Control predictivo multivariable basado en Machine Learning: Aplicación de un modelo LSTM para el control por seguimiento del nivel y de la presión en un proceso real de ingeniería química.

Andrés Felipe Parra Barragán

Ingeniero Químico

Asesor
Danny Alexandro Múnera Ramírez, PhD Informática

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería Química
Medellín
2023

Dedicatoria

Inicialmente agradecer a la vida por permitirme llegar a este punto y alcanzar este sentimiento único. Para las dos personas más importantes en mi vida, Inés Bermúdez (Tita) y Carlos Andrés Parra (Pa), sin ustedes nada de esto hubiese sido posible, gracias por los muchos años de sacrificio, enseñanzas y, sobre todo, amor incondicional. Esto es un logro para y por ustedes, con el mayor esfuerzo y cariño posible, Felipe.

Agradecimientos

Después de altos y bajos, de meses sin rumbo y con muchas ideas sobre la mesa se finaliza esta gran etapa de mi vida. Gracias al profesor Danny Munera, por recibirme con los brazos abiertos ser tan receptivo y sobre todo guiarme de la manera que lo hizo; el aprendizaje que me llevo después de este proyecto me enriquece de una manera valiosa e imprevista, gracias a esto, ya sé en qué quiero profundizar mis estudios en postgrado.

Gracias al profesor Milton Ospina por su gran interés y apoyo en la realización de este trabajo de grado, gracias por ese impacto positivo que siempre intenta dejar en los estudiantes, los motiva, los impulsa y los guía hasta que lleguen a ser unos profesionales íntegros.

Tabla de contenido

Lista de tablas

Lista de figuras

Siglas, acrónimos y abreviaturas

IA	Inteligencia artificial
ML	Machine Learning
RMSE	Error de raíz cuadrada media
LSTM	Redes neuronales recurrentes con memoria a corto plazo
GPC	Control predictivo generalizado
MPC	Sistema de control predictivo de modelos
PLC	Controlador lógico programable
OPC	Objeto de Enlace y Embebido para el Control de Procesos

Resumen

En este trabajo de grado, se investigó la viabilidad técnica de utilizar un modelo de Machine Learning basado en 500.000 datos de regulación por dos controladores con acción proporcional e integral para lograr un control predictivo multivariable por seguimiento del nivel y la presión de un tanque en un proceso real de ingeniería química. El modelo elaborado y optimizado en Python, basado en redes neuronales recurrentes con memoria a corto plazo, demostró la capacidad de capturar dependencias secuenciales en los datos y realizar predicciones precisas. El RMSE obtenido fue inferior al 2% para un rango de nivel entre 5 y 25 cm y presiones entre 0 y 100 mbar. Sin embargo, se encontraron desafíos relacionados con la falta de control óptimo para el nivel y la presión del tanque, lo que destaca la importancia de combinar el modelo de Machine Learning con estrategias de control adicionales, como sistemas de control predictivo de modelos o la elaboración de múltiples modelos de Machine Learning en conjunto con algoritmos de control adaptativo. Estas conclusiones resaltan la necesidad de seguir investigando y refinando los enfoques utilizados para lograr un control preciso y óptimo de las variables controladas en sistemas de ingeniería química.

Palabras clave: Machine Learning, Redes neuronales, LSTM, optimización, PLC, control por seguimiento, RMSE, control de nivel, control de presión, control predictivo multivariable, Python.

Abstract

In this undergraduate thesis project, the feasibility of using a Machine Learning model based on 500.000 regulation data by two controllers with proportional and integral action to achieve multivariable predictive control by monitoring the level and pressure of a tank in a real chemical engineering process was investigated. The elaborated and optimized model in Python, based on recurrent neural networks with short-term memory, demonstrated the ability to capture sequential dependencies in the data and make accurate predictions. The RMSE obtained was below 2% for levels between 5 and 25 cm and pressures between 0 and 200mbar. However, challenges were encountered related to the lack of optimal control for the tank level and pressure, highlighting the importance of combining the Machine Learning model with additional control strategies, such as model predictive control systems or the development of multiple Machine Learning models in conjunction with adaptive control algorithms. These findings underscore the need for further research and refinement of the approaches used to achieve accurate and optimal control of controlled variables in chemical engineering systems.

Keywords: Machine Learning, Neural Networks, LSTM, optimization, PLC, tracking control, RMSE, level control, pressure control, multivariable predictive control, Python.

1 Introducción

La ingeniería química es una materia rica en datos. Los profesionales recopilan y analizan datos para comprender patrones de flujo, desarrollar modelos empíricos, diseñar y optimizar reacciones químicas, así como supervisar y controlar procesos y sistemas químicos. Debido a la gran dependencia de los datos, parece natural que la inteligencia artificial y el Machine Learning desempeñen un papel importante en este campo. Además, el aumento de la potencia en los ordenadores y el tamaño de los conjuntos de datos ha provocado un cambio drástico en la accesibilidad de todos los componentes clave del desarrollo de modelos de aprendizaje automático.

El control de los procesos en la ingeniería química es fundamental para garantizar la eficacia y la calidad de los sistemas industriales [11]. Tradicionalmente, se han utilizado controladores con acción proporcional e integral para regular variables clave en estos procesos. Estos controladores son efectivos en una amplia gama de aplicaciones y han demostrado su utilidad en el control de variables como la presión, la temperatura, el caudal y la concentración [12]. Sin embargo, estos controladores tienen limitaciones en términos de su capacidad para abordar procesos con una dinámica más compleja y multivariable [11]. En muchos casos, estos controladores solo permiten el control de una variable manipulada a una variable controlada, lo que dificulta su implementación en sistemas que requieren el control simultáneo de múltiples variables interrelacionadas.

En este contexto, el presente trabajo de grado se centró en el desarrollo de un modelo basado en Machine Learning y redes neuronales para realizar el control predictivo multivariable por seguimiento del nivel y la presión en un tanque basado en datos de regulación por controladores PI. Esto, se realizó mediante la toma de datos en un proceso de forma continua controlado mediante dos controladores con acción proporcional e integral y el desarrollo, optimización y validación de un modelo basado en Machine Learning y redes neuronales en Python para realizar el control del nivel y la presión en un tanque en un proceso real.

El entrenamiento del modelo se realizó con aproximadamente 500.000 datos de cinco variables de entrada y dos variables de salida, se programó un modelo basado en redes neuronales recurrentes con memoria a corto plazo (LSTM), debido a la naturaleza temporal de los datos. En la optimización de los hiperparámetros realizada mediante una cuadrícula de búsqueda

(GridSearchCV) y las técnicas de segmentación de datos, Times Series Split y Cross Validation, se encontró que la mejor combinación de los hiperparámetros fue una longitud de secuencia de 25, 128 neuronas, 10 épocas y un tamaño de lote de 128.

Posteriormente se realizaron pruebas del modelo en el proceso real en diferentes puntos de operación dentro y fuera del rango de entrenamiento del modelo para evaluar su capacidad de predicción y generalización. Se encontró que a pesar de que el modelo es capaz de realizar predicciones con un RMSE por debajo del 2% en los porcentajes de aperturas de las válvulas de control para el nivel y la presión del tanque, no realiza un control óptimo en ningún escenario. Llegando con esto a la conclusión de que es necesario realizar una implementación en conjunto con un modelo de control predictivo generalizado (GPC), un sistema de control predictivo de modelos (MPC) o la elaboración de dos modelos basados en Machine Learning que permitan minimizar los residuales y predecir el nivel y la presión del tanque por cada paso de la secuencia dadas las condiciones anteriores del sistema.

En resumen, la ingeniería química es un campo que se basa en la recopilación y análisis de datos para comprender y controlar los procesos químicos. Con el avance de la inteligencia artificial y el aprendizaje automático, se ha abierto la puerta a nuevas posibilidades para el control predictivo multivariable en la ingeniería química. Este trabajo de grado se enfocó en el desarrollo de un modelo basado en Machine Learning y redes neuronales para controlar el nivel y la presión en un tanque, utilizando datos de un proceso real. Aunque se lograron predicciones precisas, se identificó la necesidad de combinar este modelo con estrategias de control optimizadas para obtener un control óptimo del sistema.

En los siguientes capítulos se detallarán los objetivos de este trabajo de grado (Capítulo 2), el marco teórico asociado a las herramientas para el desarrollo de un modelo de Machine Learning (Capítulo 3), el estado del arte (Capítulo 4), la metodología llevada a cabo (Capítulo 5), las implementaciones realizadas con las diferentes tecnologías usadas (Capítulo 6), los resultados y el análisis de los resultados obtenidos (Capítulo 7), las conclusiones (Capítulo 8) y las recomendaciones para futuras investigaciones (Capítulo 9).

2 Objetivos

2.1 Objetivo general

Elaborar un modelo basado en Machine Learning que permita realizar el control predictivo multivariable por seguimiento del nivel y la presión en un tanque en un proceso real.

2.2 Objetivos específicos

Recolectar los datos del proceso en forma continua y regulado por dos controladores con acción proporcional e integral.

Definir el tipo de modelo de Machine Learning que se entrenará y optimizará con los datos recolectados.

Entrenar y optimizar el modelo de Machine Learning escogido y obtener la mejor combinación de hiperparámetros estableciendo una métrica estadística justificada.

Validar el modelo de Machine Learning optimizado e interpretar los resultados obtenidos.

3 Marco teórico

El desarrollo de un control multivariable mediante el uso de un modelo de Machine Learning y redes neuronales es una aplicación específica de la inteligencia artificial.

La inteligencia artificial (IA) se refiere a la capacidad de las máquinas y los sistemas informáticos para simular el comportamiento inteligente humano. Se trata de crear sistemas que puedan percibir y comprender su entorno, tomar decisiones basadas en la información disponible y aprender de experiencias pasadas para mejorar su desempeño [1].

En el campo de la ingeniería química, la inteligencia artificial tiene un gran potencial y aplicaciones recurrentes. Los modelos de datos basados en técnicas de aprendizaje automático (Machine Learning) y el control predictivo basado en modelos de redes neuronales son ejemplos claros de cómo la IA puede ser utilizada para abordar problemas en este campo de la ingeniería.

Estos modelos pueden analizar una gran cantidad de datos, identificar patrones y tendencias, realizar predicciones, controlar y optimizar procesos químicos. En el ámbito de la programación y la ingeniería química, un modelo es una representación simplifica y abstracta de un sistema o fenómeno que se desea analizar. Un modelo es una herramienta que permite comprender, predecir y optimizar el comportamiento de un sistema real.

Los modelos basados en Machine Learning pueden ser utilizados para mejorar la eficiencia y la precisión en la formulación de productos químicos, la optimización de reacciones químicas, el control predictivo de las variables de los procesos, la detección de anomalías y la toma de decisiones basadas en datos en tiempo real. La aplicación de técnicas de Machine Learning en la ingeniería química, mencionada anteriormente, se basa en la capacidad de los algoritmos de aprendizaje automático para analizar grandes cantidades de datos y construir modelos que permitan el diseño, la optimización y el control de procesos químicos de manera más eficiente y precisa.

El Machine Learning es una técnica de la inteligencia artificial que enseña a las computadoras a aprender de la experiencia, haciendo uso de algoritmos de aprendizaje automático y métodos computacionales para "aprender" información directamente de los datos sin depender de una ecuación predeterminada como modelo. Los algoritmos de aprendizaje automático pueden dividirse fácilmente en dos clases: "no supervisados" y "supervisados" [1].

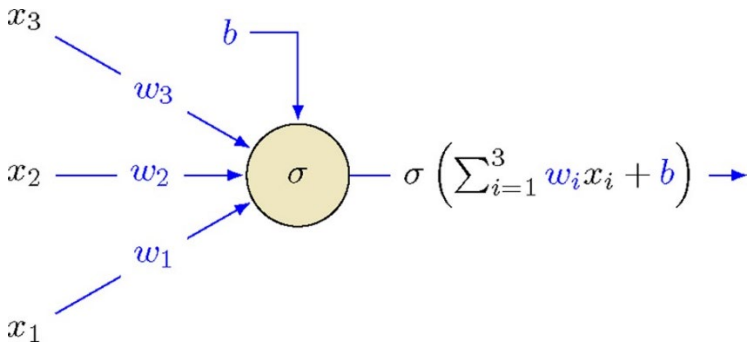
Los algoritmos de aprendizaje supervisado implican la creación de un modelo que asigna entradas conocidas a salidas conocidas. En este tipo de aprendizaje, se empieza con una serie de datos de entrenamiento que consisten en las entradas y las salidas asociadas. La mayor parte de los datos de entrenamiento se utilizan para entrenar el modelo, con una parte "reservada" para validación y testeo del modelo [2].

La cantidad de datos necesarios depende en gran medida de la aplicación, y la disponibilidad de los datos está relacionada con la fuente y el coste de adquisición de estos. La construcción de un modelo de alta calidad utiliza habitualmente entre miles y cientos de miles de observaciones, ya que se necesitan datos suficientes para inferir relaciones entre las entradas y salidas conocidas [2]. Debido a la inherente cantidad de datos en el estudio de la ingeniería química, es de gran utilidad la inteligencia artificial y el Machine Learning en ramas específicas como el desarrollo de modelos empíricos, diseño, optimización, control y monitoreo de procesos químicos. [3]. Esta necesidad de datos suficientes para inferir relaciones entre las entradas y salidas conocidas ha llevado al desarrollo de técnicas basadas en redes neuronales, donde cada "neurona" realiza cálculos ponderados sobre los datos de entrada para determinar su activación [2].

La terminología de las redes neuronales tiene su origen en las analogías con las neuronas del cerebro; por lo tanto, cada una de estas transformaciones se denomina "neurona" o "nodo". El modelo básico de un nodo es el siguiente: el nodo calcula una suma ponderada sobre el vector de entrada, combinando elementos del espacio de características, x_i, con pesos únicos, w_i. Esto se muestra en la **Figura 1**. La salida de una neurona idealizada es cero a menos que la suma total calculada supere un determinado umbral, en cuyo caso el nodo se "enciende" y propaga una señal corriente abajo [2].

