizar su comportamiento.

3.1. Modelado del Sistema

En el contexto de los observadores, el punto de partida es el modelo dinámico del sistema. En general, los sistemas se pueden representar por modelos de estado en la forma:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{3.1}$$

$$y(t) = Cx(t) \tag{3.2}$$

Donde:

- $x(t)$ es el vector de estado del sistema, que incluye las variables internas que deseamos estimar.
- $u(t)$ es el vector de entradas, que representan las señales de control que afectan el comportamiento del sistema.
- $y(t)$ es el vector de salidas, que son las mediciones disponibles.
- A , B y C son matrices que describen la dinámica del sistema.

El objetivo del observador es estimar el vector $x(t)$ basándose en $u(t)$ e $y(t)$. [7]

3.2. Observador de Luenberger

Uno de los observadores más conocidos es el *observador de Luenberger*, el cual se basa en un modelo del sistema y ajusta las estimaciones utilizando un error de observación. Este observador tiene la siguiente estructura:

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L(y(t) - C\hat{x}(t)) \quad (3.3)$$

Donde:

- $\hat{x}(t)$ es el vector de estado estimado.
- L es la *ganancia del observador*, que se ajusta para hacer que el error de estimación $e(t) = x(t) - \hat{x}(t)$ tienda a cero a medida que pasa el tiempo.

El término $L(y(t) - C\hat{x}(t))$ se conoce como la *retroalimentación del error de observación*, que corrige la estimación del estado cuando la diferencia entre la salida medida y la salida estimada $y(t) - C\hat{x}(t)$ no es cero.

3.2.1. Estabilidad del Observador

Para que el observador sea estable, las ganancias del observador L deben elegirse de manera que el error de estimación converja a cero. Esto depende de las propiedades de los autovalores de la matriz de error del observador:

$$A_{obs} = A - LC \quad (3.4)$$

Si los autovalores de A_{obs} tienen parte real negativa, entonces el error $e(t)$ convergerá a cero con el tiempo, garantizando una buena estimación del estado.

3.3. Observadores de Orden Completo y de Orden Reducido

3.3.1. Observadores de Orden Completo

El observador de Luenberger descrito anteriormente es un ejemplo de un *observador de orden completo*, que estima todos los estados del sistema. En este caso, el número de estados estimados es igual al número de estados reales del sistema.

3.3.2. Observadores de Orden Reducido

En algunas aplicaciones, no es necesario estimar todas las variables de estado, sino solo algunas de ellas. En estos casos, se puede utilizar un *observador de orden reducido*, que solo estima las variables no medidas. Esto reduce la complejidad computacional y es útil cuando ciertas variables de estado ya son accesibles mediante mediciones directas.

3.4. Observadores de Kalman

Cuando los sistemas lineales están sujetos a ruido Gaussiano tanto en las entradas como en las mediciones, el *filtro de Kalman* es la herramienta óptima para estimar los estados. Este observador, basado en técnicas de estimación bayesiana, proporciona

una estimación óptima en presencia de incertidumbre. El filtro de Kalman minimiza el error cuadrático medio de la estimación de estado, teniendo en cuenta el ruido en el proceso y en las mediciones.

El filtro de Kalman se define mediante dos fases:

- **Predicción:** Se utiliza el modelo del sistema para predecir el estado en el siguiente paso de tiempo.
- **Actualización:** Se ajusta la predicción basada en la medición disponible, ponderada por la incertidumbre en el modelo y en la medición.

Esto lo hace especialmente útil en aplicaciones donde hay ruido de sensor o variaciones no modeladas en el sistema, como en la robótica, vehículos autónomos y sistemas de navegación.

3.5. *Aplicaciones de los Observadores*

Los observadores se utilizan en una amplia gama de aplicaciones de control, tales como:

- **Sistemas de control de procesos industriales:** Donde es difícil medir variables internas debido a la naturaleza del entorno (temperatura, presión, flujo, etc.).
- **Automóviles y aeronaves:** Para estimar variables críticas como la velocidad y la posición, donde los sensores directos son insuficientes o costosos.
- **Robótica:** Para estimar la posición y orientación de robots cuando no es posible medir todos los grados de libertad directamente.
- **Sistemas de energía:** Para la monitorización y control de plantas de energía, donde no todas las variables son accesibles en tiempo real.

4. REDES NEURONALES

Las Redes Neuronales Artificiales (RNA o NN por sus siglas en inglés) son algoritmos que intentan imitar y simular el cerebro humano con el objetivo de “aprender” en base a grandes cantidades de datos. Mientras que una red neuronal biológica cuenta con axiones, dendritas, conexiones sinápticas y las mismas neuronas, una RNA está conformada por agrupaciones de “neuronas artificiales”, donde cada una realiza cálculos muy simples.

Una RNA normalmente está compuesta por tres partes: una capa de entrada, encargada de recoger la información; una o más capas ocultas, encargadas del procesamiento; y una capa de salida encargada de entregar el resultado. Cabe destacar que si la RNA cuenta con solo una capa oculta (o unas pocas) es conocida como una Red Neuronal Llana (Figura 8.1) o poco profunda, mientras que si tiene tres o más pasa a ser conocida como una “Red Neuronal Profunda” (“*Deep Neural Network*” en inglés) [1].

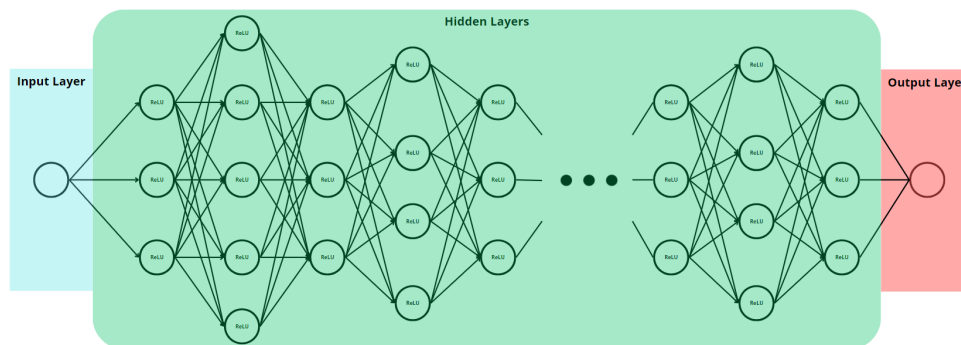


Fig. 4.1: Ejemplo de una red neuronal profunda

Las neuronas en estas RNA operan recogiendo señales de entrada, las cuales pue-

den ser datos numéricos o salidas de otras neuronas. Estas señales son multiplicadas por su peso asociado, para sumarse y ser procesadas mediante una señal de activación, la cual está encargada de determinar si la señal es transmitida o no (es decir, si la neurona se activa o no). En la figura 4.2 se muestra una representación de una neurona (o perceptrón).

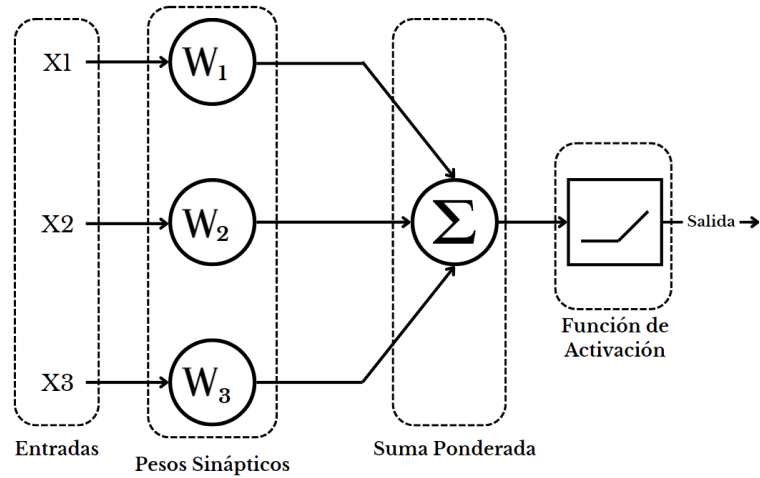


Fig. 4.2: Diagrama de un perceptrón

Las funciones de activación ReLU (Rectified Linear Unit) generan una salida igual a 0 si la entrada es negativa, y una salida igual a la entrada cuando esta sea positiva. Lo anterior se puede expresar matemáticamente como:

$$ReLU(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases} \quad (4.1)$$

y se muestra su representación gráfica en la figura 4.3.

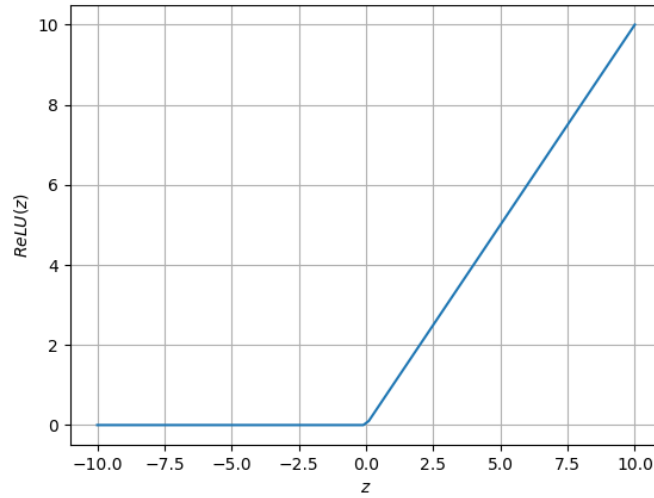


Fig. 4.3: Función de Activación ReLU

Este tipo de función de activación permite introducir no linealidad a los modelos de las RNA, y a la vez evita los problemas causados por otras funciones de activación como la tangente hiperbólica y sigmoide. Uno de los problemas asociados a las funciones de activación sigmoide y tangente hiperbólica serían el desvanecimiento del gradiente, el cual se presenta cuando se entrenan redes neuronales con métodos de aprendizaje basados en el gradiente con retropropagación, y se traduce en que los pesos en las capas iniciales apenas se actualizarán durante el proceso de entrenamiento, por lo que esto derivará en un aprendizaje lento o incluso un posible estancamiento del proceso de optimización [8].

Cabe señalar que la función de activación ReLU es una función afín por partes, ya que esta cumple con que sus dos sub dominios pueden ser descritos con la ecuación de la recta $y(x) = mx + b$.

Interconectando neuronas artificiales con funciones de activación ReLU, es posible formar funciones afín por partes mucho más complejas, capaces de delimitar regiones en un plano o hiperplano. Con lo mencionado anteriormente es posible utilizar estas redes neuronales para representar la ley de control de un MPC Explícito que,

como se menciona en 2.3, consiste en varias regiones poligonales que tienen asociadas una actuación. Esta integración entre MPC explícito y redes neuronales ReLU es tratado más en profundidad en [6].

Para una capa cualquiera de una red neuronal feedforward con funciones de activación ReLU, la salida de dicha capa toma la forma de:

$$x^k = \max\{0, W^k x^{k-1} + b^k\} \quad (4.2)$$

donde k es la capa, W^k la matriz de pesos de la capa k , b^k es el vector de sesgos de la capa k , x^{k-1} la salida de la capa anterior y $x^k \in \mathcal{R}^n$ es la salida de la capa actual.[6]

5. PLANTA DE INTERÉS

En este trabajo se emplea un motor de corriente continua (DC) como planta de control, cuyo modelo matemático ha sido ampliado para considerar la acción integral y así mejorar su respuesta frente a perturbaciones o variaciones en la referencia. El motor DC es ampliamente utilizado en aplicaciones de control debido a su simplicidad y facilidad de modelado, siendo su dinámica representativa de sistemas más complejos.

5.1. *Motor DC*

Los motores eléctricos son motores que convierten energía eléctrica en energía mecánica. Los motores de corriente continua (o motores DC por sus siglas en inglés) son, como indica el nombre, motores que son alimentados únicamente por corriente continua.

5.2. *Modelo dinámico del Motor DC*

El torque generado por un motor DC es proporcional a la corriente de armadura y a la fuerza del campo magnético. Asumiendo que la fuerza del campo magnético es constante, entonces el torque del motor es proporcional a la corriente de armadura i por un factor constante K_t . La expresión de torque queda definida en la ecuación 5.1.

$$T = K_t i \tag{5.1}$$

La fuerza contraelectromotriz e (fuerza que se opone al paso de corriente por una inductancia) es proporcional a la velocidad angular del eje por una constante K_e :

$$e = K_e \dot{\theta} \quad (5.2)$$

Utilizando la segunda ley de Newton y la ley de voltajes de Kirchhoff se obtienen las siguientes ecuaciones:

$$J\ddot{\theta} + b\dot{\theta} = K_t i \quad (5.3)$$

$$L \frac{di}{dt} + Ri = V - K_e \dot{\theta} \quad (5.4)$$

Se selecciona como variables de estado la velocidad angular $\dot{\theta}$ y la corriente eléctrica i . Se definen como entrada y salida del sistema el voltaje de armadura V y la velocidad angular respectivamente. Considerando lo anterior, se reescriben las ecuaciones 5.3 y 5.4 como:

$$\ddot{\theta} = \frac{K_t i}{J} - \frac{b\dot{\theta}}{J} \quad (5.5)$$

$$\frac{di}{dt} = \frac{V}{L} - \frac{Ri}{L} - \frac{K_e \dot{\theta}}{L} \quad (5.6)$$

En base a las ecuaciones 5.5 y 5.6 se define el espacio de estados en tiempo continuo:

$$\frac{d}{dt} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} = \overbrace{\begin{bmatrix} -b/J & K_t/J \\ -K_e/L & -R/L \end{bmatrix}}^{A_c} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} + \overbrace{\begin{bmatrix} 0 \\ 1/L \end{bmatrix}}^{B_c} V \quad (5.7)$$

$$y = \overbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}^{C_c} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} \quad (5.8)$$

En la sección 7.2 se muestra la versión en tiempo discreto del espacio de estados en tiempo continuo antes presentado.

6. OBSERVADOR MODELO EXTENDIDO

Para implementar un controlador predictivo basado en modelo es necesario conocer el estado actual de la planta. Esto no siempre es posible debido a varias razones. Una de ellas sería que las variable física definida como estado del modelo de espacio de estados de la planta no puede ser medida directamente debido a, por ejemplo, la falta de un sensor o un alto costo asociado a la medición de dicha variable.