Figura 1

Representación del modelo básico de un nodo

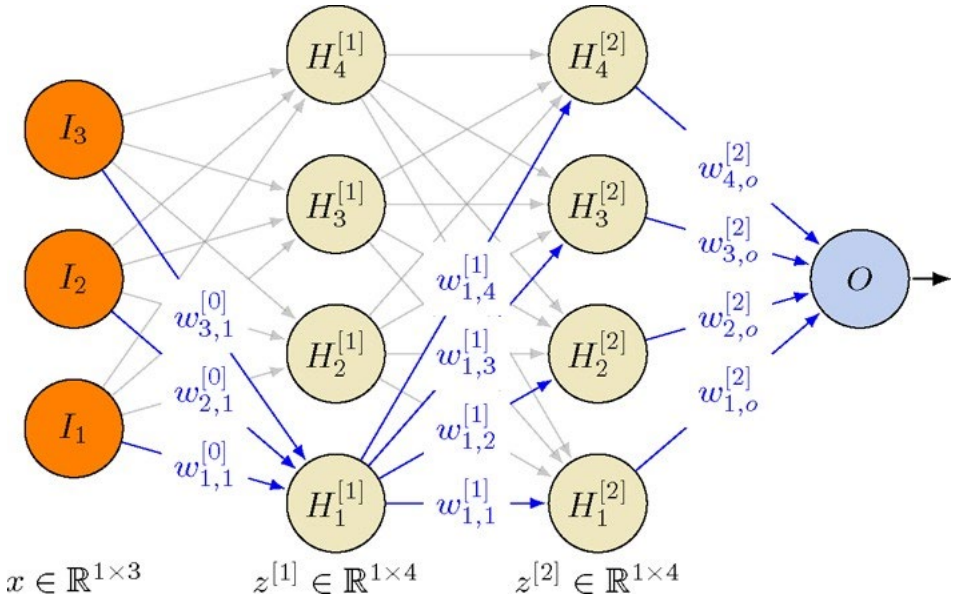


Nota. Fuente https:// bit.ly/3D8KFpW [2].

Sin embargo, las capacidades de una sola neurona pueden ser limitadas. Por ello, se agrupan muchos nodos de este tipo en una capa y, a continuación, agrupamos varias capas en una red como se muestra en la **Figura 2**. Esta figura, representa una vista general de un modelo de perceptrón multicapa, que muestra tres nodos de entrada, (I), conectados a dos capas de cuatro nodos ocultos, (H) y un nodo de salida escalar (O) [2].

Figura 2

Vista general de un modelo de perceptrón multicapa



Nota. Fuente https:// bit.ly/3D8KFpW [2].

Muchas fuentes de datos tienen una estructura adicional que limita la independencia de las observaciones individuales. Un ejemplo son los datos secuenciales, en los que las entradas tienen un orden natural, como ocurre en el procesamiento datos que dependen del tiempo. En este caso, una variable de entrada cambia a medida que el tiempo avanza. Las redes neuronales recurrentes son una forma de codificar esta naturaleza secuencial. Cuando los datos secuenciales tienen relaciones a largo plazo en secuencias largas y es necesario hacer predicciones de series temporales, es recomendado usar las redes neuronales recurrentes LSTM (Long-Short-Term Memory). Este tipo de redes neuronales tiene tres componentes principales: una celda de memoria, una puerta de entrada y una puerta de salida [2].

Las redes neuronales LSTM utilizan datos secuenciales y/o datos temporales como los que se recolectan, por ejemplo, en un proceso que involucra el control predictivo de un proceso dinámico no lineal multivariable por seguimiento en la ingeniería química, usando la automatización de procesos y los controladores lógicos programables (PLC). Por esto, es necesario usar librerías como Times Series Split, que implementa la división de las muestras de datos de series temporales que se observan a intervalos de tiempo fijos, en conjuntos de

entrenamiento/prueba. Junto a Times Series Split, la técnica de Cross Validation tienen en cuenta la estructura temporal de los datos al dividirlos en conjuntos de entrenamiento y prueba [4].

En la elaboración del modelo, es necesario la definición de los hiperparámetros. Estas variables, son variables de configuración externa, que se utilizan para administrar el entrenamiento de modelos de Machine Learning. Estos, se pueden configurar de forma manual por criterio del programador o se puede realizar un procedimiento de optimización, obteniendo el mejor conjunto de hiperparámetros para el modelo estudiado, teniendo en cuenta una métrica estadística de error específica. El ajuste de hiperparámetros es un proceso importante e intensivo desde el punto de vista computacional, ya que permiten que el modelo sea más o menos robusto. Además, Seleccionar el conjunto correcto de hiperparámetros es importante en términos de rendimiento y precisión del modelo [4]. En los hiperparámetros necesarios para la elaboración de un modelo basado en Machine Learning está el número de divisiones, la longitud de la secuencia, el número de neuronas y el tamaño del lote y el optimizador.

El número de divisiones es un hiperparámetro, que determina la cantidad de particiones que se realizarán en el conjunto de datos de series de tiempo durante la validación cruzada. Cada división tiene un conjunto de prueba y uno o más conjuntos de entrenamiento, y se mueve hacia adelante en el tiempo para capturar la estructura temporal de los datos. El valor mínimo permitido es de 2 [5].

La longitud de la secuencia es un hiperparámetro que define el número de pasos de tiempo consecutivos que se utilizan como entradas para predecir el siguiente paso de tiempo. En un problema de series de tiempo, se utiliza una ventana deslizante para crear secuencias de datos consecutivas. Cada secuencia consiste en una serie de pasos de tiempo anteriores y se utiliza para predecir el siguiente paso de tiempo [6].

El número de neuronas es otro hiperparámetro que define el programador. Las neuronas son las unidades básicas de procesamiento en una red neuronal artificial. Se denominan neuronas a los nodos interconectados que transmiten la información que el modelo va aprendiendo, a medida que la información de las variables de entrada y/o otras neuronas pasan por las diferentes capas del modelo. Las conexiones entre las neuronas están representadas por los pesos sinápticos, que determinan la influencia que una neurona tiene sobre otra. Estos pesos se ajustan durante el proceso de entrenamiento del modelo para aprender patrones y relaciones en los datos [2].

Otros hiperparámetros del modelo son el número de épocas, el tamaño de lote y el optimizador. El número de épocas en modelos basados en Machine Learning, son la cantidad de ciclos completos a través del conjunto de datos de entrenamiento e indica la cantidad de pases que el algoritmo de Machine Learning ha completado durante el entrenamiento. El tamaño de lote en modelos basados en Machine Learning, es la cantidad de muestras de entrenamiento que se utilizan en una iteración antes de que los parámetros del modelo se actualicen. En lugar de utilizar todo el conjunto de datos de entrenamiento al mismo tiempo, se divide en lotes más pequeños. El optimizador es un algoritmo que optimiza los valores de los parámetros para reducir el error cometido por la red. Es decir, es el encargado de encontrar el conjunto óptimo de parámetros que permita que el modelo se ajuste de manera precisa a los datos de entrenamiento y generalice de una manera eficiente nuevos datos. Además de la definición de los hiperparámetros, también se debe definir una función de pérdida para el algoritmo del entrenamiento del modelo. Esta función, mide la divergencia entre la predicción de un algoritmo de Machine Learning y la salida supervisada y representa el costo de equivocarse [7].

Si le definición de los hiperparámetros no se realiza por criterio del programador, por lo general se realizan procedimientos de optimización de hiperparámetros. Este procedimiento, permite establecer la combinación más adecuada de hiperparámetros basada en una métrica estadística de rendimiento establecida. La búsqueda por cuadrícula (GridSearchCV), es una técnica de optimización donde se especifica una lista de hiperparámetros con diferentes valores y una métrica estadística de rendimiento, y el algoritmo trabajo con todas las combinaciones de hiperparámetros posibles para determinar la opción más adecuada [4].

Una de las métricas más usadas para modelos que predicen variables con sentido físico es el error de raíz cuadrada media (RMSE), este error da un estimado de la desviación estándar de los valores residuales. Estos valores, son una media de la distancia de los puntos de datos de la línea de regresión y se calcula de la siguiente manera [8]:

Ecuación 1.
Modelo de cálculo para el error de raíz cuadrada media

$$RMSE = \sqrt{\frac{\sum(\hat{y}_i - y_i)^2}{n}}$$

Donde $\hat{y}_i$ es el valor predicho, y_i es el valor esperado y n es el número de datos evaluados.

En la ingeniería química, los PLC's se pueden usar en conjunto con los softwares WinCC y STEP 7 para la automatización y control de procesos. El controlador lógico programable (PLC) es la unidad central de control para un proceso. Estos tienen la función de detectar diversos tipos de señales del proceso, realizar acciones de acuerdo con lo que se ha programado, permitiendo tratar estas señales para realizar, por ejemplo, el control de las variables termodinámicas de un proceso en específico. Los componentes principales son la fuente de energía, el CPU controlador, el sistema E/S de entradas y salidas, y el software de programación.

En la implementación de un control predictivo multivariable por seguimiento haciendo uso de Machine Learning y redes neuronales recurrentes, es necesario realizar la recolección de una gran cantidad de datos de buena calidad. Gracias a los PLC's se puede realizar una recopilación de datos de diferentes variables termodinámicas como la concentración, el nivel, el flujo, la presión, la temperatura, entre otras.

El software SIMATIC WinCC permite el desarrollo de la interfaz hombre máquina (HMI). Un sistema de interfaz hombre máquina permite realizar el control, la supervisión y la adquisición de datos. Esta herramienta se integra con el PLC para realizar una automatización dinámica del proceso. WinCC permite realizar una optimización en la organización de archivos, administración de datos, representaciones gráficas del proceso, y sincroniza los tiempos de operatividad [9].

En el proceso de configuración y programación de los sistemas de automatización de SIMATIC se usa el software SIMATIC Step 7. Este software, permite crear y gestionar proyectos, configurar y parametrizar el hardware y la comunicación, gestionar símbolos, crear programas, comprobar el sistema automatizado y diagnosticar fallos en la instalación [10].

En la ingeniería química, además de ser necesaria la automatización de los procesos es fundamental y obligatorio el control de las variables de interés. El control automático de procesos

permite mantener en un valor de operación deseado las variables del proceso tales como la temperatura, presiones, flujos, nivel, entre otras. Este control, permite que las variables se controlen sin la necesidad de la intervención de un operador. Para esto, se requiere la implementación de un sistema de control y sus componentes básicos sensor transmisor, controlador y elemento final de control. La variable controlada, es la variable que se desea mantener o controlar en un valor deseado fijo o variable. Al valor deseado, se le denomina set point y cuando éste es variable, el tipo de control por realizar es un control por seguimiento [11]. Los controladores tradicionales usados son controladores PI.

El controlador PI, es un controlador con acción proporcional e integrativa, esta acción combinada reúne las ventajes de ambas acciones. Con un control proporcional, es necesario que exista error para tener una acción de control distinta de cero. Con la acción integral, un pequeño error positivo, siempre dará una acción de control creciente y viceversa. Este tipo de controladores se ajustan de una manera adecuada cuando los sistemas son de primer orden [12]. En la actualidad, los modelos basados en Machine Learning y redes neuronales recurrentes, están tomando cada vez más fuerza en la ingeniería química, desplazando poco a poco los controladores con acción proporcional e integrativa.

Cuando se está realizando la automatización industrial, en ocasiones es necesario implementar una tecnología de comunicación cliente-servidor. Una de las tecnologías de comunicación industrial estándar más utilizadas en la actualidad es OPC (Object Linking and Embedding for Process Control). OPC permite la comunicación sin restricciones o límites impuestos por los fabricantes, lo que significa que diferentes dispositivos y aplicaciones de control de diferentes marcas comerciales pueden intercambiar información de manera continua y en tiempo real [13]. Además, ha permitido una integración efectiva de sistemas y dispositivos de diferentes proveedores. Esto ha mejorado la eficiencia, la interoperabilidad y la capacidad de monitoreo y control en tiempo real en una amplia gama de aplicaciones industriales.

En un entorno OPC, un servidor OPC se comunica de manera continua con los PLC's de campo, las estaciones de interfaz hombre-máquina u otras aplicaciones. Aunque el hardware y el software provengan de diferentes marcas, el cumplimiento del estándar OPC garantiza la compatibilidad y la comunicación fluida entre ellos [13].

3.1 Herramientas para el desarrollo de un modelo basado en Machine Learning

En la elaboración de un modelo basado en Machine Learning es fundamental el uso de un lenguaje de programación específico y el uso de librerías para facilitar el procedimiento de programación.

3.1.1 Lenguaje de programación

Python es un lenguaje de programación potente que posee eficientes estructuras de datos de alto nivel y un enfoque sencillo pero eficaz de la programación orientada a objetos. Es un lenguaje tipado e interpretado, lo convierte en un lenguaje ideal para la creación de scripts y el desarrollo rápido de aplicaciones en muchas áreas y en la mayoría de las plataformas [14].

Este lenguaje de programación se utiliza para realizar tareas como la creación de sitios web, automatización de tareas y procesos, realizar análisis de datos y visualización de datos. Tiene un propósito general, es decir, se puede utilizar para crear una variedad de programas diferentes y no está especializado en un programa específico [15].

Python se ha convertido en un elemento clave en la ciencia de datos, permitiendo crear algoritmos y modelos basados en Machine Learning para completar tareas específicas en áreas como la ingeniería química. Estas actividades se facilitan por la gran variedad de bibliotecas que permiten a los programadores escribir los programas de forma más rápida, como Tensorflow, Pandas, Numpy, Matplotlib, Sklearn y Keras [15].

3.1.2 Librerías

Las librerías son conjuntos de código predefinido que contienen funciones, clases y recursos que se utilizan para facilitar el desarrollo de software. Una librería es esencialmente un conjunto de código reutilizable que ha sido creado para realizar tareas específicas.

Pandas es una librería de Python para el análisis y la manipulación de datos de código abierto, rápido, potente y flexible. Es un conjunto de herramientas que permite el tratamiento de los datos de una manera fácil y eficiente. Por ejemplo, Pandas permite usar herramientas para leer

y escribir datos entre estructuras de datos en memoria y diferentes formatos: CSV, archivos de texto, Microsoft Excel y bases de datos SQL. Es un proyecto de código abierto y se puede utilizar libremente [16].

NumPy es una librería de Python que se utiliza para trabajar con matrices. También tiene diferentes funciones matemáticas para trabajar en el dominio del álgebra, álgebra lineal, la transformada de Fourier, entre otras. Es un proyecto de código abierto y se puede utilizar libremente [17].

Matplotlib es una completa biblioteca para crear visualizaciones estáticas, animadas e interactivas en Python. Esta librería, permite crear gráficos con calidad de publicación, hacer figuras interactivas que se puedan ampliar, desplazar y actualizar, personalizar el estilo visual y el diseño, y también permite la exportación a muchos formatos de archivo. Es un proyecto de código abierto y se puede utilizar libremente [18].

TensorFlow es una plataforma integral de código abierto para el Machine Learning (ML). Cuenta con un gran y flexible ecosistema de herramientas, bibliotecas y recursos comunitarios que permite a los desarrolladores crear y desplegar fácilmente aplicaciones basadas en ML. Es un proyecto de código abierto y se puede utilizar libremente [19].

Scikit-learn (Sklearn) es la librería más útil y robusta para el Machine Learning en Python. Proporciona una selección de herramientas eficientes para el ML y el modelado estadístico, incluyendo clasificación, regresión, clustering y reducción dimensional a través de una interfaz consistente en Python. Esta biblioteca, escrita en gran parte en Python, se basa en NumPy, SciPy y Matplotlib. Es un proyecto de código abierto y se puede utilizar libremente [20].

Keras es una biblioteca de redes neuronales de alto nivel que se ejecuta sobre TensorFlow. Keras proporciona API de alto nivel para construir y entrenar modelos con facilidad, ya que está integrada en Python. Es un proyecto de código abierto y se puede utilizar libremente [21].