A la hora de implementar el MPC con acción integral es necesario obtener el estado extendido $\tilde{x}_k$ definido en las ecuaciones 2.10 y 2.11.

Para lograr esto que se diseña un observador de Luenberger, cuya forma fue expuesta en la ecuación 3.3. Basándose en las ecuaciones 2.12 y 2.13 se define el observador de Luenberger:

$$\hat{x}_{k+1} = \tilde{A}\hat{x}_k + \tilde{B}\Delta u_k + K \left[(y_k - r) - \tilde{C}\hat{x}_k \right] \quad (6.1)$$

Que puede ser reescrito en espacio de estados como:

$$\hat{x}_{k+1} = \left[\tilde{A} - K\tilde{C} \right] \hat{x}_k + \left[\tilde{B} \quad K \right] \begin{bmatrix} \Delta u_k \\ y_k - r \end{bmatrix} \quad (6.2)$$

$$\overbrace{\hat{y}_k}^{\tilde{x}_k} = \left[I_{(m+n) \times (m+n)} \right] \hat{x}_k \quad (6.3)$$

donde K es la matriz de ganancias óptimas obtenida al calcular el regulador cuadrático lineal discreto de los modelos en espacio de estado de las ecuaciones (2.12) y (2.13).

7. IMPLEMENTACIÓN DEL CONTROL PREDICTIVO BASADO EN MODELO

Es importante mencionar que es posible aproximar la ley de control de un MPC explícito con redes neuronales. Esto se basa en que, para sistemas lineales con funciones objetivo cuadráticas, la ley de control óptima es afín por partes en politopos [9]. Esto quiere decir que la solución de la red neuronal ReLU entrenada con datos provenientes de un MPC de sistemas lineales y función de costos cuadrática coincide con la solución del MPC explícito del mismo sistema, siempre y cuando la cantidad de regiones que logra definir la red neuronal coincida con las regiones del MPC explícito. En el caso de que la red neuronal no logre definir la misma cantidad de regiones que el MPC explícito, estaríamos frente a una aproximación de este [10].

En este capítulo se describe cómo se realizó el proceso de implementación del MPC. Esto fue realizado utilizando el software matemático MATLAB, aprovechando la disponibilidad de librerías de gran apoyo para el diseño de distintos tipos de controladores, redes neuronales y procesamiento de datos, que en conjunto con el entorno de programación visual Simulink incorporado en este software, facilita la evaluación visual del desempeño del trabajo realizado.

Es por lo anterior que en este capítulo se comenzará explicando las herramientas que ofrece este software, las librerías más importantes utilizadas y los bloques de Simulink utilizados para realizar la simulación.

7.1. Matlab/Simulink

MATLAB (Matrix Laboratory) es un entorno de programación y cálculo numérico ampliamente utilizado en ingeniería, ciencia, matemáticas aplicadas, entre otras áreas. Su lenguaje está optimizado para la manipulación de matrices, la implementación de algoritmos y la visualización de datos, lo que lo convierte en una herramienta muy versátil para el análisis y simulación de sistemas.

Entre las características principales de MATLAB se destacan:

- **Lenguaje de alto nivel:** MATLAB ofrece un lenguaje sencillo y expresivo que permite realizar cálculos complejos con relativamente pocas líneas de código.
- **Manipulación de matrices:** Como su nombre lo indica, MATLAB está diseñado para trabajar con matrices de forma nativa, lo cual es ideal para operaciones algebraicas y de álgebra lineal.
- **Visualización de datos:** MATLAB incluye potentes herramientas de gráficos en 2D y 3D que permiten una rápida interpretación visual de los resultados de simulaciones y cálculos.
- **Funciones matemáticas avanzadas:** MATLAB tiene una amplia colección de funciones matemáticas que abarcan desde el cálculo numérico básico hasta la optimización, la integración y la simulación numérica.

Por otro lado, Simulink es una plataforma gráfica integrada con MATLAB, diseñada específicamente para la simulación de sistemas dinámicos. A través de su interfaz gráfica, Simulink permite modelar sistemas complejos mediante la interconexión de bloques funcionales, facilitando la representación de sistemas físicos, de control y de comunicación.

Entre las principales características de Simulink se incluyen:

- **Modelado basado en bloques:** Simulink permite modelar sistemas mediante el uso de bloques que representan funciones matemáticas, físicas o lógicas. Los

bloques se conectan gráficamente para representar las interacciones entre los componentes del sistema.

- **Simulación de sistemas dinámicos:** Simulink permite simular sistemas continuos, discretos o híbridos, y es ideal para estudiar el comportamiento dinámico de sistemas en tiempo real.
- **Entorno interactivo:** Ofrece una interfaz visual intuitiva que permite ajustar los parámetros del modelo en tiempo real mientras se ejecuta la simulación, brindando una experiencia de simulación altamente interactiva.
- **Integración con MATLAB:** Simulink está completamente integrado con MATLAB, lo que permite ejecutar scripts de MATLAB para automatizar simulaciones, realizar análisis de datos post-simulación y visualizar resultados.
- **Bibliotecas especializadas:** Simulink incluye bibliotecas para distintas áreas como control automático, procesamiento de señales, electrónica de potencia y robótica. Además, los usuarios pueden crear y personalizar sus propios bloques.

7.1.1. Librerías importantes en esta memoria

Entre las librerías que más aportaron al desarrollo de esta memoria se encuentran:

- **Control System Toolbox:** Proporciona algoritmos y aplicaciones varias para diseñar, analizar y ajustar sistemas de control lineal. Esta Toolbox permite especificar sistemas como funciones de transferencia o su modelo de espacio de estados y visualizar la respuesta a escalón de estos individualmente o en lazo cerrado con un controlador diseñado.
- **Deep Learning Toolbox:** Ofrece funciones, apps y bloques de Simulink para diseñar, simular e implementar redes neuronales. Esta toolbox permite visualizar, entrenar y probar las redes neuronales diseñadas de una forma más directa y amigable con el usuario que otras aplicaciones similares.

- **Optimization Toolbox:** Proporciona funciones para buscar parámetros que resuelvan problemas de optimización. Esta toolbox incluye solvers para problemas de programación lineal, cuadrática, lineal entera mixta y no lineal. El solver de programación cuadrática es extremadamente útil a la hora implementar el MPC.

7.1.2. Bloques de Simulink útiles para esta memoria

A continuación se señalarán y describirán los bloques que fueron de utilidad, tanto para la implementación del MPC con acción integral como del controlador basado en redes neuronales

- **Matlab Function:** Permite definir e incorporar una función de Matlab en el entorno de Simulink, facilitando que las señales que este bloque reciba como entradas sean utilizadas para generar una salida.

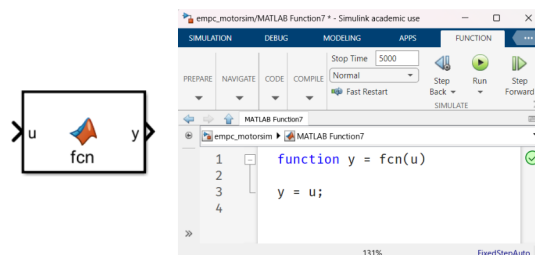


Fig. 7.1: Bloque Matlab Function en Simulink

- **Discrete State Space:** Sirve para definir un modelo en espacio de estados en tiempo discreto, especificando sus matrices A , B , C y D , estados iniciales y periodo de muestreo.

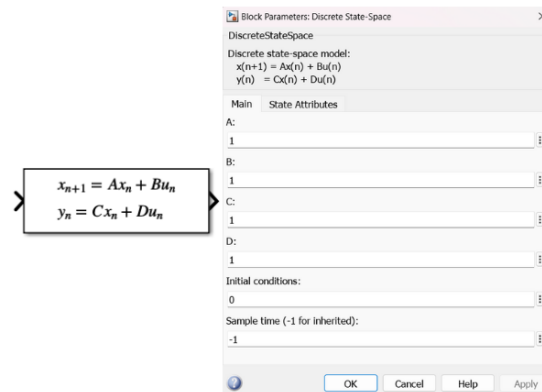


Fig. 7.2: Bloque Discrete State Space en Simulink

- **Discrete Transfer Function:** Posibilita definir funciones de transferencia en tiempo discreto. Será útil para definir integradores y derivadores en tiempo discreto.

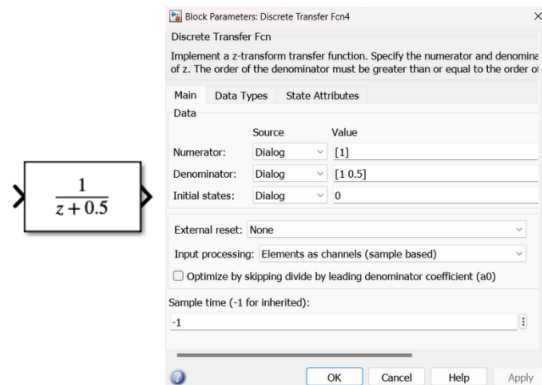


Fig. 7.3: Bloque Discrete Transfer Function en Simulink

- **Zero-Order Hold:** Retiene su entrada durante el periodo de muestreo que se especifica.

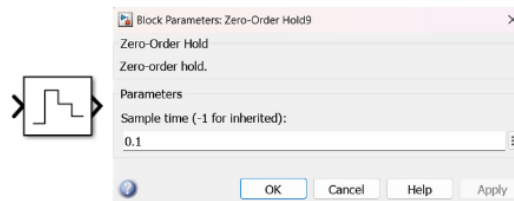


Fig. 7.4: Bloque Zero-Order Hold en Simulink

- **Constant:** Genera una señal real o compleja constante. También es posible especificarle una matriz, por lo que es útil para ingresar variables calculadas en un código de Matlab y se encuentren en el workspace al entorno Simulink.

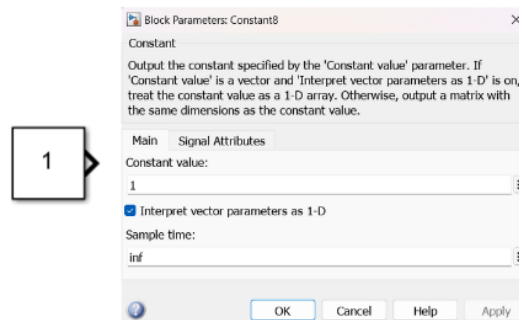


Fig. 7.5: Bloque Constant en Simulink

- **Sum:** Realiza la adición o sustracción de sus entradas. Este bloque permite operar entradas escalares, vectoriales o matriciales.

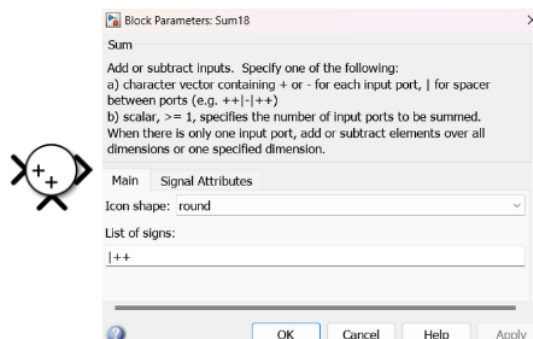


Fig. 7.6: Bloque Sum en Simulink

- **Mux:** Permite juntar datos de un vector. Utilizado para empaquetar los datos antes de que pasen por el observador.



Fig. 7.7: Bloque Mux en Simulink

- **To Workspace:** Registra los datos conectados a su puerto de entrada en el workspace. Esto permite utilizar datos generados en Simulink en un código de Matlab, lo que posibilita el uso de estos datos en alguna librería. Con esto es posible generar datos de entrenamiento para la red neuronal.

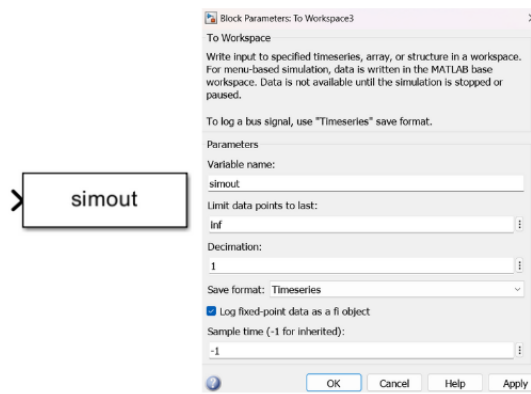


Fig. 7.8: Bloque To Workspace en Simulink

7.2. Implementación del MPC en Simulink

Se comienza designando los valores de las constantes del modelo dinámico del motor DC definido en la sección 5.2 en base a valores descritos en [11]:

Constante	Valor	Unidades
K_e	0.01	$V/rad/s$
K_t	0.01	$N \cdot m/A$
J	0.01	$kg \cdot m^2$
L	0.5	H
R	1	Ω
b	0.1	$N \cdot m \cdot s$

Tab. 7.1: Constantes motor DC

Estos valores son reemplazados en las matrices A_c , B_c y C_c del modelo de espacio de estados en tiempo continuo del motor de las ecuaciones 5.7 y 5.8. Dichas matrices sirven para definir un espacio de estados en MATLAB con el comando $ss(A,B,C,D)$. Este comando considera que un modelo en espacio de estados está definido como:

$$\dot{x} = Ax + Bu \quad (7.1)$$

$$y = Cx + Du \quad (7.2)$$

Como el modelo del motor DC definido no tiene paso directo de la entrada a la salida, la matriz $D = 0$.

Otro comando útil de MATLAB es $c2d(sysc,Ts)$, el cual toma como argumentos un sistema en tiempo continuo, que puede ser definido en espacio de estados, y un tiempo de muestreo Ts , que representa el periodo de muestreo del sistema de tiempo discreto resultante. Este comando devuelve el sistema ingresado en tiempo discreto, con el periodo de muestreo definido. Tener el sistema a controlar de esta forma será necesario a la hora de aplicar la técnica de control predictivo.

El modelo en espacio de estados en tiempo discreto del motor DC queda definido

de la siguiente forma:

$$\overbrace{\begin{bmatrix} \dot{\theta}[k+1] \\ i[k+1] \end{bmatrix}}^{x_{k+1}} = \overbrace{\begin{bmatrix} 0.3678 & 0.05635 \\ -0.001127 & 0.8187 \end{bmatrix}}^{A_d} \overbrace{\begin{bmatrix} \dot{\theta}[k] \\ i[k] \end{bmatrix}}^{x_k} + \overbrace{\begin{bmatrix} 0.006856 \\ 0.1813 \end{bmatrix}}^{B_d} \overbrace{V[k]}^{u_k} \quad (7.3)$$

$$y = \overbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}^{C_d} \overbrace{\begin{bmatrix} \dot{\theta}[k] \\ i[k] \end{bmatrix}}^{x_k} \quad (7.4)$$

Como fue mencionado en la sección 2.1, los algoritmos estándar de Control predictivo no necesariamente logran la acción integral, por lo que en esta memoria se trabajará con el modelo independiente de las perturbaciones desconocidas señalado en la sección antes mencionada. Para esto es necesario extender el modelo en espacio de estados en tiempo discreto de las ecuaciones 7.3 y 7.4 a la forma de las ecuaciones 2.10 y 2.11. Para esto se utiliza el siguiente código para ampliar las matrices A_d , B_d y C_d :

```

1 nx=size(A_d,1); nu=size(B_d,2); ny=size(C_d,1);
2 At=[A_d,zeros(nx,ny);C_d,eye(ny,ny)];
3 Bt=[B_d;zeros(ny,nu)];
4 Ct=[C_d,eye(ny,ny)];

```

Listing 7.1: Matrices ampliadas

Con las matrices extendidas ya creadas, es posible comenzar a trabajar en el desarrollo de los elementos de la función cuadrática en forma estándar de la ecuación 2.19. En dicha ecuación J_0 es una constante que no depende de Δu_k , por lo que no será utilizada. El cálculo del término cuadrático objetivo H de la ecuación 2.19 fue definido en la ecuación 2.20. Los componentes necesarios para el cálculo de la ecuación anterior son definidos en las ecuaciones 2.16, 2.17 y 2.18. El cálculo de lo anterior puede ser realizado fácilmente en MATLAB por el siguiente código.

```

1 P = 1*(eye(L));
2 Q = 1*(eye(L));
3 %% matriz H_L
4 for i=1:L-1
5     zH = zeros(i,1);
6     for j=1:L-i
7         Hcol = Ct*(At^(j-1))*Bt;
8         zH = vertcat(zH,Hcol);
9     end
10
11     if i==1
12         H_L=zH;
13     else
14         H_L=horzcat(H_L,zH);
15     end
16 end
17 %% Matriz de observabilidad O_L (Constante)
18 for i=1:L
19     if i==1
20         O_L = Ct;
21     else
22         O_Laux = Ct*(At^(i-1));
23         O_L= vertcat(O_L,O_Laux);
24     end
25 end
26
27 %% F_L (Constante)
28 F_L = [O_L*Bt H_L];

```

```

29
30 %% Termino cuadratico H
31 H = F_L' * Q * F_L + P;

```

Listing 7.2: Término cuadrático objetivo

Por otro lado, el término objetivo lineal f_k de la ecuación 2.19 fue definido en la ecuación 2.21. El término p_L de esta ecuación depende del estado del sistema, como se puede observar en su definición (ecuación 2.15), por lo que este término será calculado cada vez que el MPC reciba una nueva entrada.

Se observa que las matrices O_L y F_L pueden ser consideradas constantes, ya que estas no dependen de un valor variable en el tiempo y consisten únicamente en operaciones que relacionan las matrices del modelo en espacio de estados extendido definido. Es por esto que estas matrices son calculadas en MATLAB y luego son ingresadas a Simulink utilizando el bloque *Constant* señalado en la sub-sección 7.1.

Tanto el modelo del motor DC en tiempo discreto especificado en las ecuaciones de espacio de estados 7.3 y 7.4, como el observador extendido de las ecuaciones 6.2 y 6.3 se incorporan a Simulink utilizando el bloque *Discrete State Space* señalado en la sub-sección 7.1. En la Figura 7.9 se presenta la implementación del modelo del motor de corriente continua (DC) en Simulink, mientras que la Figura 7.10 ilustra la implementación del observador extendido correspondiente al sistema en el mismo entorno de simulación.

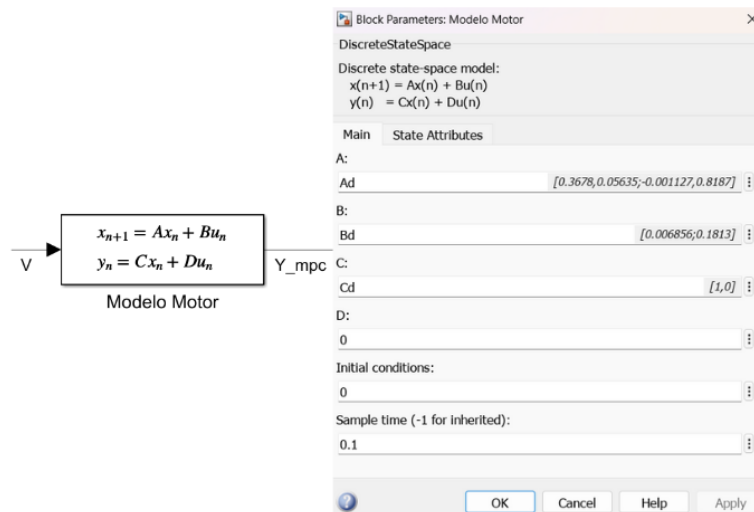


Fig. 7.9: Modelo del Motor DC incorporado a Simulink

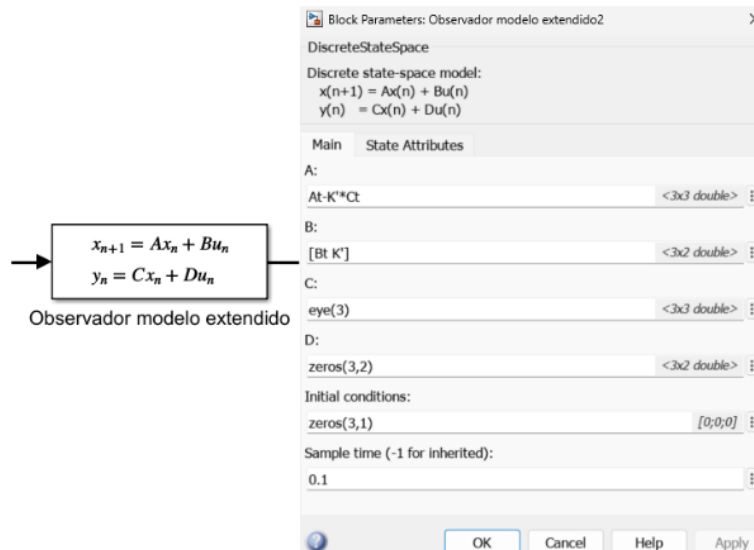


Fig. 7.10: Observador extendido incorporado a Simulink

Por otro lado, el MPC encargado de resolver el problema de optimización fue incorporado al entorno Simulink mediante el uso del bloque *Matlab Function* descrito en la sección 7.1. En la Figura 7.11 se muestra el bloque MPC incorporado al entorno de simulación:

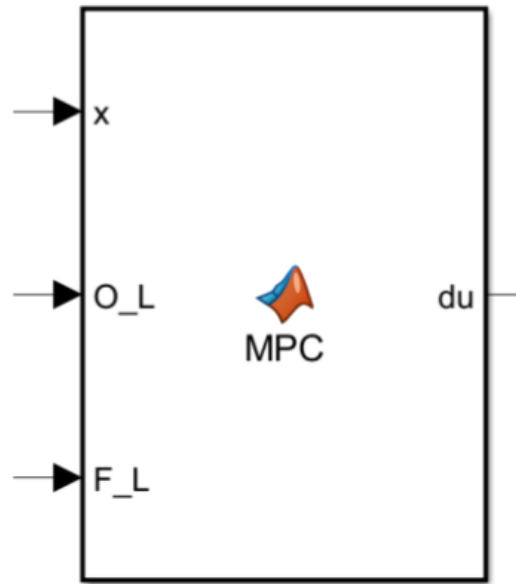


Fig. 7.11: MPC incorporado a Simulink

Dicho bloque recibe como entradas el estado extendido que da como resultado el observador extendido de la Figura 7.10 y las matrices O_L y F_L , que son incorporadas por el bloque *Constant*, y entrega como salida la variación de la acción de control. La conexión descrita se presenta en la Figura 7.12.

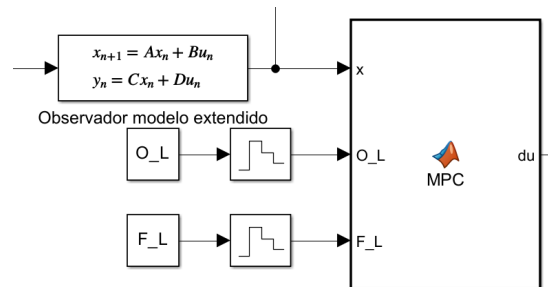


Fig. 7.12: Entradas bloque MPC en Simulink

La función que se encuentra definida dentro del bloque de la Figura 7.11 se presenta en el siguiente código:

```
1 function du = MPC(x, O_L, F_L)
```

```

2   persistent du_prev u_prev
3
4   Np = 10;
5   if isempty(du_prev)
6       du_prev = zeros(Np,1);
7   end
8   if isempty(u_prev)
9       u_prev = 0;
10  end
11  P = 1*(eye(Np));
12  Q = 1*(eye(Np));
13  umax = 120;
14  umin = -120;
15  dumax = 2;
16  dumin = -2;
17  r = 0;
18  A_t = [0.367800000000000,0.056350000000000,0;
19         -0.001127000000000,0.818700000000000,0;
20         1,0,1];
21  p_L = O_L*A_t*x;
22  ref=r*ones(Np,1);
23  H = F_L'*Q*F_L+P;
24  f_k = F_L'*Q*(p_L-ref);
25  options = optimoptions("quadprog","Algorithm","
        active-set");
26
27  I_Np = [eye(Np);-eye(Np)];
28  A_cons = [I_Np;

```

```

29         tril(ones(Np));
30         -tril(ones(Np))];
31     b_cons = [ repmat(dumax,Np,1);
32               repmat(dumin,Np,1);
33               repmat(umax,Np,1) - u_prev;
34               repmat(umin,Np,1) + u_prev];
35
36     dus = quadprog(H,2*f_k,A_cons,b_cons,[],[],[],[],
37                   du_prev,options);
38     du = dus(1);
39     du_prev = dus;
40     u = u_prev+ du;
41     u_prev = u;
41 end

```

Listing 7.3: Bloque Matlab Function MPC

El código antes expuesto define y resuelve el problema de optimización cuya función objetivo fue descrita en la ecuación 2.19 y sus matrices que representan las restricciones de este fueron expuestas en las ecuaciones 2.25 y 2.26 y entrega como resultado una variación de actuación (du).

La salida del bloque MPC sigue dos caminos. Por un lado, esta salida pasa por un integrador en tiempo discreto, incorporado al entorno Simulink mediante el bloque *Discrete Transfer Function* expuesto en la sección 7.1, que tiene como objetivo el transformar dicha variación de actuación en el voltaje que recibe como entrada el modelo del motor en corriente continua. Lo anteriormente dicho se muestra en la Figura 7.13.

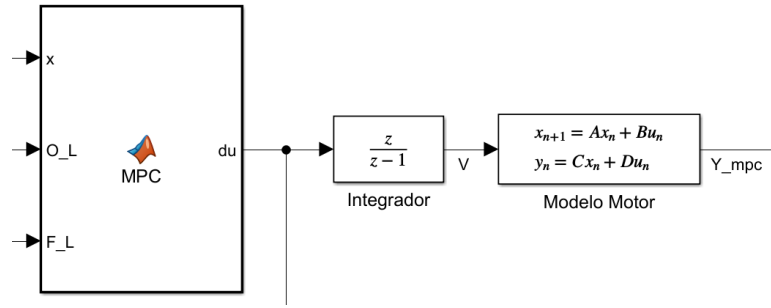


Fig. 7.13: Conexión salida bloque MPC a motor DC

Por otro lado, la señal que entrega del bloque MPC también es realimentada al observador extendido para que, en conjunto con el error entre referencia y la salida del motor DC, este estime el estado que es entregado al MPC como entrada. En la Figura 7.14 se expone como queda finalmente conectado el sistema controlado por el MPC. En el capítulo 9, específicamente en la sección 9.1 de dicho capítulo se muestran los resultados del control realizado por el MPC.

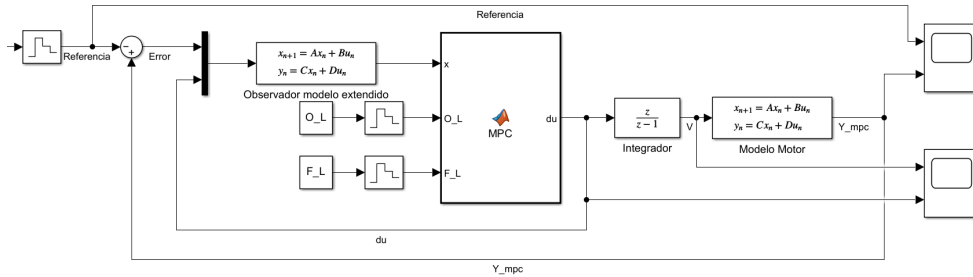


Fig. 7.14: Sistema Motor DC - MPC

7.2.1. Extracción de datos de simulación

La principal razón de simular el controlador MPC era obtener datos que serían utilizados para entrenar una red neuronal que imitara el comportamiento del controlador. Para lograr lo anterior se utilizan dos bloques *To Workspace* descritos en la sección 7.1. Uno es conectado en la entrada x para obtener los estados que generan la salida du , cuyo valor es extraído por otro de estos bloques. Lo descrito anteriormente se

puede ver representado en la Figura 7.15.