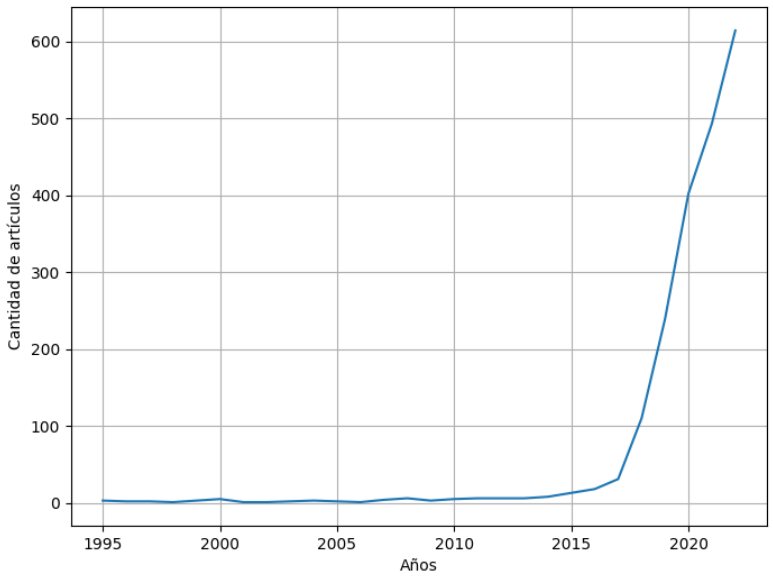
4 Estado del arte

En los últimos años, se ha evidencia un aumento significativo en la investigación centrada a la aplicación del Machine Learning en áreas de ingeniería. Esto se aplica especialmente en la ingeniería química, donde los procesos están altamente influenciados por su naturaleza dinámica, no lineal y su representación matemática, física y fenomenológica es complicada. Por lo tanto, se vuelve de vital importancia la implementación de estrategias avanzadas que permitan, por ejemplo, la representación matemática, física y fenomenológica de un proceso, por medio de modelos basados en Machine Learning. Además de la representación de un proceso, es sumamente importante el control predictivo de las variables termodinámicas de interés.

Lo anterior, se evidencia por medio de las siguientes búsquedas realizadas en Scopus. La primera se realizó el 24 de junio del 2023 con la siguiente cadena: (TITLE-ABS-KEY(Machine learning AND recurrent neural networks) AND (LIMIT-TO (PUBSTAGE,"final")) AND (LIMIT-TO (DOCTYPE,"ar") OR LIMIT-TO (DOCTYPE,"re")) AND (LIMIT-TO (SUBJAREA,"ENGI"))).

Figura 3

Número de publicaciones por año, relacionadas con Machine Learning, redes neuronales recurrentes aplicadas en ingeniería



Nota. Elaboración propia.

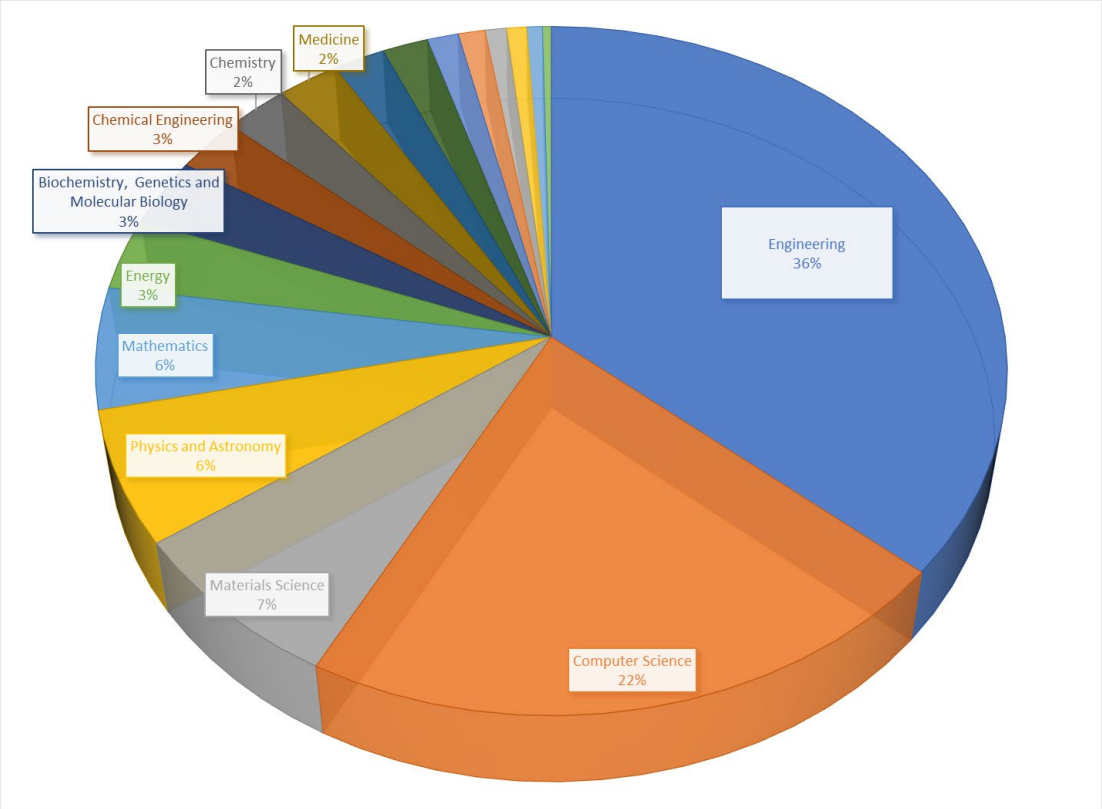
Figura 3 se evidencia un crecimiento exponencial en la cantidad de publicaciones relacionadas con Machine Learning y redes neuronales recurrentes aplicadas en ingeniería a partir del 2018. Este crecimiento evidencia el creciente interés en estas temáticas. Por esto, las investigaciones en las que se aborde específicamente el uso de estas técnicas en un campo particular, como la ingeniería química, se vuelven altamente atractivas y representan un nicho de investigación prometedor.

Ahora, con la siguiente cadena: (TITLE-ABS-KEY(Machine learning AND recurrent neural networks) AND (LIMIT-TO (PUBSTAGE,"final")) AND (LIMIT-TO (DOCTYPE,"ar") OR LIMIT-TO (DOCTYPE,"re")) AND (LIMIT-TO (SUBJAREA,"ENGI"))), se realizó la búsqueda de publicaciones relacionadas con Machine Learning y redes neuronales recurrentes en diferentes campos de aplicación. En la siguiente figura y en la **Tabla 1**, se presenta un resumen de los resultados obtenidos en la búsqueda realizada el 24 de junio de 2023.

En la **Figura 4** se evidencia que el campo de mayor aplicación para investigaciones enfocadas a Machine Learning y redes neuronales recurrentes es en la ingeniería en los últimos años. Además, específicamente la ingeniería química, representa un 3% del total de publicaciones, siendo esto, un total de 162 publicaciones en estos tópicos al 2022.

Figura 4

Distribución porcentual de la cantidad de publicaciones relacionadas con Machine Learning y redes neuronales recurrentes por campo de aplicación



Nota. Elaboración propia.

Lu et al. (2007), propusieron una metodología de diseño de diseño sistémico para desarrollar un control predictivo basado en redes neuronales difusas recurrentes con acción integral para una clase de sistemas de procesos no lineales.

Wu et al. (2019), realizaron un estudio que se centró en el diseño de un sistema de control predictivo de modelos (MPC) para un proceso de un reactor químico mediante un modelo basado en redes neuronales recurrentes. El sistema de control se fundamentó en Laypunov (LMPC) y utiliza modelos de redes neuronales recurrentes como modelo de predicción para lograr un control de lazo cerrado. La estructura de este modelo de regresión se basó en un algoritmo de aprendizaje de redes neuronales recurrentes, K-fold y validación cruzada para la etapa de entrenamiento y validación del modelo [22]. Este trabajo muestra como los modelos basados en redes neuronales

recurrentes pueden captar la dinámica de un proceso en una determinada región de funcionamiento, con una toma de datos en lazo abierto y posteriormente usarse como herramientas para un sistema de control predictivo.

Singh et al. (2017), entrenaron y validaron un modelo basado en redes neuronales recurrentes de una columna de destilación binaria continua (BDC) utilizando datos experimentales y estudio demostró que la predicción de este tipo de modelos puede superar un modelo de primeros principios [1] para procesos no lineales complejos a gran escala, debido a su alto grado de libertad para resolver el complejo problema de regresión no lineal utilizando el conjunto de datos del proceso [23].

Wu et al. (2019), plantearon una formulación de un control predictivo de modelos (MPC) diseñado para establecer el control de una variable con respecto a su valor de referencia fijo, maximizando el rendimiento del anhídrido ftálico en un reactor de lecho fijo y alejando la temperatura del punto caliente para evitar la desactivación del catalizador utilizando un modelo de regresión por conjuntos.

[1] Modelo basado en leyes físicas experimentales, ecuaciones de balance y restricciones.

Tabla 1

Número de publicaciones relacionadas con Machine Learning y redes neuronales recurrentes por campo de aplicación

Campo de aplicación	Número de publicaciones
Engineering	2287
Computer Science	1371
Materials Science	473
Physics and Astronomy	398
Mathematics	361
Energy	214
Biochemistry, Genetics and Molecular Biology	201
Chemical Engineering	**162**
Chemistry	156
Medicine	153
Environmental Science	128
Social Sciences	113
Health Professions	77
Earth and Planetary Sciences	66
Decision Sciences	53
Neuroscience	49
Agricultural and Biological Sciences	39
Immunology and Microbiology	22
Economics, Econometrics and Finance	6
Multidisciplinary	5
Pharmacology, Toxicology and Pharmaceutics	3
Arts and Humanities	3
Nursing	2
Psychology	1
TOTAL	6344

Nota. Elaboración propia.

Los autores usaron una base de datos, de un amplio régimen de funcionamiento, creada a partir de un algoritmo de solución de fluidodinámica computacional (CFD) de última generación, a partir de la cual usaron técnicas de redes neuronales recurrentes y aprendizaje de conjuntos para derivar un modelo de regresión de conjunto homogéneo utilizando una interfaz de programa de aplicación (API) de última generación, Keras. El modelo CFD, el modelo de regresión por conjuntos y el MPC se combinan para crear un sistema de lazo cerrado mediante la vinculación de ANSYS Fluent [2] a SciPy [3] a través de una interfaz de paso de mensajes (MPI) con mecanismo de sincronización de bloqueo [24].

Los reactores de lecho fijo con reacciones altamente exotérmicas exponen varios problemas de control en la ingeniería química debido a su extrema no linealidad, su dinámica no lineal y distribuida espacialmente, los puntos calientes móviles, el alto riesgo de fuga térmica, las restricciones en las variables de entrada y de estado manipuladas, y las limitadas mediciones en línea con alta incertidumbre y grandes tiempos de retraso. Por esto, es un reto desarrollar diferentes estrategias de control avanzado que permitan minimizar los riesgos asociados a los problemas mencionados anteriormente. Por esta razón, el desarrollo de algoritmos de control basados en Machine Learning y redes neuronales recurrentes, son una excelente transición de una idea novedosa en el mundo académico a una aplicación rentable en la industria [24].

Wu et al. (2021), plantearon una contribución basada en el modelado de una red neuronal recurrente LSTM y control predictivo de procesos no lineales utilizando un conjunto de datos de entrenamiento ruidosos, donde el ruido puede provenir de diferentes fuentes, como la variabilidad del sensor, y la varianza común de la planta. Las redes de memoria a corto plazo (LSTM) se han utilizado ampliamente para modelar sistemas dinámicos no lineales a partir de procesos que involucran series temporales; el control se realiza bajo un controlador basado en el modelo Lyapunov. Los autores argumentan que para desarrollar un modelo de Machine Learning que pueda incorporarse fácilmente a los controladores (MPC) y proporcionar una predicción precisa de la dinámica del proceso se necesita un conjunto datos de series temporales de alta calidad, por ejemplo, datos de laboratorio, sensores de procesos industriales, o simulaciones informáticas. No obstante, las mediciones industriales suelen incluir ruido, factor determinante y punto crítico poco

[2] Software de simulación computacional utilizado en ingeniería para el análisis de fluidos y transferencia de calor.
[3] Librería de Python utilizada para la computación científica.

abordado en el modelado de Machine Learning. Algunas de las técnicas propuestas para este tipo de datos son el filtro de Kalman y sus modificaciones, el filtro de Savitzky-Golay, que se basan en la suposición del tipo de ruido y la estructura del sistema [25].

Alhajeri et al. (2022), desarrollaron una investigación enfocada en el efecto del ruido gaussiano y no gaussiano en el rendimiento de los modelos de redes neuronales recurrentes basados en estructuras de procesos, que se utilizan para aproximar una clase de sistemas no lineales de múltiples entradas y salidas. Además, utilizaron dos técnicas diferentes, el abandono de Monte Carlo y la co-enseñanza. Estas dos técnicas se emplearon para reducir el sobreajuste en las redes neuronales recurrentes cuando se utilizan datos ruidosos en el proceso de entrenamiento y, por lo tanto, para mejorar la precisión de lazo abierto, así como el rendimiento de lazo cerrado bajo un controlador predictivo de modelo basado en Lyapunov (LMPC). El trabajo presentado por los autores considera el proceso de producción de etilbenceno mediante dos CSTR en serie, y se modela mediante el simulador de dinámica Aspen Plus, confirmando el amplio campo de aplicación que tiene los modelos basados en Machine Learning en la ingeniería química [26].

Así, numerosos trabajos en la literatura investigan métodos de aprendizaje automático con diversos tipos de ruido. En ejemplo, en el trabajo de Hsu y Wang (2009), se evalúa la robustez de un modelo basado en redes neuronales recurrentes de tipo Wiener se evalúa mediante dos tipos de ruido: ruido blanco y ruido sinusoidal [27].

Aunque los modelos basados en redes neuronales recurrentes convencionalmente se han utilizado con éxito en el control predictivo de modelos (MPC) para regular procesos químicos con la precisión de aproximación deseada, el desarrollo de estos modelos puede mejorar estructuralmente mediante la incorporación de conocimientos físicos para lograr una mayor precisión y eficiencia computacional. Alhajeri et al. (2022), plantearon una contribución que investigó el rendimiento de los MPC en dos estructuras diferentes de redes neuronales recurrentes en un proceso de control predictivo de modelos basado en Laypunov (LMPC). Los autores utilizaron un ejemplo de un proceso químico complejo a gran escala simulado por Aspen Plus Dynamics para demostrar mejoras en el modelo basado en redes neuronales recurrentes y el rendimiento de un MPC basado en redes neuronales recurrentes, cuando se tiene en cuenta el conocimiento previo del proceso [28].