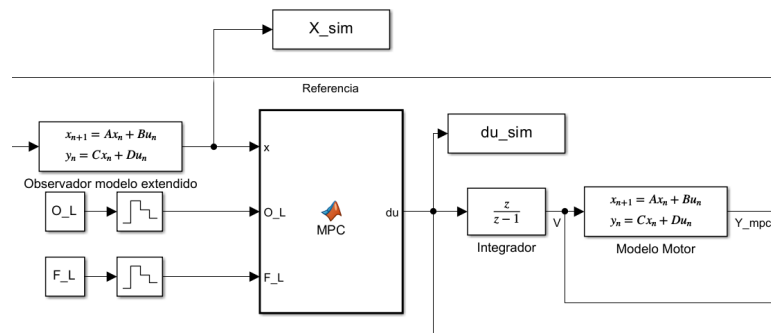


Fig. 7.15: Conexión bloques To Workspace

Para que los datos extraídos sean útiles es necesario que estos abarquen la mayor cantidad de casos posibles. Esto se traduce en que el conjunto de datos debe contener las acciones que toma el MPC para la mayor combinación posible de los estados del sistema que, como se mencionó antes, es la entrada del MPC. Es por esto que se utiliza como referencia de velocidad una señal que cambia su valor periódicamente a otro dentro de un rango definido, que es implementado a Simulink mediante el bloque *Uniform Random Number* expuesto en la Figura 7.16. Por otra parte, en la Figura 7.17 se muestra cómo responde el MPC a dicha señal.

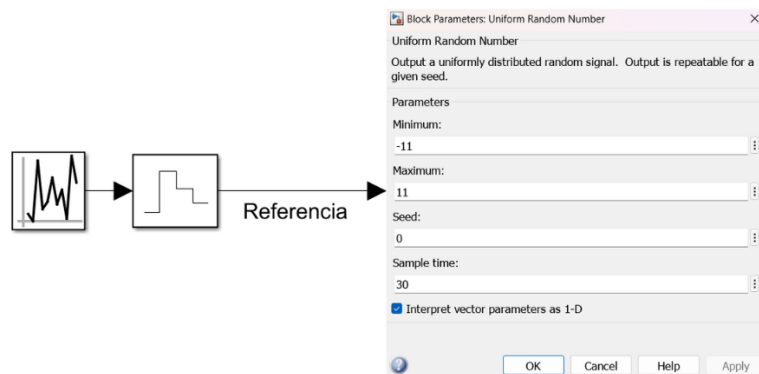


Fig. 7.16: Señal de referencia para extraer datos de entrenamiento

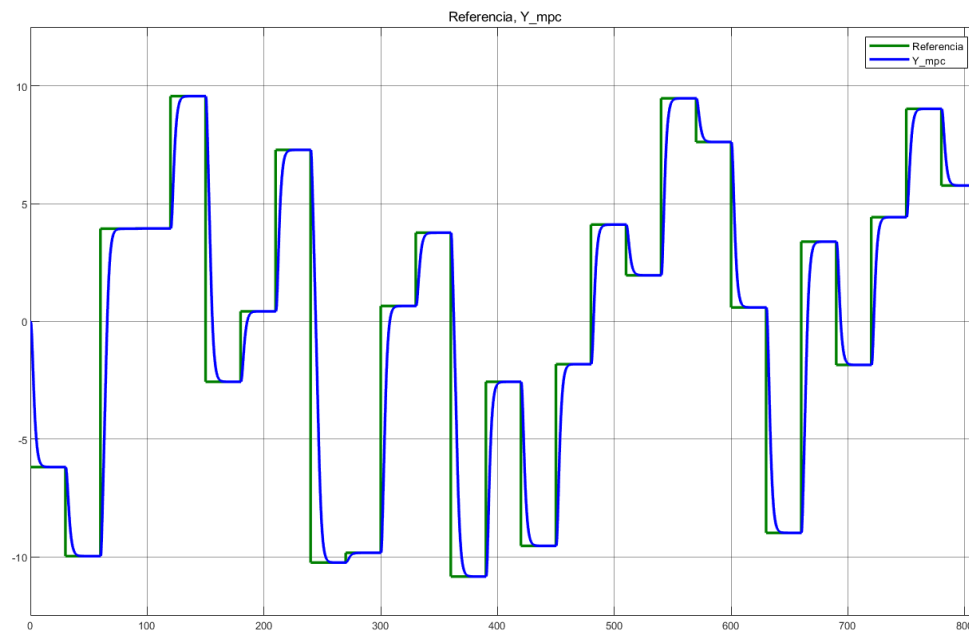


Fig. 7.17: Respuesta MPC a referencia para datos de entrenamiento

Los datos extraídos son utilizados posteriormente en el entrenamiento de la red neuronal, que será revisado en la sección 8.2.

8. IMPLEMENTACIÓN DE LA RED NEURONAL

8.1. *Diseño de arquitectura Red Neuronal*

Como fue expuesto en el Capítulo 2, específicamente en la Sección 2.3, un MPC explícito evalúa en qué región se encuentra el estado actual del sistema y aplica la ley de control explícita asociada a dicha región. El control óptimo puede representarse como una función afín por partes, lo que se traduce en que la acción de control es una función lineal de los estados del sistema dentro de la región correspondiente.

Las redes neuronales, como ya ha sido mencionado anteriormente, son capaces de clasificar datos en distintas categorías [12], por lo que si se considera que una región del MPC explícito es un tipo de categoría, se puede utilizar una red neuronal para identificar en qué región se encuentran los estados del sistema [13], y con esto, aplicar la función lineal correspondiente a dicha región para calcular la acción de control a tomar .

Como se demuestra en [1], cualquier ley de control de un MPC explícito es posible representarla como una red neuronal profunda de cierto tamaño con funciones de activación ReLU. Sin embargo, el año 2023 se probó que toda red neuronal profunda ReLU puede ser reescrita como un equivalente funcional de tres capas ocultas [14], por lo que la arquitectura utilizada en esta memoria será de una red neuronal feed-forward con tres capas ocultas ReLU totalmente conectadas como se muestra en la Figura 8.1.

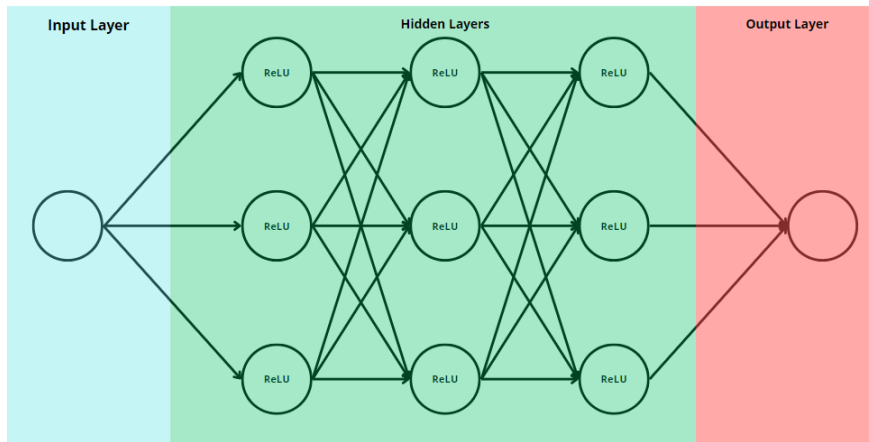


Fig. 8.1: Red Neuronal con tres capas ocultas

Se debe aclarar que las redes neuronales con este tipo de arquitectura imitan el problema de programación cuadrática paramétrica del MPC explícito, no el MPC en si mismo. Esto significa que la implementación en redes neuronales, a diferencia del bloque MPC que se creará, no contará con memoria de estados o variables pasadas.

Cabe destacar que la cantidad de neuronas por capa oculta será modificada durante el proceso de entrenamiento con el objetivo de encontrar la cantidad que mejor cumpla el objetivo buscado.

8.2. Entrenamiento Red Neuronal

Para entrenar la red neuronal definida en la sección anterior, se utilizarán los datos extraídos de la simulación del MPC, cuyo procedimiento fue descrito en la subsección 7.2.1. Para esto se utilizará la herramienta *Regression Learner* de *Deep Learning Toolbox*, la cual permite entrenar modelos utilizando machine learning supervisado. Es posible abrir dicha herramienta ingresando *regressionLearner* en la línea de comandos, lo que abre la ventana que se muestra a continuación:

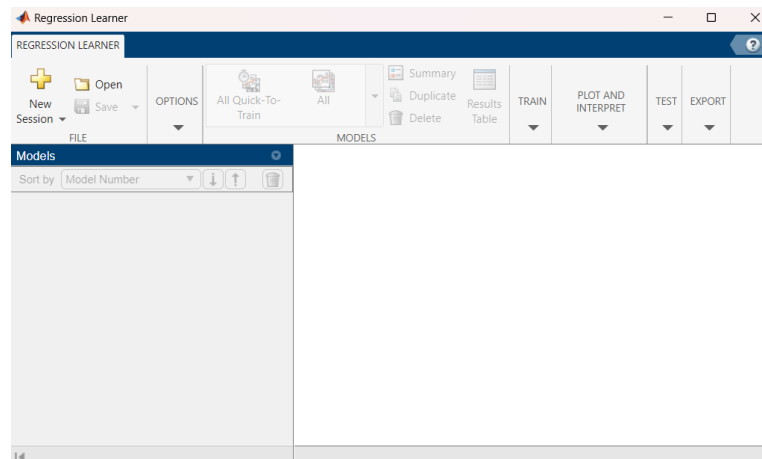


Fig. 8.2: Ventana principal *Regression Learner*

En esta ventana se debe seleccionar “New Session”, lo que permite seleccionar entre si usar datos de entrenamiento provenientes de un archivo o datos que se encuentren en el “Workspace”. En este caso, cómo se extrajeron datos desde Simulink al Workspace, se seleccionará la segunda opción. Esto abrirá la ventana expuesta en la Figura 8.3:

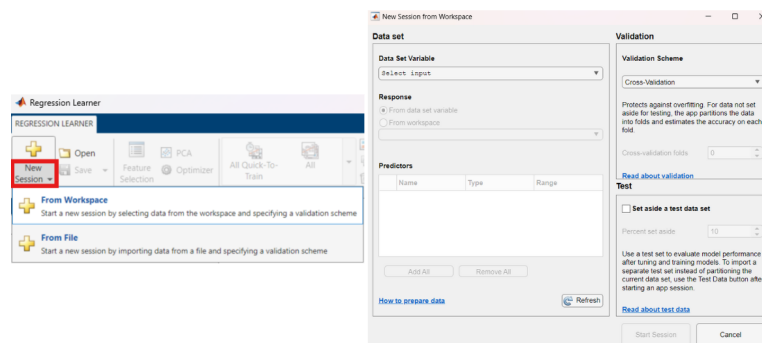


Fig. 8.3: Ventana de selección de datos en *Regression Learner*

En esta ventana se debe seleccionar como “Input” los datos de las entradas del sistema, lo que en este caso serían los estados del modelo extendido, y como respuesta los datos de la salida, lo que sería la actuación u que entrega el MPC. Luego se debe hacer click en el botón de “Start Session” ubicado en la esquina inferior derecha. Esto

cargará los datos en la ventana principal, donde será posible visualizar los datos que utilizaremos para realizar la regresión.

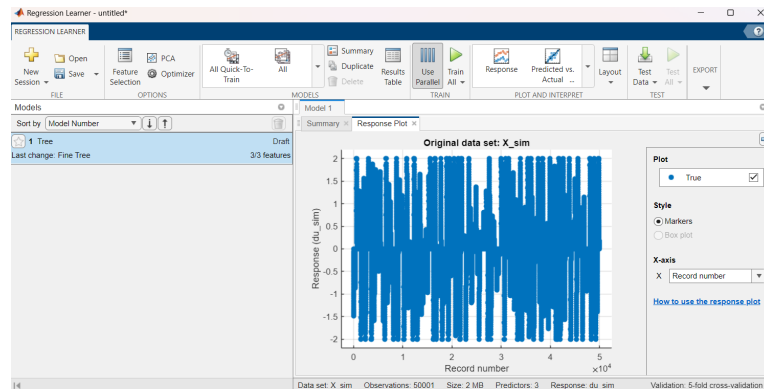


Fig. 8.4: Ventana principal *Regression Learner* con datos seleccionados

Una vez con los datos cargados en la herramienta se define la arquitectura de la red neuronal, lo que se realiza seleccionando la flecha que se encuentra en la sección “MODELS” de la parte superior. Esto expandirá una pestaña en la que se mostrarán distintos modelos de regresión lineal, entre los que se encuentran los que utilizan redes neuronales. En este caso se utiliza la arquitectura descrita en la sección anterior (tres capas ocultas):

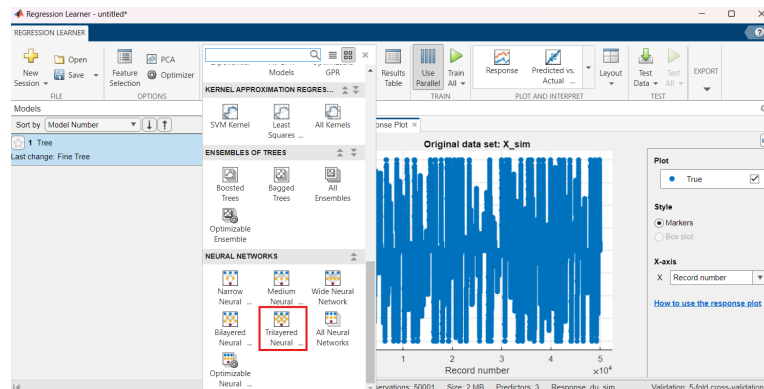


Fig. 8.5: Selección de la arquitectura en *Regression Learner*

Esto agregará dicha arquitectura a la sección de modelos, a la vez que permitirá definir los hiperparámetros del modelo, que serían el número de neuronas por capa, la

función de activación de dichas capas (ReLU) y el número máximo de iteraciones de entrenamiento, como se muestra en la Figura 8.6

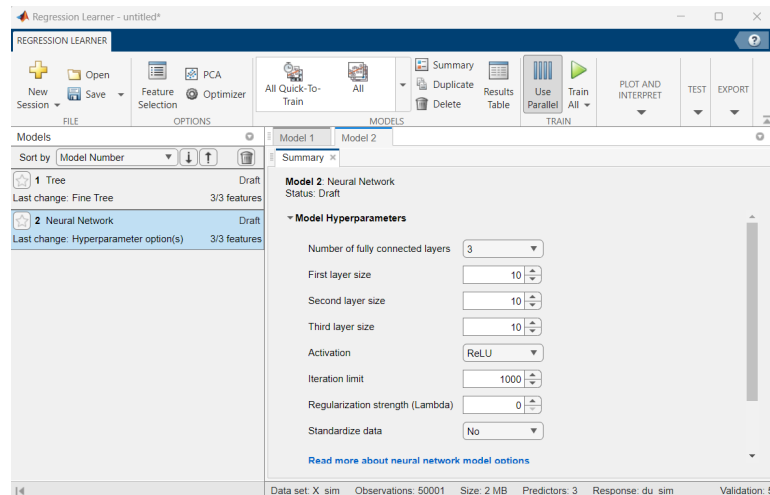


Fig. 8.6: Sección de definición de hiperparámetros del modelo en *Regression Learner*

Repitiendo el proceso antes mencionado es posible definir varias redes con hiperparámetros distintos. Esto sirve para entrenar varios modelos al mismo tiempo y comparar su desempeño en base a su error cuadrático medio. Para entrenar los modelos solo faltaría hacer click en el botón de “Train All”.

El modelo entrenado resultante con mejor resultado se exporta al Workspace de Matlab, donde puede ser utilizado para generar nuevas respuestas en base a distintas entradas. Se verá en más profundidad cómo procesar este modelo para que sea apto de implementar en Simulink en la sección 8.3.

8.3. Implementación en Simulink

Para implementar la red neuronal entrenada en Simulink primero será necesario extraer los pesos (weights) y sesgos (bias) de esta. Con esto en mente se utiliza el siguiente código, el cual se encarga de copiar los datos antes mencionados, que se encuentran encapsulados en un objeto *network*, en matrices de pesos y sesgos correspondientes a cada capa oculta de la red neuronal, para luego guardarlo en un archivo

.mat que será cargado a Simulink para su utilización.

```
1 %% regressionlearner
2 net2= trainedModel.RegressionNeuralNetwork;
3 input_weights = net2.LayerWeights{1};
4 input_bias = net2.LayerBiases{1};
5 output_weights1 = net2.LayerWeights{2};
6 output_bias1 = net2.LayerBiases{2};
7 output_weights2 = net2.LayerWeights{3};
8 output_bias2 = net2.LayerBiases{3};
9 output_weights3 = net2.LayerWeights{4};
10 output_bias3 = net2.LayerBiases{4};
11 save('weight_bias_regression.mat','input_weights',"
      input_bias","output_weights1","output_bias1","
      output_weights2","output_bias2","output_weights3","
      output_bias3");
```

Listing 8.1: Código generador de matrices de pesos y sesgos

En Simulink se utiliza una arquitectura muy similar a la utilizada para implementar el MPC presentada en la sección 7.2, con la diferencia que el bloque Matlab Function del MPC, junto con los bloques constantes O_L y F_L , son reemplazados por un bloque Matlab Function que recibe como entrada los estados del sistema y entrega como salida el aumento o decremento de la acción de control du . El sistema con la red neuronal como controlador se muestra en la Figura 8.7.

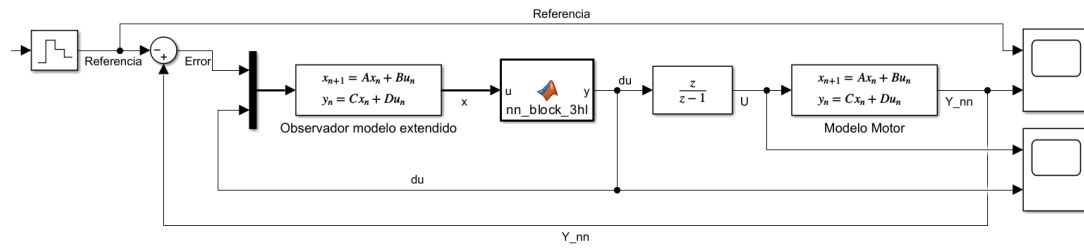


Fig. 8.7: Sistema Motor DC - NN

Al interior del bloque Matlab Function que lleva como nombre *nn_block_3hl* mostrado en la figura anterior se encuentra la siguiente función:

```

1 function y = nn_block_3hl(u)
2     %#codegen
3     persistent net;
4     if isempty(net)
5         net = load('weight_bias_regression.mat');
6     end
7     iw = net.input_weights;
8     ib = net.input_bias;
9     ow1 = net.output_weights1;
10    ob1 = net.output_bias1;
11    ow2 = net.output_weights2;
12    ob2 = net.output_bias2;
13    ow3 = net.output_weights3;
14    ob3 = net.output_bias3;
15    y = ow3*max(ow2*max(ow1*max(iw*u+ib,0)+ob1,0)+ob2,0)
16        + ob3;
17 end

```

Listing 8.2: Bloque Matlab Function NN

La función anterior carga el archivo .mat que contiene los pesos y sesgos sinápticos que fueron obtenidos mediante entrenamiento y los desempaqueta. De esta forma se obtienen las matrices de pesos y sesgos independientes de cada capa de la red neuronal, lo que permite operar dichas matrices en función de la entrada. Se utiliza la función *max* para emular la función de activación ReLU y de esta forma utilizar la expresión matemática de la ecuación 4.2 para calcular la acción de control. Los resultados de simulación de este controlador serán expuestos en el capítulo 9.

9. RESULTADOS DE SIMULACIÓN

En este capítulo se expondrán los resultados obtenidos de simular el lazo de control con el MPC y el lazo de control con la red neuronal. Se mostrará que la implementación del MPC incorpora correctamente las restricciones y que la red neuronal es capaz de imitar dicha característica.

9.1. Resultados MPC

A la hora de las pruebas del bloque MPC se modificaron las restricciones de actuación máxima y mínima (u_{max} y u_{min}) y de razón de cambio máxima y mínima de la actuación (d_{umax} y d_{umin}), para verificar que el sistema cumplía con dichas restricciones.

En la Figura 9.1 se muestra cómo el MPC sigue a la referencia cuando se restringe la entrada de voltaje al motor DC entre $[-120, 120][V]$ y la razón de cambio de dicho voltaje se limita a $\pm 2[V]$ entre cada muestreo, mientras que la Figura 9.2 muestra que dichas señales si cumplen con sus limitantes. Estas son las restricciones usadas en el MPC utilizado para generar los datos de entrenamiento de la red neuronal.

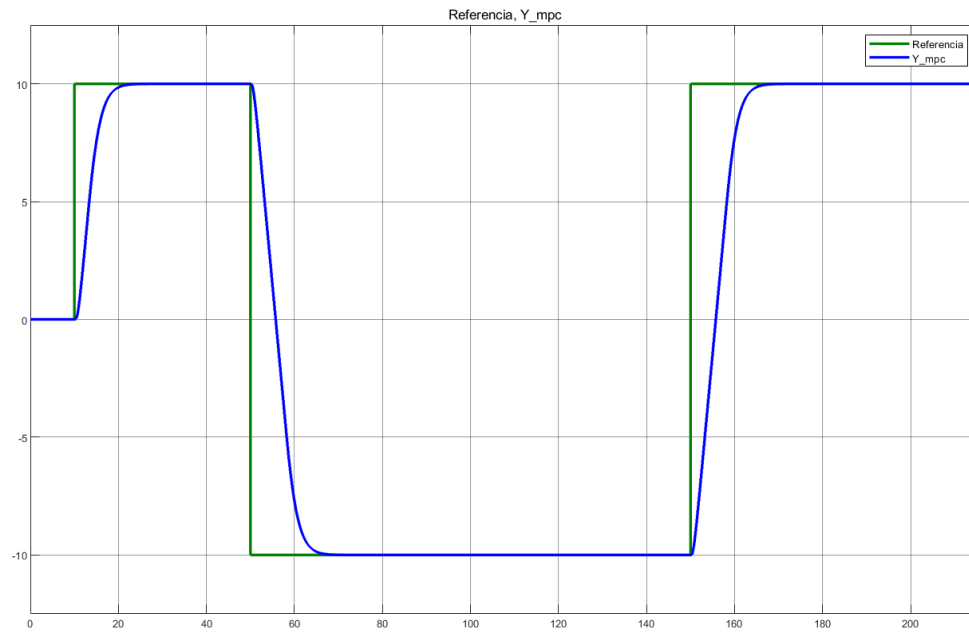


Fig. 9.1: Salida del motor DC con restricciones $u \in [-120, 120]$ y $du \in [-2, 2]$. MPC

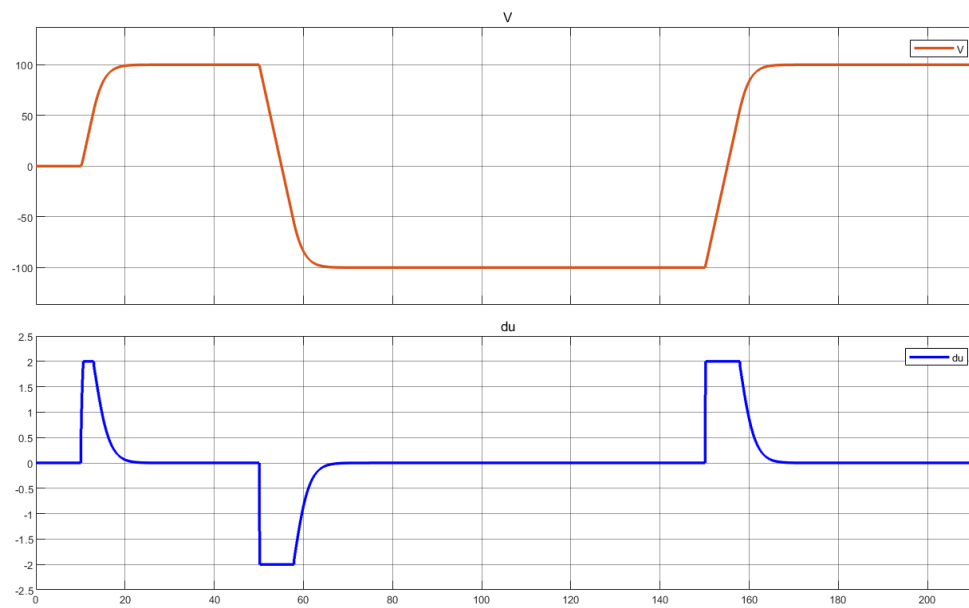


Fig. 9.2: Entrada del motor DC con restricciones $u \in [-120, 120]$ y $du \in [-2, 2]$. MPC

Por otro lado, en las siguientes figuras se exponen la respuesta del MPC al modificar las restricciones pero manteniendo la referencia.

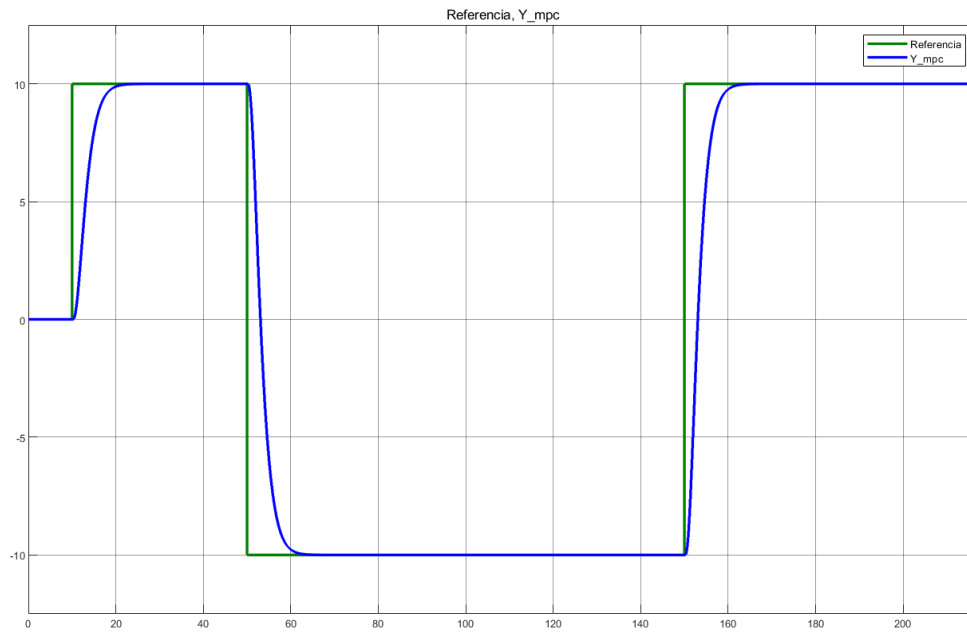


Fig. 9.3: Salida del motor DC con restricciones $u \in [-120, 120]$ y $du \in [-5, 5]$. MPC

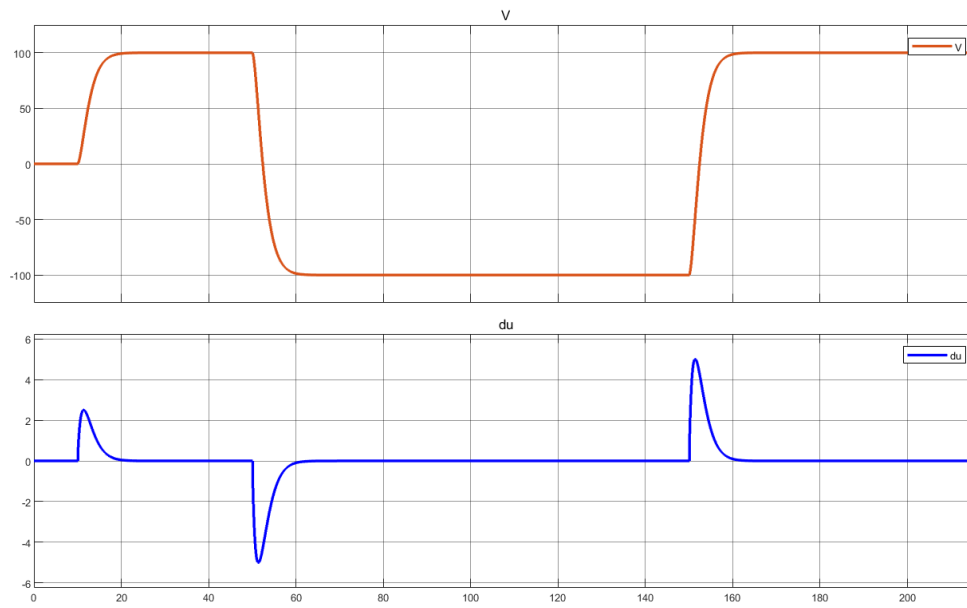


Fig. 9.4: Entrada del motor DC con restricciones $u \in [-120, 120]$ y $du \in [-5, 5]$. MPC

En las Figuras 9.5 y 9.7 se exponen los casos cuando el MPC tiene como restricción de voltaje de $\pm 90[V]$ y se busca seguir una referencia de velocidad de -10

y $10[\text{rad/s}]$. En las Figuras 9.2 y 9.4 se observa que dichas referencias se alcanzan con voltajes de -100 y $100[\text{V}]$ respectivamente, por lo que, debido a la restricción antes mencionada, el controlador no logra seguir la referencia. En las Figuras 9.6 y 9.8 se muestra cómo el voltaje, una vez alcanzada su restricción, deja de aumentar o disminuir.