Alnajdi et al. (2023), estudiaron sobre el control predictivo de sistemas no lineales con retardo temporal usando modelos basados en Machine Learning. Inicialmente, construyeron un modelo de Machine Learning basado en redes neuronales recurrentes, específicamente un modelo LSTM, que lograra capturar la dinámica del proceso en ausencia de retardos temporales. Posteriormente, crearon un MPC para realizar la estabilización del sistema no lineal sin retardos. Para gestionar los retrasos en las entradas, diseñaron un MPC basado el modelo LSTM con una metodología basada en un modelo de control de Lyapunov (LMPC) basada en retroalimentación del predictor basado en otro modelo LSTM. El predictor lo utilizaron para predecir estados futuros utilizando la medición del proceso y, a continuación, los estados predichos se utilizan para inicializar el MPC basado en el modelo LSTM. Las redes neuronales recurrentes con memoria a corto plazo (LSTM) permite modelar sistemas dinámicos, no lineales y superar los problemas numéricos que suelen presentar las redes neuronales recurrentes tradicionales, pudiéndose aplicar a estrategias de control avanzado, como el control predictivo de modelos (MPC). Los autores usaron un proceso que involucraba un reactor de tanque agitado continuo no isotérmico y bien mezclado (CSTR), con una reacción irreversible, exotérmica y elemental de segundo orden de segundo orden y elemental que transforma un reactivo A en un producto deseado B ($A \rightarrow B$). Las variables controladas son la concentración del reactivo A y la temperatura del reactor y lograron demostrar la capacidad de los modelos para estabilizar el sistema en lazo cerrado con retardos de tiempo [29].

Çıtmacı et al. (2023), desarrollaron un esquema de control multi-entrada y multi-salida (MIMO) para un reactor de electrodo cilíndrico giratorio (RCE), que permite comprender los efectos de la transferencia de masa y la cinética de la reacción de reducción del CO_2. El modelo integró técnicas de modelado de redes neuronales artificiales y recurrentes, optimización no lineal y diseño de controladores de proceso. Las variables controladas fueron las tasas de producción de dos productos del reactor experimental, etileno y monóxido de carbono. Estas, se controlan manipulando dos entradas, el potencial aplicado y la velocidad de rotación del catalizador. Los autores, lograron demostrar el excelente rendimiento del sistema de control en lazo cerrado y la regulación de las salidas en tres puntos de consigna diferentes, incluido un punto de consigna óptimo desde el punto de vista económico y energético [30].

Según lo presentado por Wu et al. (2019), Çıtmacı et al. (2023), y otros autores, se ha demostrado la viabilidad y eficacia de las técnicas de control técnicas de control predictivo neuronal para la identificación y control de sistemas dinámicos no lineales.

Dado al amplio potencial y las exitosas aplicaciones de los modelos basados en Machine Learning y redes neuronales recurrentes en el campo de la ingeniería química, se ha reconocido la necesidad de realizar investigaciones en este ámbito. En este sentido, siguiendo las investigaciones previas realizadas por Wu et al. (2021), Alnajdi et al. (2023), y otros autores, se ha decidido llevar a cabo este trabajo de grado. El objetivo principal es aplicar las ventajas proporcionadas por el Machine Learning para desarrollar un modelo de control predictivo multivariable por seguimiento para el nivel y la presión de un tanque, mediante la creación, el entrenamiento y la validación de un modelo de redes neuronales recurrentes con memoria a corto plazo (LSTM). Se buscará mejorar la precisión y eficiencia en el control de las variables críticas en el proceso. Este enfoque novedoso tiene como propósito contribuir al avance de las técnicas de control avanzado en la ingeniería química y demostrar su aplicabilidad en un caso práctico.

5 Metodología

En este capítulo se presenta la metodología utilizada para desarrollar el control del proceso basado en un modelo de Machine Learning. La **Figura 5** presenta un resumen de las etapas realizadas para la elaboración de un esquema de control por seguimiento con múltiples entradas y salidas (MIMO) de un proceso dinámico no lineal multivariable de nivel y de presión en un tanque que integre un modelo predictivo basado en Machine Learning y redes neuronales recurrentes.

Figura 5

Metodología para la elaboración de un control multivariable basado en Machine Learning y redes neuronales recurrentes

Nota. Elaboración propia.

5.1 Etapa 1: Preparación del proceso

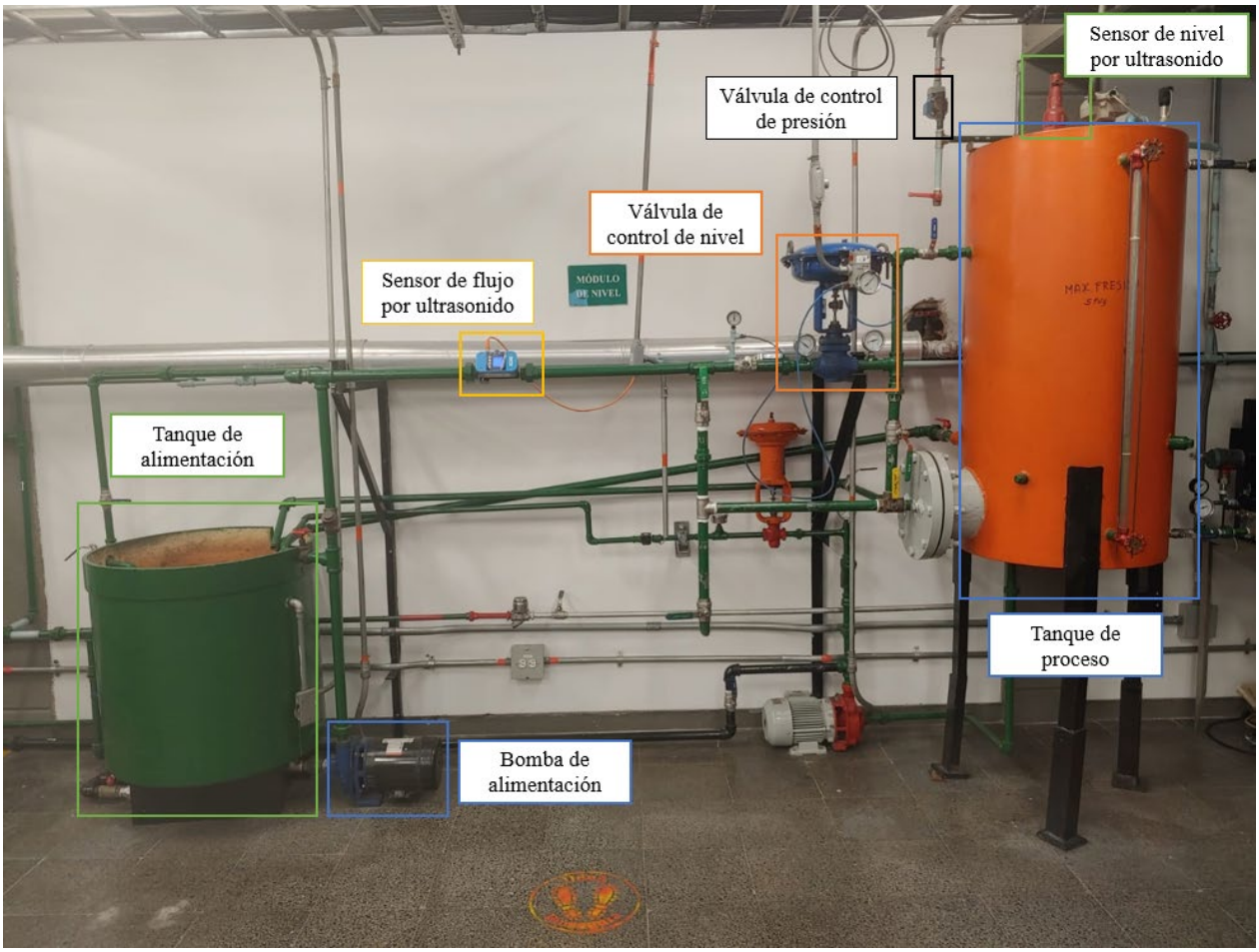
El proceso trabajado es un módulo de nivel ubicado en la sala de control de la Universidad de Antioquia, usado para controlar el nivel de agua y la presión dentro de un tanque mediante una válvula neumática y una válvula solenoide. Como muestra la **Figura 6** el proceso cuenta con una red de tuberías para realizar las conexiones entre el tanque de suministro de agua (verde) y el tanque donde se van a controlar las variables o tanque de proceso (naranja). También, existen diferentes válvulas manuales para configurar específicamente a que altura puede entrar el agua al tanque de proceso y por cuál tubería saldrá el agua desde el tanque de proceso hacía el tanque de suministro. El suministro de agua se realiza mediante el impulso de una bomba EMERSON modelo BM55A, este equipo tiene un rango para la frecuencia de trabajo de 0-60 Hz.

El equipo cuenta con un sensor de flujo por ultrasonido con un rango de 0-60 litros por minuto (lpm). Este mide el flujo de agua que entra al tanque de proceso. Una válvula neumática que trabaja en un rango de 3-15 psi y una válvula solenoide que trabaja en un rango de 4-20 mA. El tanque de proceso cuenta con un sensor de nivel por ultrasonido LU PROBE de SIEMENS. La presión en el tanque de proceso se mide con un transmisor de presión tipo tabaco en un rango de 0-600 mbar. El tanque de proceso tiene un suministro de aire continuo, lo que permite trabajar el tanque presurizado y realizar un control de presión sobre él. Todos estos equipos le envían señales al PLC, permitiendo así realizar el censo y la captura de datos cuando el equipo esté en proceso continuo.

Este proceso cuenta con una automatización ya realizada mediante un PLC SIMATIC S7-300 y el uso de softwares como WinCC y STEP 7. También, cuenta con un aplicativo ya creado que permite realizar el control del nivel y de la presión en el tanque de proceso mediante dos controladores PI. Esto, se realiza manipulando la válvula neumática y la válvula solenoide que regulan la entrada de agua y la salida de aire del tanque de proceso respectivamente. En la **Figura 7** *y* **Figura 8** se evidencia el aplicativo existente y la vista del seguimiento que se le puede realizar a las variables controladas.

Figura 6

Diagrama de equipos del sistema de control multivariable de nivel y de presión en un tanque



Nota. Elaboración propia.

En este trabajo se realizó una modificación al proceso. El valor de referencia de nivel del tanque de proceso no se trabajó como un valor fijo, sino un valor cambiante. Esta variable, sigue la siguiente función dependiente del tiempo:

Ecuación 2.

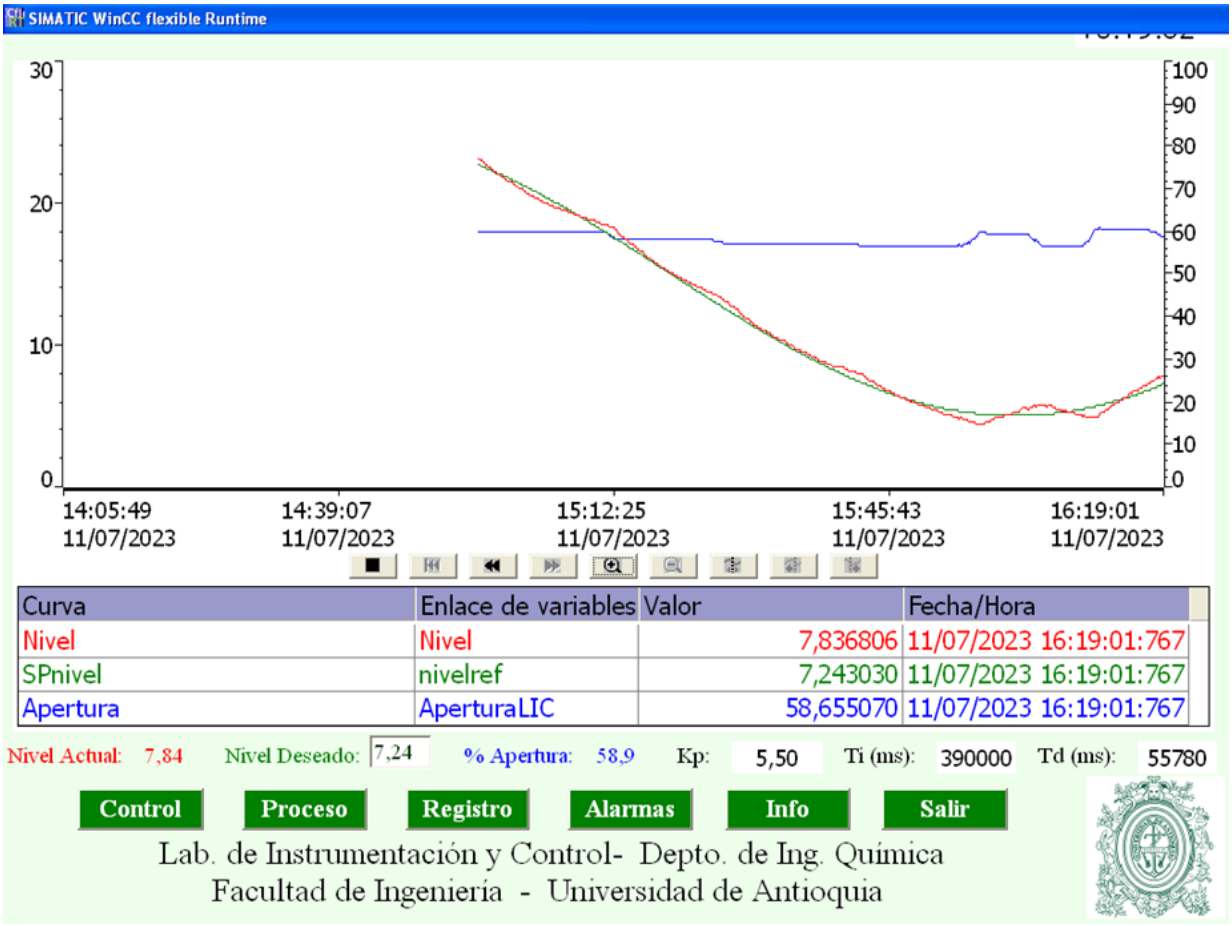
Función del valor de referencia de nivel dependiente del tiempo

$$SP_{nivel}(t) = 10 \cdot \sin\left(\frac{\pi}{2} \cdot \frac{t}{2500}\right) + 15$$

Con esta modificación, el control que se realizó en este trabajo fue un control por seguimiento y no en un valor fijo. Este tipo de control permite estudiar la adaptabilidad a los cambios de un proceso y a diferentes condiciones operativas, optimización del rendimiento, mayor estabilidad y robustez. Esto es beneficioso en procesos químicos donde se requiere una alta precisión y estabilidad en las variables controladas.

Figura 7

Gráfica del seguimiento del valor de referencia de nivel y el nivel real del agua en el tanque de proceso



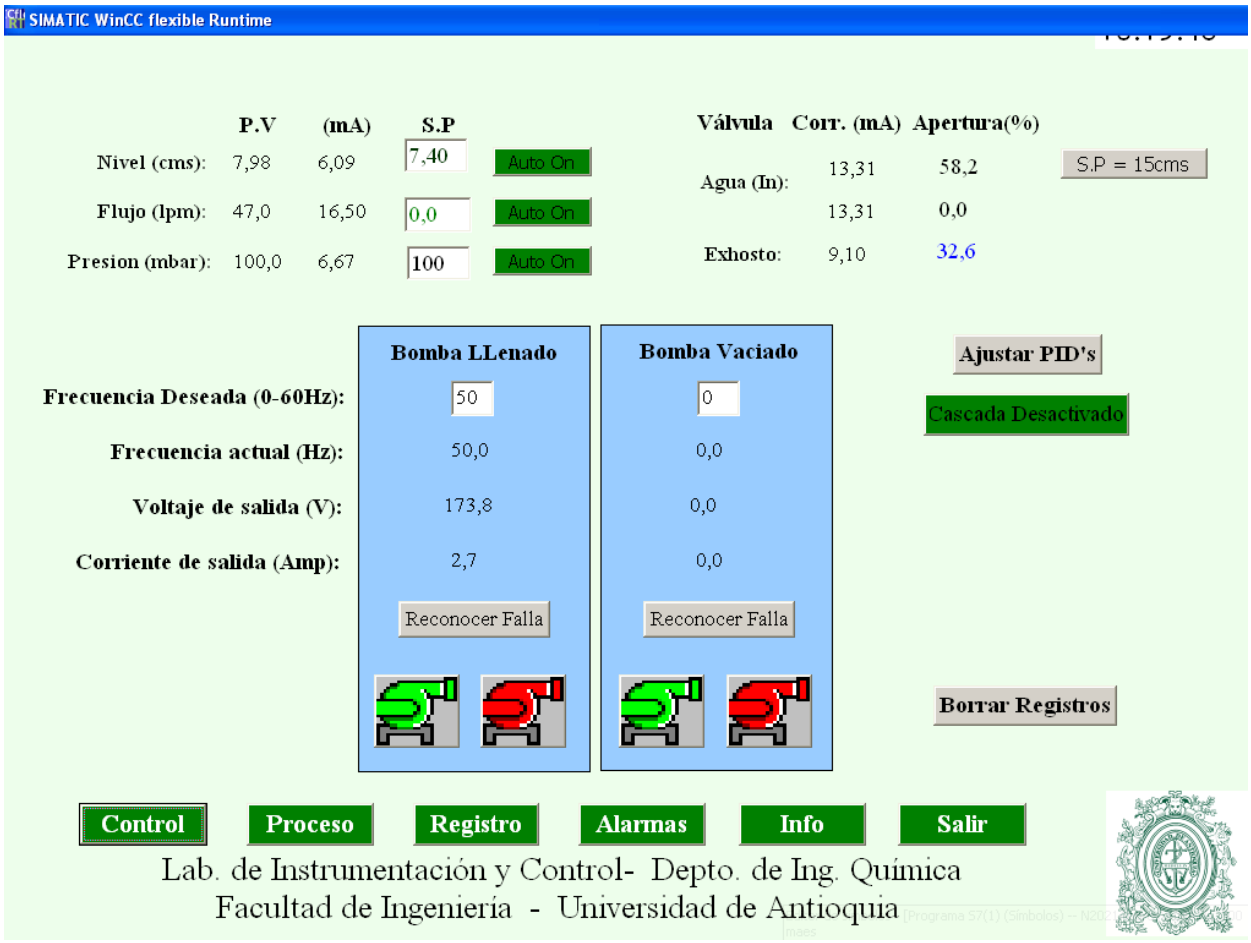
Los controladores que se utilizaron para realizar el control de las variables de interés fueron dos controladores con acción proporcional e integral y previamente sintonizados. Los parámetros de sintonía son: $K_p = 5{,}5$ y $TI = 390000\ ms$, para el controlador que manipula la válvula para el

control de nivel y $K_p = -2{,}0$ y $TI = 20000\ ms$, para el controlador que manipula la válvula para el control de presión. K_p es la ganancia del proceso y TI es el tiempo integral.

Para comenzar con la toma de datos se garantizó que los parámetros de sintonía estuviesen en los valores correspondientes y que estuvieran activados los lazos de control; para esto, los botones mostrados en la **Figura 8** para nivel y para presión, deben estar en automático. Posteriormente, se escogió el valor del valor de referencia fijo para la presión, se le asignó una frecuencia a la bomba de alimentación y se inició el proceso.

Figura 8

Interfaz hombre-maquina (HMI) del PLC y del proceso para el control multivariable de nivel y de presión en un tanque



5.2 Etapa 2: Toma de datos

La toma de datos se realiza usando dos controladores PI que están en funcionamiento en el proceso físico en el laboratorio de Control de Procesos en la Universidad de Antioquia en el módulo de nivel y en lazo cerrado. Esto, se realizó de manera continua a través del PLC y un programa realizado en STEP 7 que permite exportar los ficheros de cada variable, durante cada segundo a un archivo de Excel.

Para realizar la recolección de datos se escogieron las siguientes variables de entrada y salida para la creación del modelo predictivo basado en Machine Learning y redes neuronales recurrentes:

- **Variables de entrada:** valor de referencia de nivel [cm], nivel real del tanque [cm], valor de referencia de presión [mbar], presión real del tanque [mbar] y frecuencia de la bomba de alimentación [Hz].
- **Variables de salida:** porcentaje de apertura de la válvula neumática que controla el nivel del tanque de proceso [%], porcentaje de apertura de la válvula solenoide que controla la presión del tanque de proceso.

Además, se escogieron los siguientes puntos de operación:

Tabla 2
Puntos de operación para la toma de datos

Frecuencia de la bomba de alimentación (Hz)	40 – 42 – 44 – 46 – 48 – 50
Valor de referencia de la presión en el tanque (mbar)	100 – 120 – 140

Nota. Elaboración propia.

Teniendo en cuenta la **Tabla 2**, un experimento para la toma de datos consiste en iniciar el proceso definiendo un valor para la frecuencia de la bomba de alimentación y un valor de referencia para la presión del tanque de proceso. Con esto, se realizó un total de 18 experimentos. Cada experimento tuvo una duración de aproximadamente 3,5 a 4,5 horas de toma de datos. Se tomaron alrededor de 275.000 datos para cada variable sumando todos los experimentos y otros 275.000 datos realizando una réplica para un total de 550.000 datos aproximadamente, disponibles para el

entrenamiento, validación y testeo del modelo. La revisión de los datos tomados se puede realizar en el siguiente enlace: **Datos tomados**.

5.3 Etapa 3: Entramiento de modelos

Inicialmente para la elaboración del modelo se utilizó Visual Studio Code como editor de código y Python como lenguaje de programación. Se realizó un aplicativo utilizando las librerías de Pandas, Numpy, Tensorflow, Sklearn y Keras para el tratamiento y organización de los datos tomados y para la creación, entrenamiento, validación y testeo de diferentes modelos basados en datos.

Se realizó una prueba inicial de creación y entrenamiento utilizando cuatro modelos diferentes uno de regresión lineal, uno de regresión no lineal, uno basado en redes neuronales recurrentes (LSTM) y uno basado en series de tiempo, ARIMA.

Después, se escogió una métrica estadística para realizar la escogencia del modelo que mejor representaba la naturaleza del proceso y de los datos tomados. Finalmente, se analizó la necesidad de la optimización del modelo escogido para obtener la mejor combinación de hiperparámetros y el menor error posible.

5.4 Etapa 4: Optimización del modelo

Una vez se escogió el tipo de modelo, se realizó el procedimiento para la optimización de los hiperparámetros. Los hiperparámetros que se tuvieron en cuenta para el proceso de optimización del modelo fueron, la longitud de la secuencia, el número de neuronas, el número de épocas y el número de lotes.

La optimización se realizó mediante un algoritmo en Python usando librerías como Tensorflow, Pandas, Numpy, Keras, Scikeras y Sklearn; además, se usaron las técnicas de Times Series Split con validación cruzada y una búsqueda de hiperparámetros por cuadrícula (GridSearchCV), estableciendo diferentes niveles para cada hiperparámetro escogido.

5.5 Etapa 5: Prueba del modelo

Para realizar la prueba del modelo predictivo optimizado fue necesario utilizar un OPC donde se configuró el servidor para que el aplicativo desarrollado en Python accediera a él como cliente y pudiera recibir las variables de entrada al modelo, procesarlas y sobre escribir las variables de salida del modelo en el PLC. Las variables de salida que se reescriben en el PLC son los porcentajes de apertura de la válvula de control neumática, encargada del control del nivel de agua en el tanque del proceso y de la válvula solenoide, encargada del control de la presión en el tanque del proceso. Para garantizar que el modelo entrenado se usara de la misma manera a como se entrenó, se creó una ventana deslizante de datos de entrada con la misma longitud de secuencia que se usó en el entrenamiento y validación del modelo, que permitirá el uso de este, en tiempo real y continuamente en el proceso, realizando una realimentación de las variables del proceso y realizando un control predictivo del nivel y la presión del tanque del proceso.

Se realizó la configuración del programa del PLC directamente en el OPC, para así lograr la lectura y escritura de las variables del proceso. Posteriormente, se capturaron estas variables por medio de tags, para que en el aplicativo de Python fuese posible la lectura y escritura de estas.

Al realizar la configuración del servidor, fue necesario realizar la importación del programa del PLC, un archivo con extensión .s7p. Además, se realizó la especificación del tipo de PLC usado y la lista de las variables con las que se trabajó.

Luego, se realizó una conexión Ethernet o por Wifi al servidor OPC desde el PLC y desde el aplicativo de Python. Al realizar la escritura de las variables en el PLC, se buscó realizar el control del nivel y de la presión por medio de un modelo basado en Machine Learning y redes neuronales recurrentes, reemplazando los controladores PI existentes. Finalmente, se realizó una medición del RSME para las variables de salida del modelo que permitió evaluar el rendimiento en las predicciones del modelo y también de las variables controladas con respecto a su valor de referencia para evaluar el rendimiento del control.

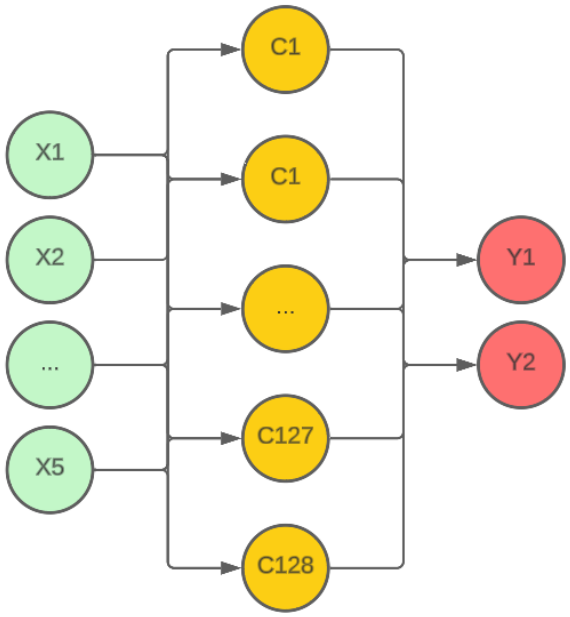
6 Implementación

Los modelos implementados y entrenados con los datos recopilados fueron uno regresión lineal, uno de regresión no lineal, uno basado en redes neuronales recurrentes (LSTM) y uno basado en series de tiempo, ARIMA. Los códigos de estos modelos se muestran en el **Anexo 1.**

La métrica estadística escogida para la selección del modelo fue el error de raíz cuadrada media (RMSE), mostrado en la **Ecuación 1** para realizar la escogencia del modelo que mejor representaba la naturaleza del proceso y de los datos tomados. El modelo escogido fue un modelo basado en redes neuronales recurrentes con memoria a corto plazo (LSTM).

Figura 9

Arquitectura de 3 capas para el modelo basado en redes neuronales recurrentes con memoria a corto plazo



Nota. Las (X) representan las entradas al modelo, las (C) representan las neuronas de la capa intermedia y las (Y) representan las salidas del modelo. Elaboración propia.

Los datos de las variables de entrada y salida del modelo se escalaron entre valores de 0 a 1, se utilizó la técnica de Times Series Split para la creación del conjunto de entrenamiento y el conjunto de validación, teniendo en cuenta la naturaleza temporal de los datos. Posteriormente, se crearon

secuencias de entrenamiento con ventanas deslizantes. El modelo LSTM elaborado, cuenta con una capa LSTM, una capa densa intermedia para permitir que el modelo capture relaciones más complejas y aprenda representaciones más profundas de los datos y una capa densa de salida, una representación esquemática del modelo LSTM elaborado se muestra en la **Figura 9**. La función de pérdida para evaluar el desempeño del modelo es el RMSE. El código del modelo escogido se muestra en el **Anexo 1.**

Teniendo en cuenta pruebas preliminares realizadas, se estableció que el número de separaciones usadas en la técnica Times Series Split será de 5 y se escogió a Adam como el optimizador del modelo, ya que tiene una tasa de aprendizaje adaptativa en todo el proceso de entrenamiento del modelo, como lo recomienda su documentación [5]. Luego, para el proceso de optimización del modelo, se escogió la siguiente cuadrícula de búsqueda para los hiperparámetros que mejor se adapten al proceso en cuestión.

Tabla 3
Cuadrícula de búsqueda para los hiperparámetros óptimos y optimización del modelo escogido

Hiperparámetros	**Niveles**		
Longitud de la secuencia	5	25	125
Número de neuronas	32	64	128
Número de Épocas	10	20	40
Número de lotes	32	64	128

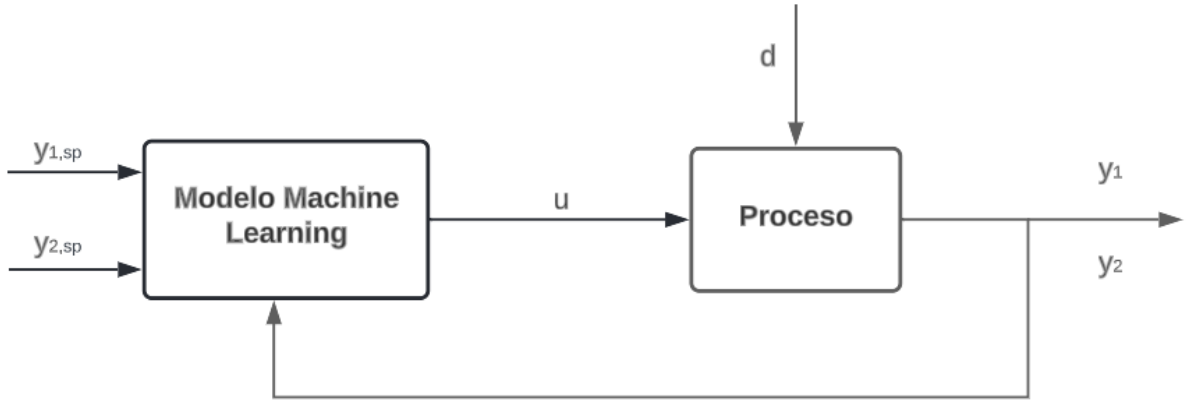
Nota. Elaboración propia.