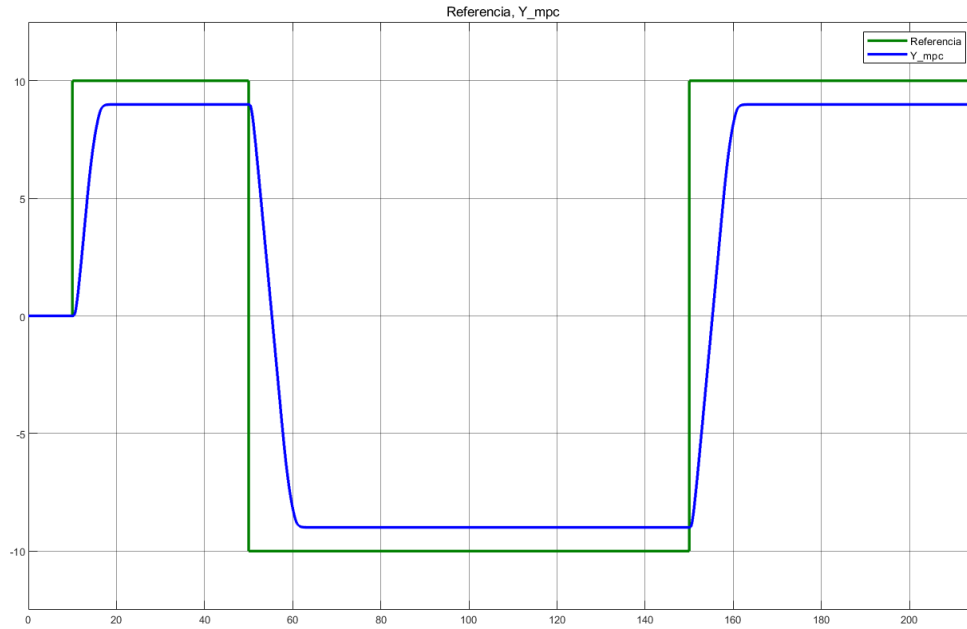


Fig. 9.5: Salida del motor DC con restricciones $u \in [-90, 90]$ y $du \in [-2, 2]$. MPC

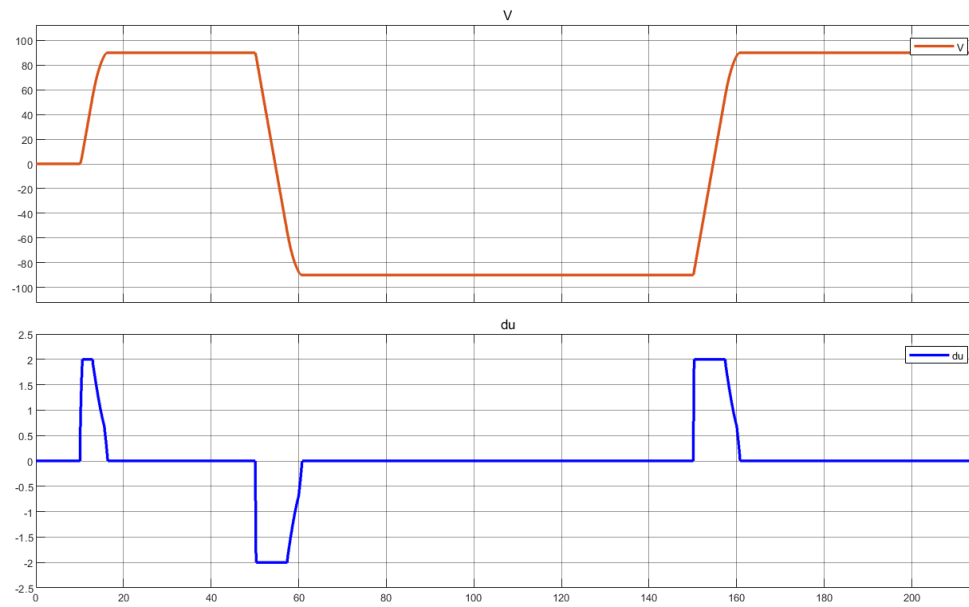


Fig. 9.6: Entrada del motor DC con restricciones $u \in [-90, 90]$ y $du \in [-2, 2]$. MPC

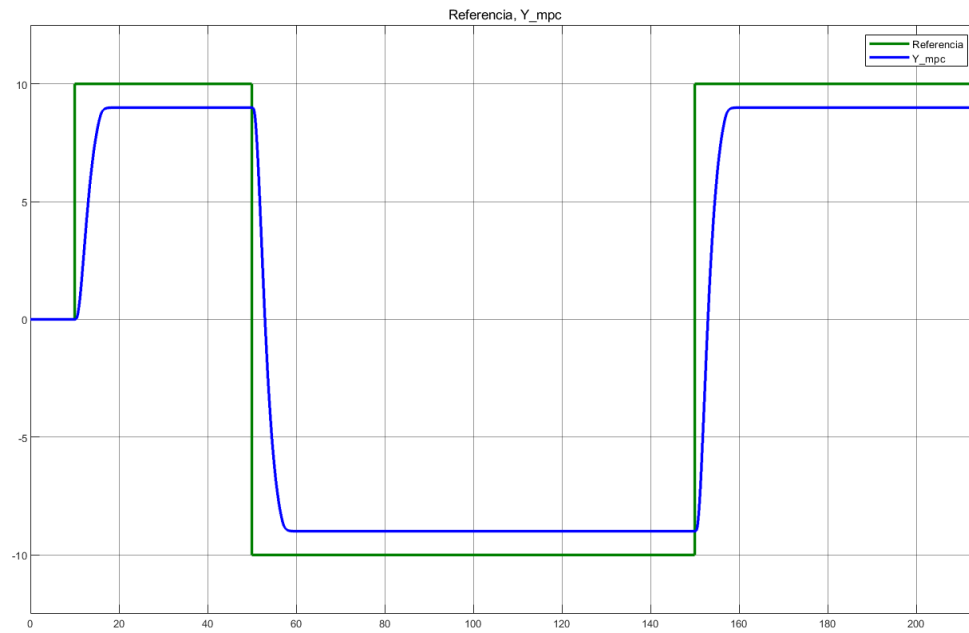


Fig. 9.7: Salida del motor DC con restricciones $u \in [-90, 90]$ y $du \in [-5, 5]$. MPC

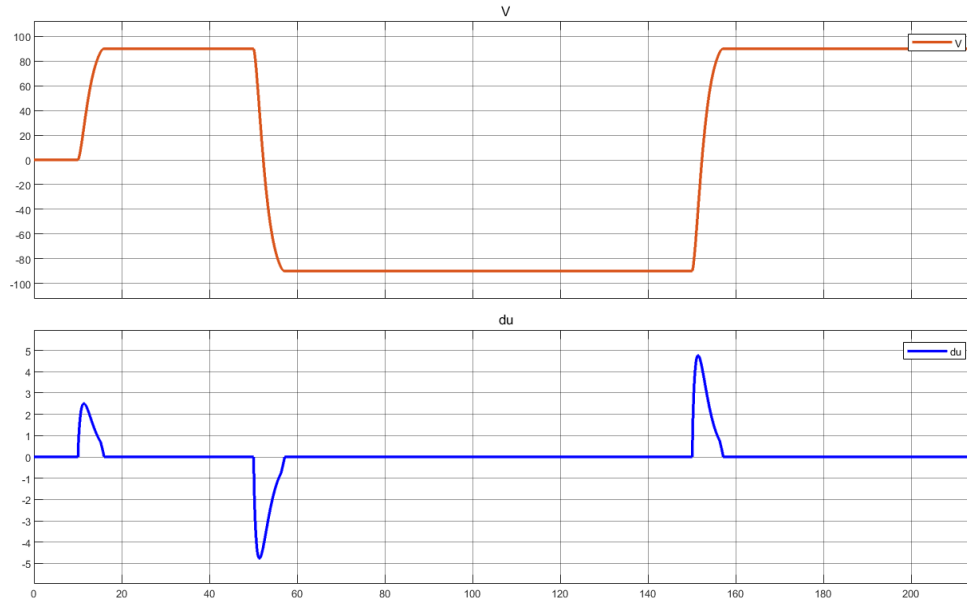


Fig. 9.8: Entrada del motor DC con restricciones $u \in [-90, 90]$ y $du \in [-5, 5]$. MPC

Se nota claramente que en las Figuras 9.5 y 9.7 el controlador no es capaz de seguir la referencia. Uno puede pensar que esto se debe a la falta de acción integral por parte de este, pero la verdad es que este fenómeno se produce debido a que la restricción que se le aplica al voltaje máximo y mínimo que entrega el MPC ($u \in [-90, 90][V]$). Como se muestra en la figura 9.2, el voltaje que necesita el modelo del motor utilizado para llegar a la velocidad de $\pm 10[rad/s]$ que solicita la referencia son $\pm 100[V]$, por lo que, como queda demostrado en la Figura 9.6, el MPC respeta su restricción y no supera el máximo y mínimo voltaje impuesto.

9.2. Demostración de acción integral MPC

Para demostrar que el diseño de este controlador MPC si cuenta con acción integral se incorpora un bloque de banda muerta a la entrada del modelo del motor DC. Esta nueva configuración se muestra en la Figura 9.9:

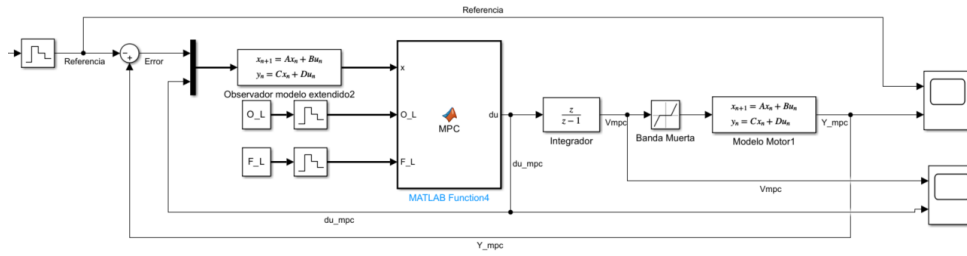


Fig. 9.9: Sistema Motor DC - MPC con banda muerta

El bloque antes mencionado define un rango en torno a un valor (en este caso 0) donde el voltaje de entrada no se trasmite a la salida. Esto emula a lo que pasa en un verdadero Motor DC, donde existe un rango de voltajes que no logran poner en movimiento a este si se encuentra detenido en un inicio.

Si el controlador logra corregir el error en estado estacionario y seguir la referencia perfectamente, entonces quedaría demostrada la existencia de la acción integral. Con este objetivo en mente, se simulará nuevamente el MPC con distintos rangos de banda muerta.

Se comienza utilizando una banda muerta entre $\pm 5[V]$. En la Figura 9.10 se observa el seguimiento de a referencia del MPC, mientras que en la Figura 9.11 se presenta el voltaje de entrada al motor DC y la salida del MPC (du) que, al ser integrado, genera dicho voltaje para el valor de banda muerta dicho.

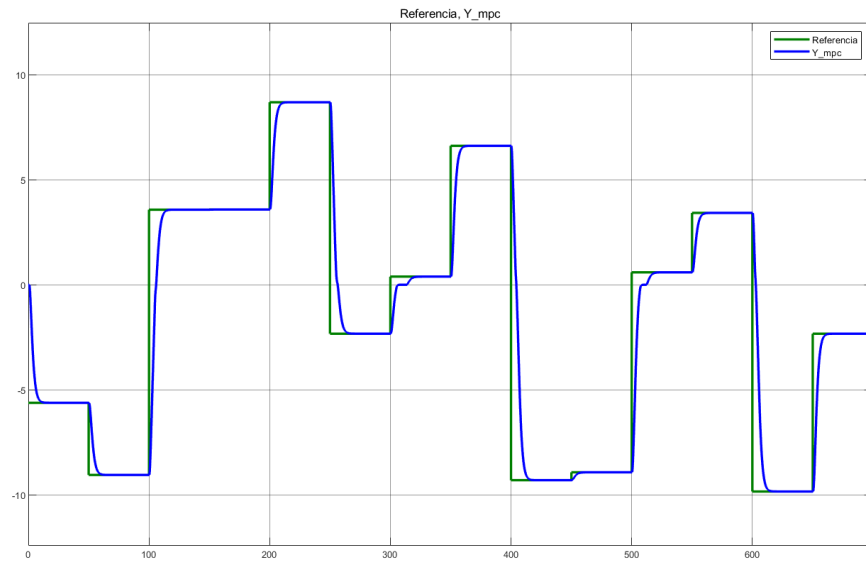


Fig. 9.10: Seguimiento de referencia Motor DC con banda muerta de $\pm 5[V]$

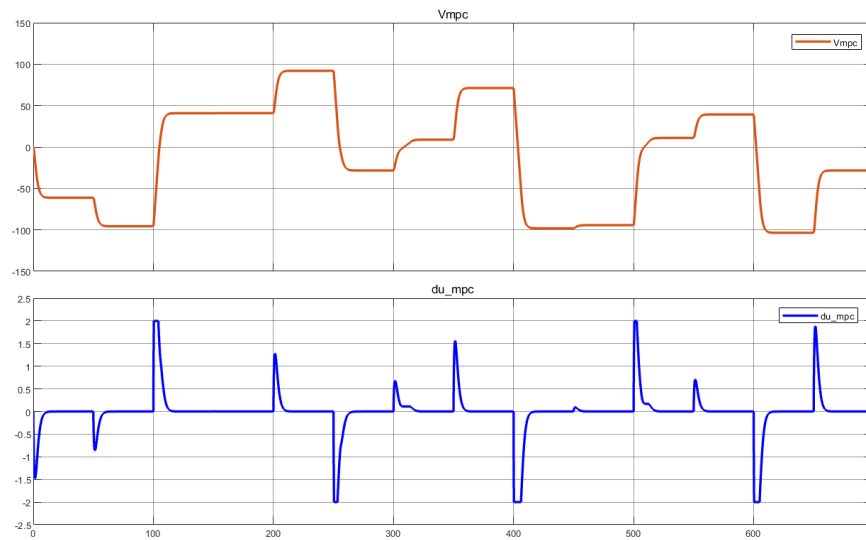


Fig. 9.11: Voltaje de entrada Motor DC y salida del MPC (du) con banda muerta de $\pm 5[V]$

Observando las dos figuras recién presentadas queda claramente expuesto que el controlador si supera la banda muerta y logra seguir a la referencia en estado estacionario. Esto demuestra que esta implementación del MPC cuenta con acción integral.