El código del algoritmo, realizado en Python, utilizado para la optimización de los hiperparámetros se encuentra en el **Anexo 2.** En el **Anexo 3**, se encuentra el código utilizado para realizar la conexión del modelo y la escritura de las variables de salida en el PLC.

El esquema del sistema de control predictivo basado en Machine Learning se presenta en la siguiente figura. El controlador PI tradicional se reemplazó por el modelo basado en redes neuronales recurrentes.

Figura 10

Esquema del sistema de control predictivo multivariable usando un modelo de Machine Learning



Nota. ($y_{1,sp}$ y $y_{2,sp}$) son los valores de referencia para las variables controladas, (y_1 y y_2) son las variables controladas, (u) son las acciones de control hacía las variables manipuladas y d son las perturbaciones del proceso. Elaboración propia.

Para la prueba del modelo, se hizo necesario la instalación de tres librerías específicas en Python relacionadas con la conexión de Python como cliente al servidor OPC creado. Estas librerías fueron asyncua, asyncio y logging [19]. Para las pruebas del modelo se escogieron tres puntos de operación dentro del rango de trabajo establecido en la **Tabla 2** y dos puntos de operación fuera del rango de entrenamiento del modelo para examinar la generalización del modelo y la capacidad de control del nivel y la presión en el tanque en el proceso real, esto se presenta de forma resumida en la **Tabla 4** Cada prueba se realizó en un lapso de aproximadamente dos horas de operación continua.

Tabla 4

Puntos de operación para la prueba del modelo basado en Machine Learning y redes neuronales recurrentes

Puntos de operación	Frecuencia de la bomba [Hz]	Valor de referencia para la presión [mbar]
Prueba 1	45	100
Prueba 2	50	120
Prueba 3	48	140
Prueba 4*	35	80
Prueba 5*	53	160

* Pruebas para los puntos de operación fuera del rango de entrenamiento.

7 Resultados y análisis

En este capítulo se presentan los resultados obtenidos al sintonizar los parámetros de los controladores PI de nivel y de presión, así como los detalles del entrenamiento, la validación y la optimización del modelo de Machine Learning seleccionado. Estos avances representan la implementación de un sistema de control avanzado para el proceso de ingeniería química en cuestión.

7.1 Sintonía de los parámetros

Se realizó el proceso de sintonía para los parámetros de los controladores PI y se obtuvo lo presentado en la siguiente tabla.

Tabla 5
Parámetros de sintonía para el controlador PI

Método	Presión		Nivel	
	K_p	T_I [ms]	K_p	T_I [ms]
Ziegler Nichols de Lazo cerrado	-2,0	20000	5,5	390.000

Nota. K_p es la ganancia del controlador y T_I es el tiempo integral en (ms).

7.2 Datos tomados y modelo seleccionado

La recolección de datos de las variables de entrada y de salida para el entrenamiento, la optimización y la validación del modelo que se obtuvieron, se encuentran en el siguiente enlace disponible para consulta, **Datos tomados**. Estos datos presentan la estructura presentada en la siguiente figura.

Tabla 6

Estructura de los datos recopilados para el entrenamiento y la validación del modelo basado en Machine Learning y redes neuronales recurrentes

h_{PV} [cm]	h_{SP} [cm]	P_{PV} [mbar]	P_{SP} [mbar]	Frecuencia de la Bomba [Hz]	Porcentaje de apertura para la VC de Presion [%]	Porcentaje de apertura para la VC de Nivel [%]
7,872107	15,01382	100	100	40	35,03874	91,20398
7,978009	15,02073	100	100	40	35,00402	91,88036
7,942708	15,02639	100	100	40	34,18805	92,84037
8,048612	15,03267	99,65278	100	40	34,01444	93,51098
8,083912	15,03896	100	100	40	34,58736	93,87999
8,189815	15,04524	100	100	40	34,55263	94,23695
8,260417	15,05152	100	100	40	34,41375	94,97116

El nivel real del agua (h_{PV}) en el equipo y el valor de referencia de nivel (h_{SP}) y la presión en el tanque dependen y varían con el tiempo. El valor de referencia de presión y la frecuencia de la bomba varían según lo presentado en la **Tabla 2**.

Se escogió un modelo que siguiera la naturaleza temporal de los datos y que además en un entrenamiento preliminar, tuviera un RMSE por debajo del 4%. Los resultados del RSME obtenidos para los modelos entrenados se resumen en la siguiente tabla.

Tabla 7

RMSE de posibles modelos para el control multivariable de nivel y de presión trabajado

Modelo	RMSE	
	Presión[1]	Nivel[2]
Linear Regressor	2,913	4,784
Forest Regressor	4,840	8,626
ARIMA	4,892	1,079
LSTM	2,767	3,036

[1] RMSE asociado al porcentaje de apertura de la válvula de control de la presión del sistema.

[2] RMSE asociado al porcentaje de apertura de la válvula de control del nivel del sistema.

Teniendo en cuenta el error de raíz cuadrada media (RMSE) y consideraciones técnicas enfocadas al caso específico de ingeniería química y la naturaleza de los datos, donde el valor de referencia del nivel del tanque de proceso depende y cambia con el tiempo, se escogió elaborar, entrenar y validar un modelo predictivo basado en redes neuronales recurrentes de memoria a corto plazo (LSTM).

El tipo de modelo de Machine Learning que mejor se adaptó a la naturaleza temporal de los datos de nivel y de presión en el tanque fue el modelo basado en redes neuronales recurrentes con memoria a corto plazo con un RMSE de 2,77% para las predicciones realizadas al porcentaje de apertura de la válvula de control de la presión del tanque y 3,04% para las predicciones realizadas al porcentaje de apertura de la válvula de control del nivel del tanque sin optimizar.

La capacidad de los modelos basados en redes neuronales recurrentes para capturar y aprender de las dependencias secuenciales en los datos de tiempo es fundamental en este contexto. Cuando se tiene en cuenta la información pasada en cada paso de tiempo, el modelo puede realizar predicciones más precisas y capturar la dinámica del sistema en estudio. Además, la memoria a corto plazo permite que el modelo retenga información relevante de los pasos de tiempo anteriores, lo cual es especialmente importante en el caso de variables que exhiben cambios progresivos o patrones de fluctuación. Estos resultados indican que el modelo basado en redes neuronales recurrentes con memoria a corto plazo tiene el potencial de mejorar el control de la presión y del nivel en el tanque.

7.3 Entrenamiento, validación y optimización del modelo seleccionado

Posteriormente, en el proceso de entrenamiento y validación del modelo, es necesario realizar una optimización de los hiperparámetros del modelo. La optimización de hiperparámetros que se obtuvo, siguiendo la **Tabla 3** fueron los siguientes:

Tabla 8

Resultados de la optimización de hiperparámetros mediante las técnicas de Times Series Split, Cross Validation y GridSearchCV

Hiperparámetro	Valor
Longitud de la secuencia	25
Número de neuronas	128
Número de Épocas	10
Número de lotes	128

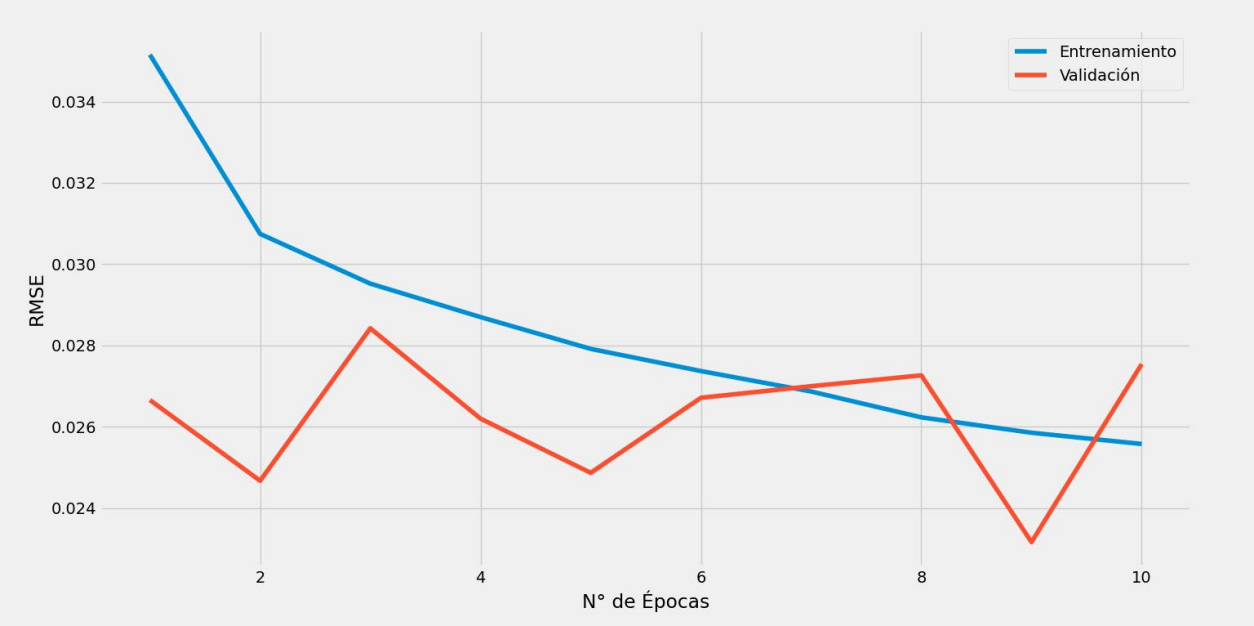
Nota. Elaboración propia.

Esta combinación de hiperparámetros dio como resultado el menor valor de RMSE para el proceso de validación y testeo del modelo de cada variable de salida, 1,90% para el porcentaje de apertura para la válvula de control del nivel de agua en el tanque de proceso y 1,78% para el porcentaje de apertura para la válvula de control de la presión en el tanque de proceso.

Al realizar el entrenamiento final del modelo con los hiperparámetros obtenidos en el proceso de optimización se obtuvieron RMSE por debajo del 4% para la etapa de entrenamiento del modelo por cada época, y por debajo del 3% en la etapa de validación como se evidencia en la **Figura 11**. Como se muestra en **Figura 12**, el modelo predice los datos de salida de manera acertada y con RMSE cercanos al 2% para cada variable.

Figura 11

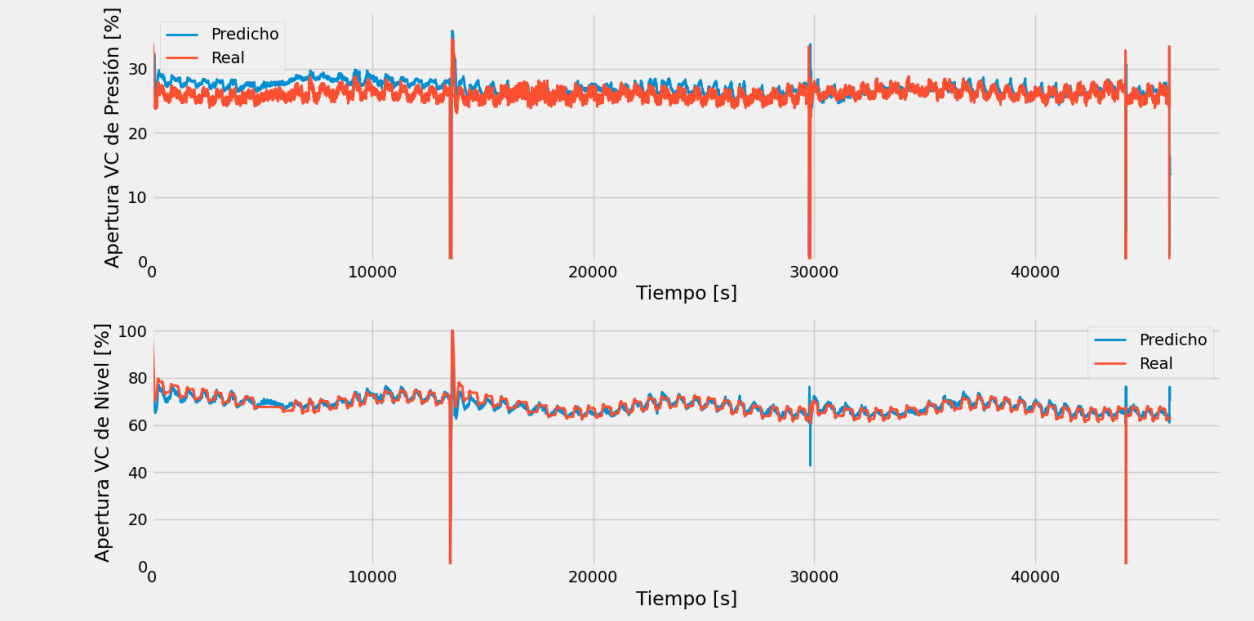
RMSE en cada época en las etapas de entrenamiento y validación del modelo



Nota. Elaboración propia.

Figura 12

Gráfico de las predicciones realizadas por el modelo LSTM para las variables de apertura de las VC para el nivel y para la presión del proceso



Nota. Elaboración propia.

El RMSE obtenido para las predicciones en el porcentaje de apertura para la válvula de control de la presión fue de 1,78% y el para la válvula de control del nivel fue de 1,90%. Luego de realizar el proceso de entrenamiento se realizaron las pruebas de los diferentes escenarios planteados en la **Tabla 4**.

En la **Figura *11*** se evidencia un comportamiento anormal en la gráfica del RSME por época para la etapa de validación del modelo. Desde la primera época se tiene un RMSE por debajo del 2,8%, siendo este un error muy bajo para las primeras épocas en la etapa de validación del modelo. En las etapas posteriores existe una fluctuación del RMSE y no se evidencia una convergencia a un valor estacionario a medida que el número de épocas aumenta. Es importante analizar más a fondo estos resultados y considerar posibles soluciones. Una opción sería ajustar la arquitectura del modelo con respecto al número y tipo de capas usadas, para encontrar un mejor equilibrio entre el ajuste y la generalización. También es recomendable realizar un análisis detallado de los datos de entrenamiento y de validación para identificar posibles diferencias o características distintivas que puedan estar influyendo en el rendimiento del modelo elaborado o aumentar la muestra de datos utilizados para el proceso de la elaboración del modelo.

En la **Figura 13** y **Figura 14** se muestran los resultados del control de nivel y de presión de la prueba realizada a una presión de referencia de 100 mbar y una frecuencia para la bomba de 45 Hz. En este caso, el valor de la frecuencia de la bomba, aunque está dentro del rango establecido en la **Tabla 2** para entrenamiento del modelo, no es un punto específico donde se tomaron datos.