De igual manera, en las siguientes figuras se presentan dos casos más con bandas muertas más amplias. Las Figuras 9.12 y 9.13 muestran el efecto de una banda muerta de $\pm 10[V]$, mientras que el comportamiento del MPC frente a una banda muerta aún más amplia ($\pm 20[V]$) queda expuesto en las Figuras 9.14 y 9.15.

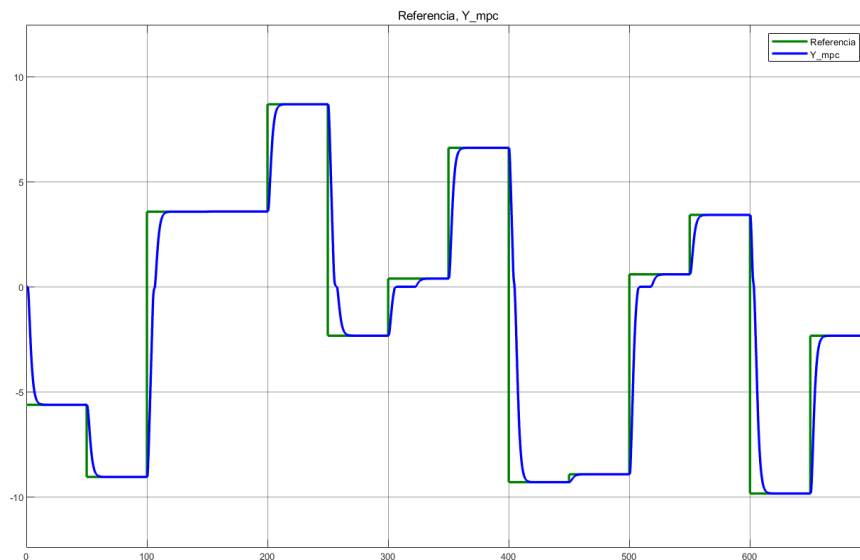


Fig. 9.12: Seguimiento de referencia Motor DC con banda muerta de $\pm 10[V]$

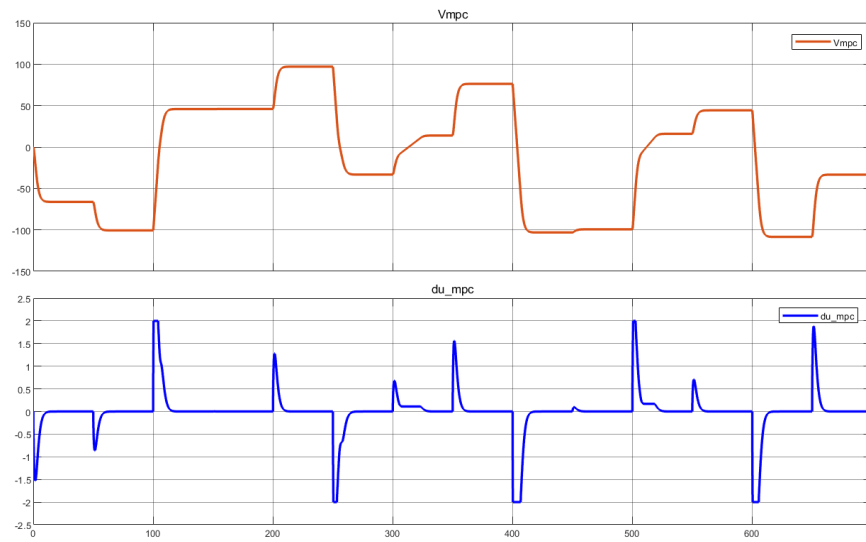


Fig. 9.13: Voltaje de entrada Motor DC y salida del MPC (du) con banda muerta de $\pm 10[V]$

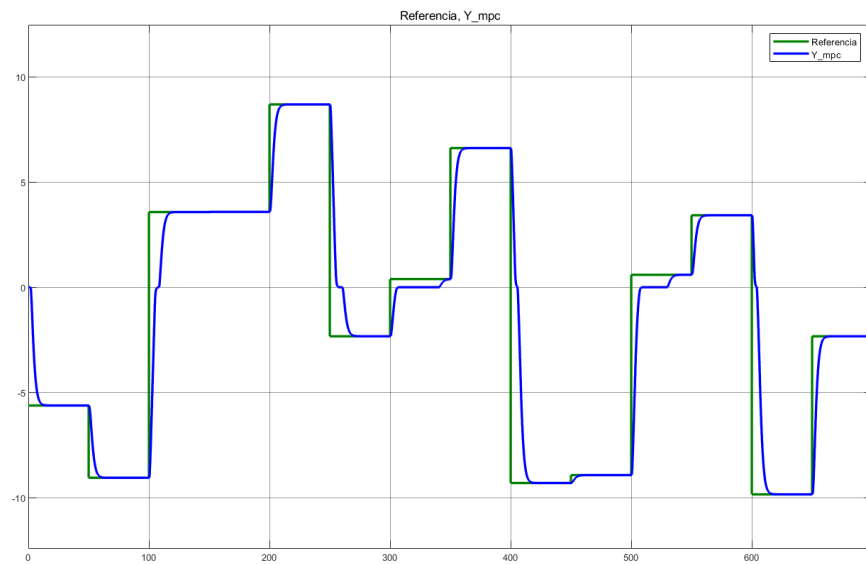


Fig. 9.14: Seguimiento de referencia Motor DC con banda muerta de $\pm 20[V]$

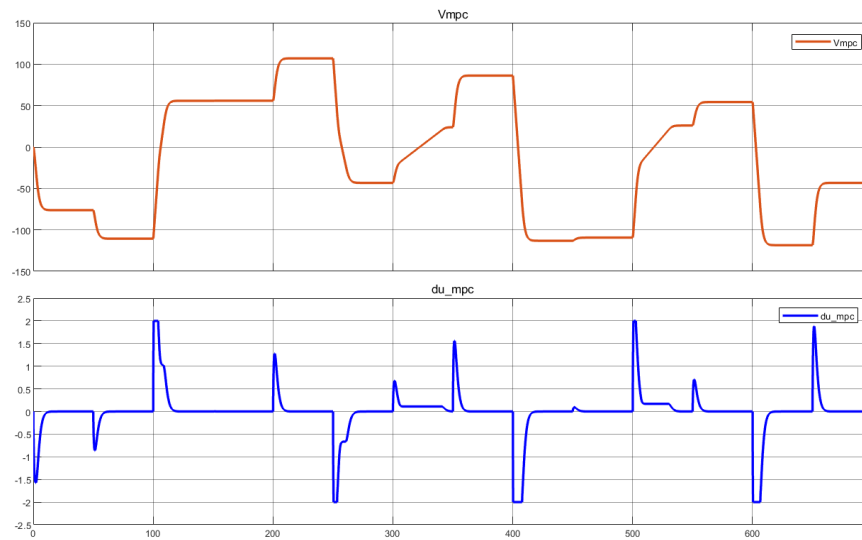


Fig. 9.15: Voltaje de entrada Motor DC y salida del MPC (du) con banda muerta de $\pm 20[V]$

Como había sido mencionado en capítulos anteriores, el esquema del controlador MPC con acción integral utilizado puede ser realizado sin el uso de observadores si es que se tiene acceso directo a los estados del sistema. Es por esto que a continuación se mostrará una comparativa del desempeño del controlador MPC con observador y el con medición directa de estados frente a distintas bandas muertas.

Para lograr lo antes mencionado fue necesario modificar el modelo en espacio de estados del motor DC que estamos usando (ecuaciones 7.3 y 7.4). Esta modificación consiste en hacer la matriz C_d una matriz identidad, lo que hace que la salida del motor DC sea el estado de este. Dicho estado consiste en la velocidad angular (salida del modelo usado anteriormente) y su corriente eléctrica. El nuevo modelo en espacio

de estados discreto del motor DC utilizado a continuación es:

$$\overbrace{\begin{bmatrix} \dot{\theta}[k+1] \\ i[k+1] \end{bmatrix}}^{x_{k+1}} = \overbrace{\begin{bmatrix} 0.3678 & 0.05635 \\ -0.001127 & 0.8187 \end{bmatrix}}^{A_d} \overbrace{\begin{bmatrix} \dot{\theta}[k] \\ i[k] \end{bmatrix}}^{x_k} + \overbrace{\begin{bmatrix} 0.006856 \\ 0.1813 \end{bmatrix}}^{B_d} \overbrace{V[k]}^{u_k} \quad (9.1)$$

$$y = \overbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}^{C_d2} \overbrace{\begin{bmatrix} \dot{\theta}[k] \\ i[k] \end{bmatrix}}^{x_k} \quad (9.2)$$

Como fue señalado, este modelo va a entregar como salida los estados, que en este caso serían dos variables. Los estados del motor DC deberán ser realimentados para calcular el estado del modelo extendido que utiliza el esquema MPC utilizado. Recordemos que este estado extendido es:

$$\overbrace{\begin{bmatrix} \Delta x_k \\ y_{k-1} - r_{k-1} \end{bmatrix}}^{\tilde{x}_k} \quad (9.3)$$

donde $\Delta x[k] = x[k] - x[k-1]$. Este estado extendido puede describirse como un vector columna cuyo primer elemento es la primera derivada del estado del sistema y su segundo elemento es el error de la velocidad angular respecto a su referencia en el instante anterior. El primer elemento descrito puede ser obtenido fácilmente utilizando una función de transferencia discreta, mientras que el segundo elemento ya lo obteníamos anteriormente. Esta configuración descrita queda expuesta en la Figura 9.16

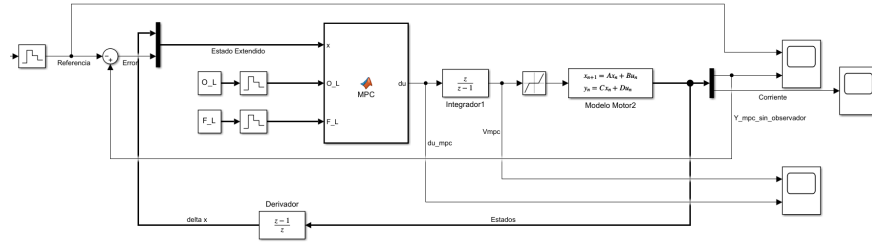


Fig. 9.16: Sistema Motor DC - MPC sin observador

Al comparar el seguimiento a referencia del MPC con la configuración inicialmente presentada frente al sin observador cuando hay presencia de banda muerta de $\pm 20[V]$ se obtiene la siguiente figura:

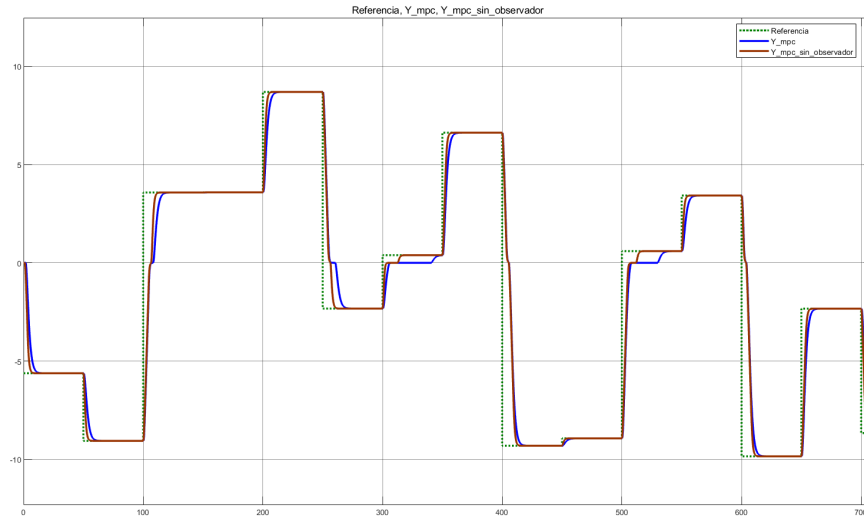


Fig. 9.17: Comparativa seguimiento a referencia MPC con/sin observador

En base a los resultados recién expuestos queda claramente definido que el sistema que tiene acceso directo a los estados del motor DC realiza un mejor seguimiento a referencia y supera mucho más rápido la banda muerta. Esto permite concluir que el esquema de control utilizado en esta memoria efectivamente tiene acción integral y puede ser utilizado tanto en sistemas con medición directa de sus estados, como en sistemas en los que sea necesario estimarlos mediante un observador; siendo el primer caso el preferible si es posible.

9.3. Resultados Red Neuronal

En la Figura 9.18 se muestra el seguimiento a referencia del sistema controlado por la red neuronal entrenada con datos del MPC con voltaje restringido entre $\pm 120[V]$ y variación de voltaje máximo de $\pm 2[V]$. Se observa que sigue de igual manera que el MPC a la referencia y en la Figura 9.19 se muestra que este sistema, a pesar de ser controlado por una red neuronal, si cumple con la restricción de máxima variación de voltaje (du).