Figura 13

Resultados del control predictivo multivariable por seguimiento de nivel 100 mbar y 45 Hz

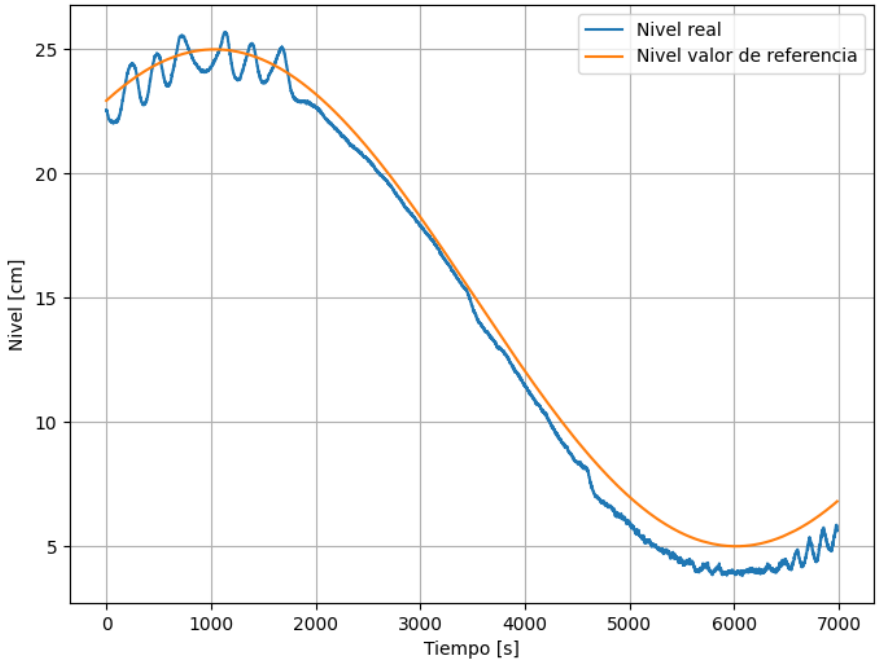
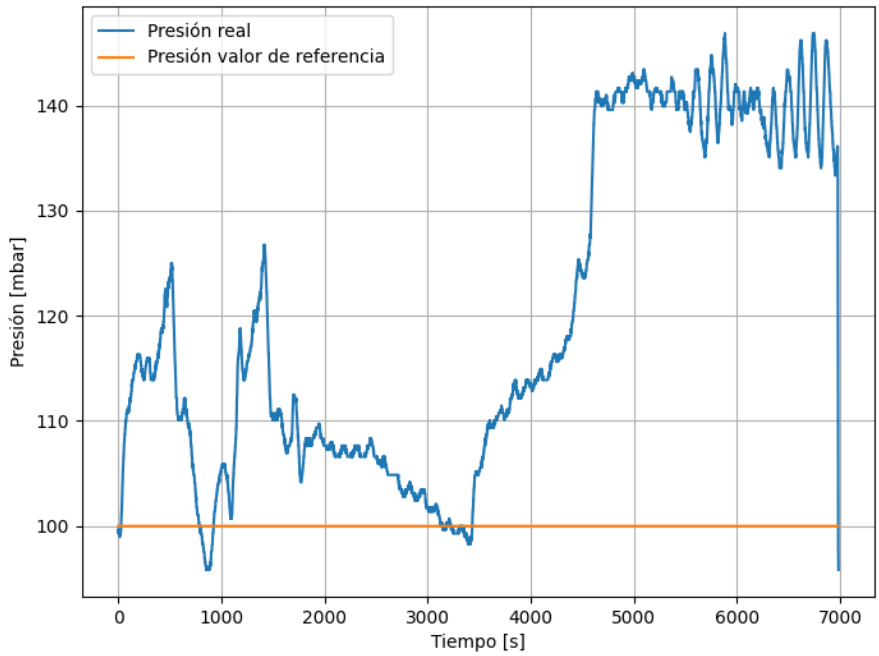


Figura 14

Resultados del control predictivo multivariable de presión a 100 mbar y 45 Hz



En la **Figura *15*** y **Figura *16*** se muestran los resultados del control de nivel y de presión de la prueba realizada a una presión de referencia de 120 mbar y una frecuencia para la bomba de 50 Hz. En la **Figura 17** y **Figura 18** se muestran los resultados del control de nivel y de presión de la prueba realizada a una presión de referencia de 140 mbar y una frecuencia para la bomba de 48 Hz. También se realizaron dos pruebas fuera del rango de entrenamiento mostrado en la **Tabla 2**. En la **Figura 19** y **Figura 20** se muestran los resultados del control de nivel y de presión de la prueba realizada a una presión de referencia de 80 mbar y una frecuencia para la bomba de 35 Hz.

Figura 15

Resultados del control predictivo multivariable por seguimiento de nivel 120 mbar y 50 Hz

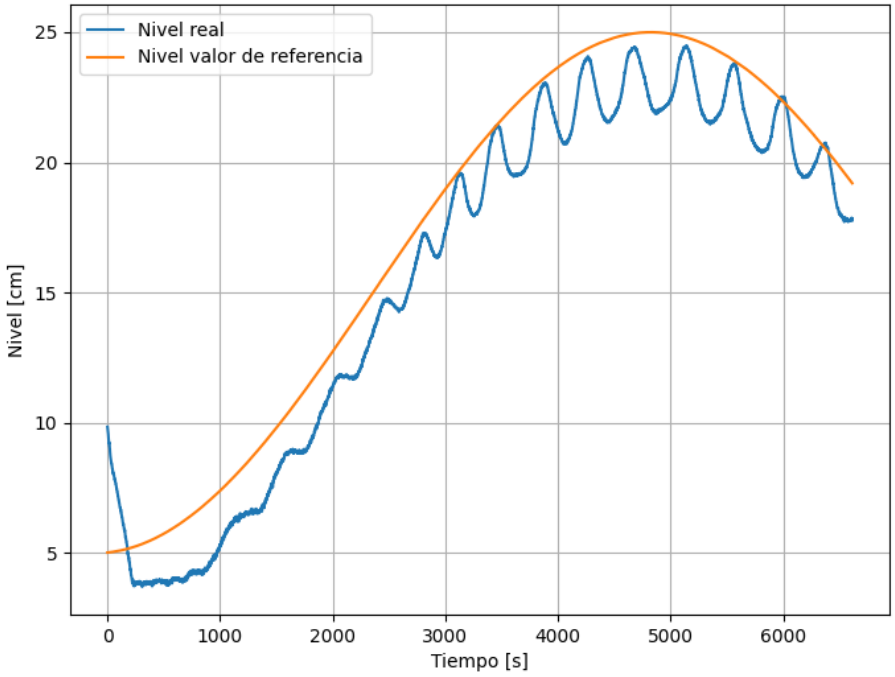


Figura 16

Resultados del control predictivo multivariable de presión a 120 mbar y 50 Hz

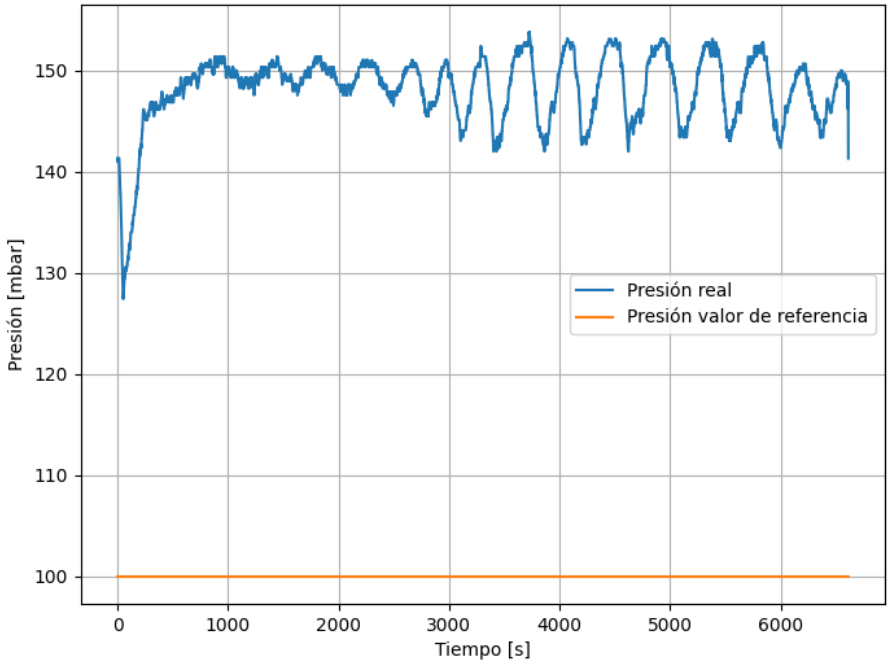


Figura 17

Resultados del control predictivo multivariable por seguimiento de nivel 140 mbar y 48 Hz

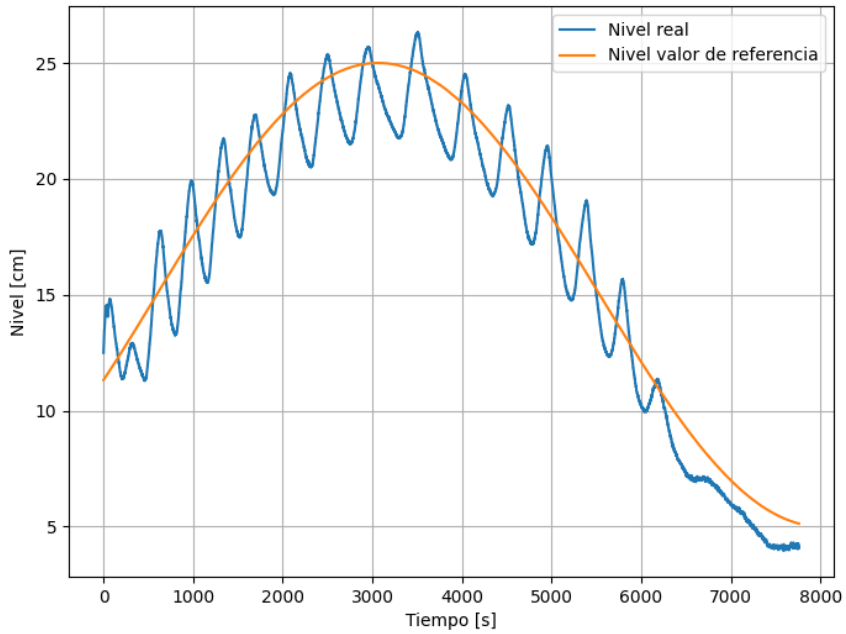


Figura 18

Resultados del control predictivo multivariable de presión a 120 mbar y 48 Hz

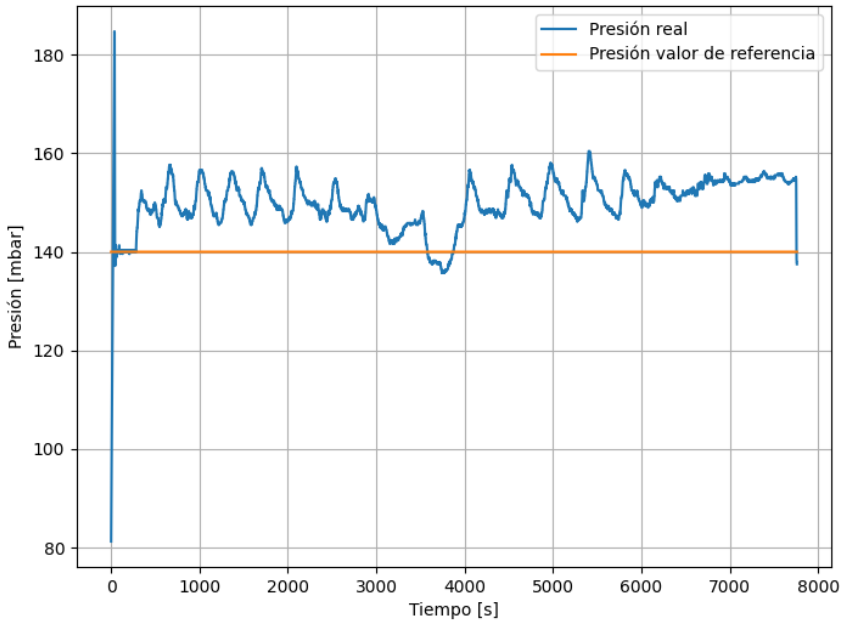


Figura 19

Resultados del control predictivo multivariable por seguimiento de nivel 80 mbar y 35 Hz

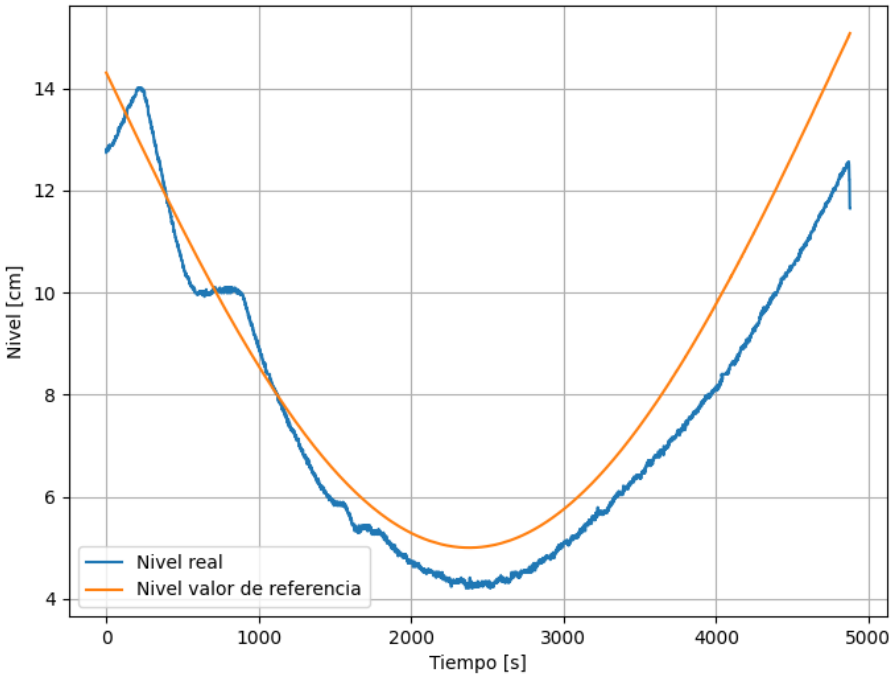


Figura 20

Resultados del control predictivo multivariable de presión 80 mbar y 35 Hz

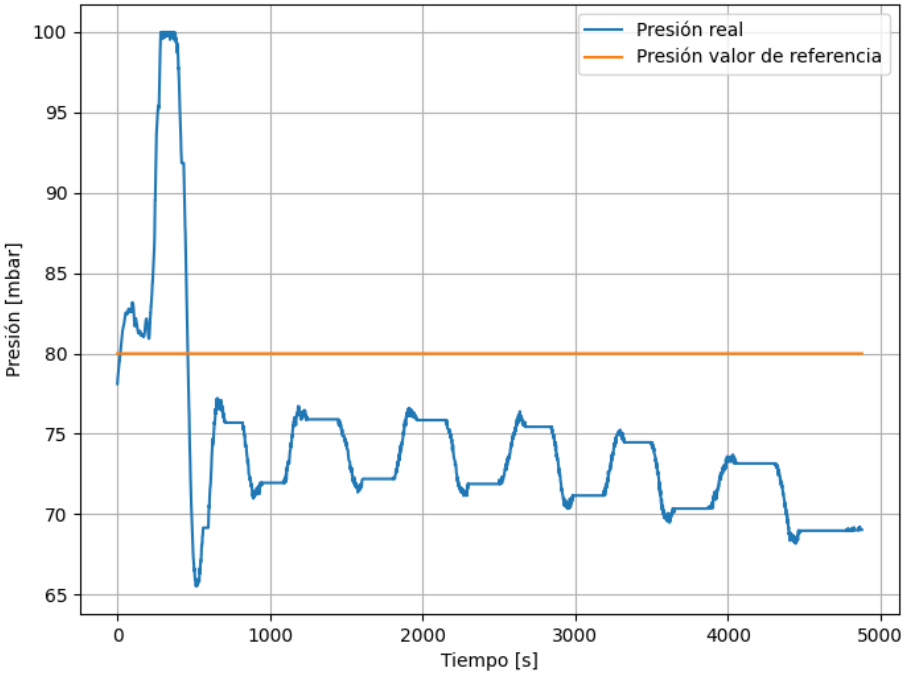


Figura 21

Resultados del control predictivo multivariable por seguimiento de nivel 160 mbar y 53 Hz