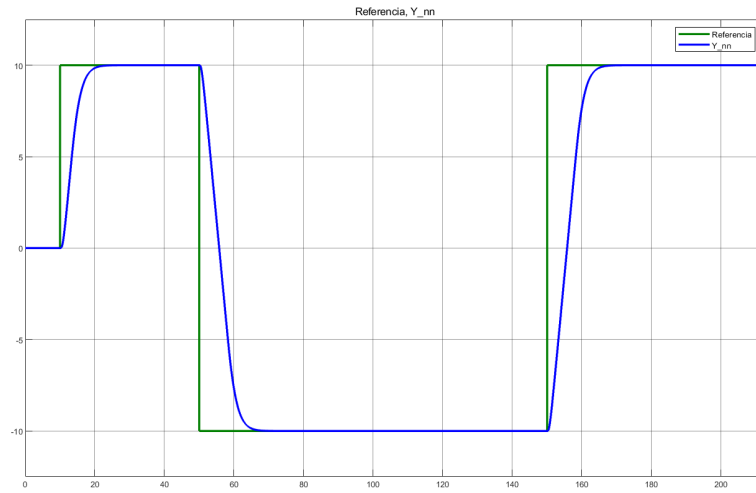


Fig. 9.18: Salida motor DC con restricciones $u \in [-120, 120]$ y $du \in [-5, 5]$. NN

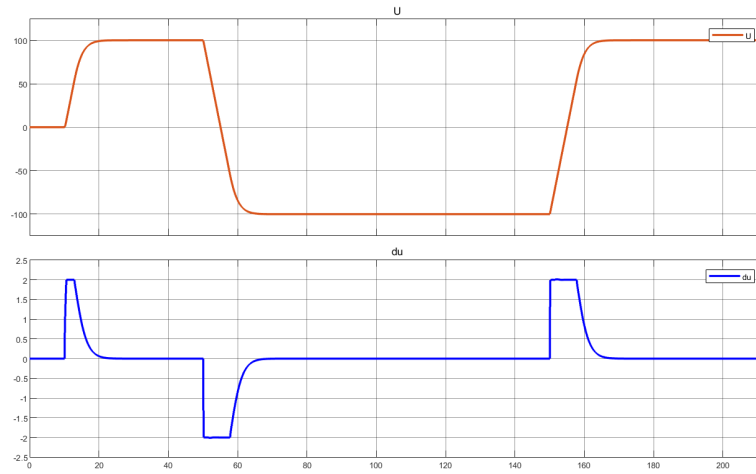


Fig. 9.19: Entrada del motor DC con restricciones $u \in [-120, 120]$ y $du \in [-2, 2]$. NN

Sin embargo, la restricción asociada al máximo y mínimo voltaje que puede recibir el motor no fue bien representada por dicha red neuronal, quedando demostrado en la Figura 9.20, que compara las entradas del motor DC siendo controlado por el MPC y la red neuronal cuando la referencia de velocidad buscada no puede ser alcanzada por el MPC debido a la restricción de voltaje máximo.

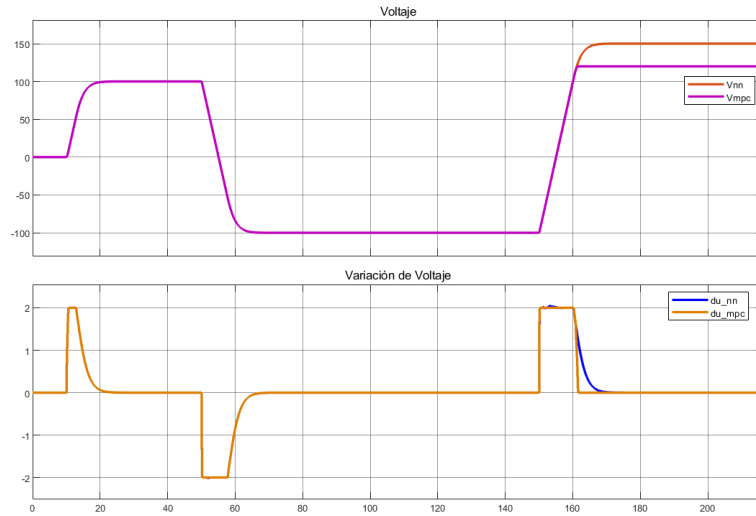


Fig. 9.20: Comparación actuaciones entre MPC y RN

En la Figura 9.21 se expone como el MPC no logra seguir la referencia en el

último escalón debido a que se para llegar a ese valor sería necesario exceder el voltaje máximo permitido. Por otro lado, la red neuronal si logra seguir la referencia, pero al costo de sobrepasar la restricción de voltaje de entrada máximo del motor.

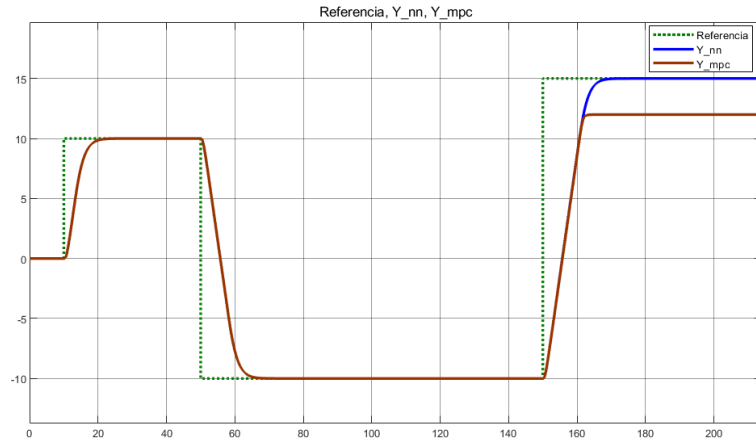


Fig. 9.21: Comparación seguimiento a referencia entre MPC y RN

Finalmente, en la Figura 9.22 se compara el seguimiento a referencia del MPC y de la red neuronal cuando ambos sistemas tienen una banda muerta de $\pm 20[V]$:

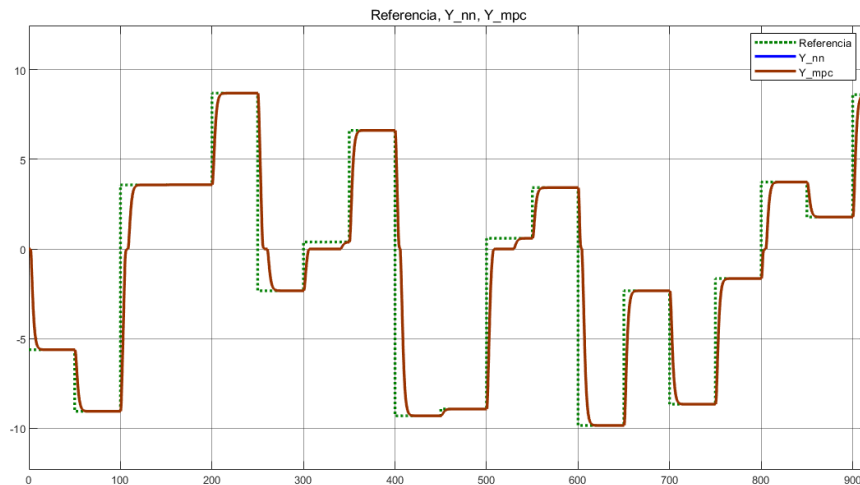


Fig. 9.22: Comparación seguimiento a referencia MPC - RN con banda muerta de $\pm 20[V]$

En la figura anterior no existe una diferencia apreciable entre estos dos controladores frente a la existencia de una banda muerta, por lo que la red neuronal, al igual

que el MPC, cuenta con acción integral.

10. CONCLUSIONES

El presente trabajo de titulación abordó la implementación de un controlador predictivo basado en modelo (MPC) utilizando redes neuronales, con el objetivo de optimizar el desempeño de un motor de corriente continua (DC) bajo diversas condiciones de operación. A lo largo de esta memoria se desarrollaron los modelos matemáticos necesarios, se implementaron algoritmos de control y se realizó una comparación de resultados entre el MPC tradicional y el controlador basado en redes neuronales.

A continuación, se presentan las principales conclusiones extraídas de este trabajo:

10.1. Conclusiones Generales

- Se logró implementar con éxito un controlador predictivo basado en modelo (MPC) tanto en su versión tradicional como en su versión aproximada mediante redes neuronales. La aproximación mediante redes neuronales demostró ser capaz de replicar el comportamiento del MPC explícito con un alto grado de precisión, lo que valida su uso como alternativa en sistemas donde se requieren tiempos de respuesta reducidos.
- El uso de redes neuronales permitió reducir significativamente la carga computacional en comparación con el MPC tradicional. Esto es especialmente útil en aplicaciones de control en tiempo real, donde el cálculo constante de las soluciones óptimas del MPC puede resultar costoso en términos de tiempo y recursos computacionales.
- La arquitectura de redes neuronales con tres capas ocultas y funciones de acti-

lación ReLU mostró ser suficiente para aproximar la ley de control del MPC explícito, lo que coincide con estudios previos que sugieren que cualquier red neuronal profunda ReLU puede ser representada funcionalmente con este tipo de arquitecturas.

- En los casos en que se aplicaron restricciones más estrictas a la acción de control (por ejemplo, límites en el voltaje de entrada), se observó que el controlador basado en redes neuronales tuvo dificultades para cumplir con dichas restricciones de manera precisa. Esto indica que, si bien las redes neuronales pueden aproximar el comportamiento del MPC, la capacidad de imponer restricciones explícitas de forma robusta sigue siendo un área de mejora.

10.2. Limitaciones

- Aunque el controlador basado en redes neuronales logró imitar el comportamiento del MPC, no siempre fue capaz de manejar correctamente las restricciones de entrada impuestas en el sistema. Esta limitación sugiere que el enfoque actual podría beneficiarse de mejoras en el diseño de la red neuronal o la incorporación de técnicas adicionales para garantizar el cumplimiento estricto de las restricciones.
- El entrenamiento de las redes neuronales fue dependiente de los datos generados por simulaciones del MPC. Por lo tanto, la precisión del modelo neuronal depende en gran medida de la calidad y variedad de los datos de entrenamiento. Si las condiciones de operación reales del sistema se desvían significativamente de los escenarios simulados, el rendimiento del controlador basado en redes neuronales podría degradarse.
- A pesar de que el uso de redes neuronales reduce el tiempo de cómputo, este enfoque requiere de un proceso de entrenamiento inicial que puede ser computacionalmente costoso, especialmente cuando se maneja un gran volumen de

datos o arquitecturas neuronales más complejas.

10.3. Trabajos Futuros

Para mejorar el rendimiento del controlador basado en redes neuronales, se sugiere explorar el uso de arquitecturas más avanzadas, como las redes neuronales recurrentes (RNN), las cuales tienen la capacidad de captar mejor las dinámicas temporales del sistema, reduciendo así los errores en la estimación de estados. Asimismo, se recomienda estudiar el efecto de entrenar la red neuronal utilizando directamente la señal de actuación que recibe el sistema a controlar, en lugar de entrenarla con la razón de cambio de dicha actuación, como se hizo en este trabajo. Este cambio podría mejorar el cumplimiento de las restricciones impuestas al sistema.

Otra línea de investigación interesante sería analizar cómo la cantidad de capas ocultas en una red neuronal ReLU influye en el número de restricciones que esta es capaz de cumplir correctamente al simular un MPC.

Además, sería de gran valor realizar una validación experimental del sistema implementado en un entorno físico, para evaluar la efectividad del controlador en condiciones reales de operación. Esta validación experimental permitiría identificar desafíos no considerados durante las simulaciones, como la presencia de ruido en las mediciones o perturbaciones no modeladas. Finalmente, una dirección prometedora sería la optimización del proceso de entrenamiento mediante técnicas de aprendizaje activo o la generación automática de escenarios de entrenamiento, lo que podría mejorar la capacidad de generalización del controlador basado en redes neuronales y reducir la dependencia de los datos simulados.

Bibliografía

- [1] 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, 2020.
- [2] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas, and M. Morari, “Approximating explicit model predictive control using constrained neural networks,” in *2018 Annual American Control Conference (ACC)*, 2018, pp. 1520–1527.
- [3] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017. [Online]. Available: <https://books.google.cl/books?id=MrJctAEACAAJ>
- [4] F. Behrooz, N. Mariun, M. Marhaban, M. A. Mohd Radzi, and A. Ramli, “Review of control techniques for hvac systems—nonlinearity approaches based on fuzzy cognitive maps,” *Energies*, vol. 11, p. 495, 02 2018.
- [5] D. Ruscio, “Model predictive control with integral action: A simple mpc algorithm,” *Modeling, Identification and Control: A Norwegian Research Bulletin*, vol. 34, pp. 119–129, 01 2013.
- [6] J. Katz, I. Pappas, S. Avraamidou, and E. N. Pistikopoulos, “The integration of explicit mpc and relu based neural networks,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11 350–11 355, 2020, 21st IFAC World Congress. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405896320308429>

- [7] K. Ogata, *Modern Control Engineering*. Prentice Hall, 2010.
- [8] H. H. Tan and K. H. Lim, “Vanishing gradient mitigation with deep learning neural network optimization,” in *2019 7th International Conference on Smart Computing Communications (ICSCC)*, 2019, pp. 1–4.
- [9] M. Kvasnica, J. Hledík, I. Rauová, and M. Fikar, “Complexity reduction of explicit model predictive control via separation,” *Automatica*, vol. 49, no. 6, pp. 1776–1781, 2013. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0005109813001076>
- [10] Y. Cao and R. B. Gopaluni, “Deep neural network approximation of nonlinear model predictive control,” *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11 319–11 324, 2020, 21st IFAC World Congress. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405896320308351>
- [11] H. Maghfiroh, A. A. A. Rizqi, O. Wahyunggoro, and A. Cahyadi, “Optimal energy control of dc motor speed control: Comparative study,” 11 2013.
- [12] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.
- [13] J. C. Cuevas-Tello, “Apuntes de redes neuronales artificiales,” 2018. [Online]. Available: <https://arxiv.org/abs/1806.05298>
- [14] M. J. Villani and N. Schoots, “Any deep relu network is shallow,” 2023. [Online]. Available: <https://arxiv.org/abs/2306.11827>