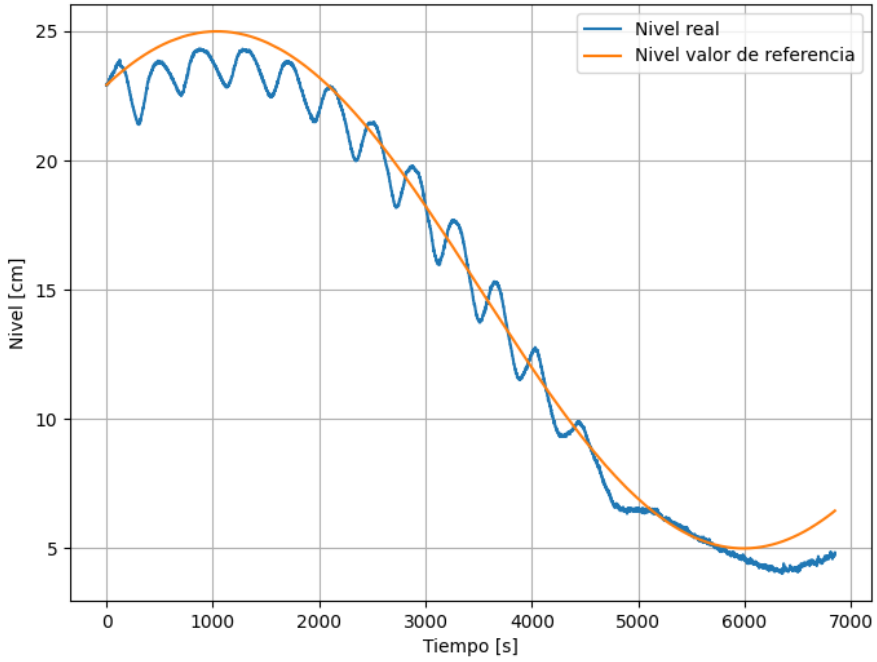
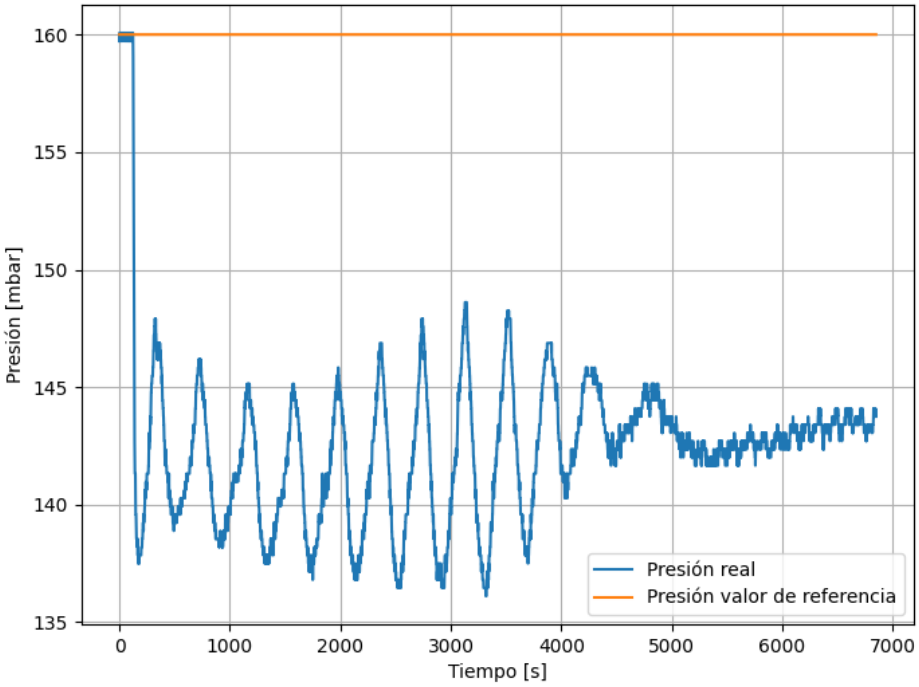


Figura 22

Resultados del control predictivo multivariable de presión 160 mbar y 53 Hz



Se logra realizar el control multivariable del nivel y la presión en ciertos escenarios como se evidencia en la **Figura 15**, **Figura *17***, **Figura *18***, **Figura *19*** y **Figura *21***. A pesar de esto, este control no es óptimo, ya que la variable real y la variable de referencia nunca llegan a mantenerse en el mismo valor. Esto puede indicar que el modelo aún presenta limitaciones o que se requiere un ajuste adicional en los hiperparámetros o la arquitectura, para así lograr mejorar su precisión y capacidad de seguimiento.

Además, se encontraron casos en los que la variable controlada se aleja gradualmente del valor de referencia a medida que pasa el tiempo, como se evidencia en las **Figura *16***, más atrás**Figura *20*** y **Figura *22***. Esto indica que existe una falta de control en estos casos, lo cual puede ser problemático y requerir una mayor investigación para identificar las posibles causas de este comportamiento indeseado. El desarrollo de este trabajo propone que el modelo de Machine Learning aprenda a partir de dos controladores con acción proporciona e integral. A pesar de que el modelo es capaz de realizar predicciones con un RMSE por debajo del 2% en los porcentajes de aperturas de las válvulas de control para el nivel y la presión del tanque, no realiza un control

óptimo en ningún escenario. Por esto, se recomienda realizar una implementación en conjunto con un modelo de control predictivo generalizado (GPC), un sistema de control predictivo de modelos (MPC). El uso de alguna de estas alternativas permitiría aprovechar las capacidades de predicción del modelo basado en Machine Learning y combinarlo con estrategias de control avanzado. Con esto, se lograría ajustar continuamente las acciones de control lo que podría conducir a un control óptimo y preciso de las variables controladas.

Otra alternativa, es la elaboración de dos modelos basados en Machine Learning que permitan minimizar los residuales y predecir el nivel y la presión del tanque por cada paso de la secuencia dadas las condiciones anteriores del sistema. Estos modelos podrían trabajar en conjunto con algoritmos de control adaptativo que ajusten las acciones de control en tiempo real en función de las predicciones generadas por los modelos.

8 Conclusiones

En este trabajo de grado se planteó el problema de determinar si un modelo de Machine Learning basado en datos de regulación por controladores PI era suficiente para lograr un control predictivo multivariable del nivel y la presión en un tanque en un proceso real de ingeniería química. Para lograr este objetivo se planteó una metodología para la toma de 500.000 datos en un proceso de forma continua controlado mediante dos controladores con acción proporcional e integral. Adicionalmente, se construyó y optimizó un modelo de Machine Learning basado en redes neuronales en Python para realizar el control multivariable por seguimiento del nivel y la presión en un tanque en un proceso real.

Los resultados mostraron que el modelo basado en redes neuronales recurrentes con memoria a corto plazo logró capturar las dependencias secuenciales en los datos y realizó predicciones con un RMSE por debajo del 2%, usando una longitud de secuencia de 25, 128 neuronas, 10 épocas y un tamaño de lote de 128, hiperparámetros obtenidos en el proceso de optimización del modelo mediante la técnica de búsqueda por cuadrícula. Entonces, se logró identificar que el estudio y desarrollo de este tipo de modelos son necesarios y útiles en el control de los procesos en la ingeniería química.

Sin embargo, se observó un comportamiento anormal durante la etapa de validación, con fluctuaciones en el RMSE entre la tercera y la novena etapa y falta de convergencia a un valor estacionario. Esto indica la necesidad de un ajuste adicional en la arquitectura o hiperparámetros del modelo, un análisis detallado de los datos de entrenamiento y de validación para identificar posibles diferencias o características distintivas que puedan estar influyendo en el rendimiento del modelo elaborado o aumentar la muestra de los datos utilizados para el proceso de la elaboración del modelo. A pesar de estos desafíos, se confirmó que el modelo basado en redes neuronales recurrentes con memoria a corto plazo tiene el potencial de mejorar el control de la presión y del nivel en el tanque.

Se identificó que existen discrepancias entre las variables reales y las variables de referencia, así como una falta de control en algunos casos donde la variable controlada se aleja gradualmente del valor de referencia a lo largo del tiempo. Por lo tanto, se recomienda combinar el modelo de Machine Learning con estrategias de control adicionales, como un sistema de control

predictivo de modelos (MPC), un control predictivo generalizado (GPC) o la elaboración de dos modelos basados en Machine Learning que permitan minimizar los residuales y predecir el nivel y la presión del tanque por cada paso de la secuencia dadas las condiciones anteriores del sistema. Estos hallazgos resaltan la importancia de ajustar de manera continua las acciones de control para lograr un control óptimo y preciso de las variables controladas en un sistema de ingeniería química.

9 Recomendaciones

Se recomienda apreciar la necesidad de un ajuste adicional en la arquitectura o hiperparámetros del modelo de Machine Learning, así como realizar un análisis detallado de los datos de entrenamiento y de validación para identificar posibles diferencias o características distintivas que puedan estar influyendo en el rendimiento del modelo elaborado o aumentar la muestra de los datos utilizados para el proceso de la elaboración del modelo.

Asimismo, se recomienda combinar el modelo de Machine Learning con estrategias de control adicionales para lograr un control óptimo de las variables controladas. Por ejemplo, considerar la implementación de un sistema de control predictivo de modelos (MPC), un control predictivo generalizado (GPC) o la elaboración de dos modelos basados en Machine Learning que permitan minimizar los residuales y predecir el nivel y la presión del tanque por cada paso de la secuencia dadas las condiciones anteriores del sistema.

Estas recomendaciones buscan mejorar la precisión y rendimiento del modelo, permitiendo aprovechar las fortalezas de las estrategias de control adicionales en conjunto con el modelo de Machine Learning desarrollado. Logrando así, un control predictivo multivariable por seguimiento del nivel y de la presión de un tanque en un proceso real óptimo.

Referencias

[1] "What is Machine Learning? | How it Works, Tutorials, and Examples - MATLAB & Simulink." https://www.mathworks.com/discovery/machine-learning.html (accessed Jun. 24, 2023).

[2] J. P. Janet and H. J. Kulik, *Machine Learning in Chemistry*. Washington, DC, USA: American Chemical Society, 2020.

[3] H. Gao, L. T. Zhu, Z. H. Luo, M. A. Fraga, and I. M. Hsing, "Machine Learning and Data Science in Chemical Engineering," *Ind. Eng. Chem. Res.*, vol. 61, no. 24, pp. 8357–8358, 2022, doi: 10.1021/acs.iecr.2c01788.

[4] "¿Qué es el ajuste de hiperparámetros? - Explicación de los métodos de ajuste de hiperparámetros - AWS." https://aws.amazon.com/es/what-is/hyperparameter-tuning/ (accessed Jul. 10, 2023).

[5] "sklearn.model_selection.TimeSeriesSplit — scikit-learn 1.3.0 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html (accessed Jul. 10, 2023).

[6] "Clasificación de secuencias mediante deep learning - MATLAB & Simulink." https://www.mathworks.com/help/deeplearning/ug/classify-sequence-data-using-lstm-networks_es.html (accessed Jul. 10, 2023).

[7] "función de funciones de pérdida (MicrosoftML) - SQL Server Machine Learning Services | Microsoft Learn." https://learn.microsoft.com/es-es/sql/machine-learning/r/reference/microsoftml/loss?view=sql-server-ver16 (accessed Jul. 10, 2023).

[8] "RMSE (Error cuadrático medio)." https://docs.oracle.com/cloud/help/es/pbcs_common/PFUSU/insights_metrics_RMSE.htm#PFUSU-GUID-FD9381A1-81E1-4F6D-8EC4-82A6CE2A6E74 (accessed Jul. 10, 2023).

[9] "Wincc – Industrias GSL." https://industriasgsl.com/blogs/automatizacion/wincc-siemens (accessed Jul. 10, 2023).

[10] J. J. G. van Merriënboer and P. A. Kirschner, "Step 7," *Ten Steps to Complex Learn.*, pp. 199–225, 2018, doi: 10.4324/9781315113210-10.

[11] C. A. Smith and A. B. Corripio, "Priciples and Practice of a Process Control," *New York*, p.

783, 1997.

[12] V. Mazzone and R. Centrífugo De Watt, "Controladores PID," Accessed: Jul. 10, 2023. [Online]. Available: http://iaci.unq.edu.ar/caut1.

[13] "Qué es OPC y qué es un OPC Server." https://www.kepserverexopc.com/que-es-opc-y-que-es-un-opc-server/ (accessed Jul. 10, 2023).

[14] "The Python Tutorial — Python 3.11.4 documentation." https://docs.python.org/3/tutorial/index.html (accessed Jul. 11, 2023).

[15] "What Is Python Used For? A Beginner's Guide | Coursera." https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python (accessed Jul. 11, 2023).

[16] "pandas - Python Data Analysis Library." https://pandas.pydata.org/about/index.html (accessed Jul. 11, 2023).

[17] "Introduction to NumPy." https://www.w3schools.com/python/numpy/numpy_intro.asp (accessed Jul. 11, 2023).

[18] "Matplotlib — Visualization with Python." https://matplotlib.org/ (accessed Jul. 11, 2023).

[19] "GitHub - tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone." https://github.com/tensorflow/tensorflow (accessed Jul. 10, 2023).

[20] "Scikit Learn Tutorial." https://www.tutorialspoint.com/scikit_learn/index.htm (accessed Jul. 10, 2023).

[21] "Pytorch Vs Tensorflow Vs Keras: Here are the Difference You Should Know." https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article (accessed Jul. 10, 2023).

[22] Z. Wu, A. Tran, D. Rincon, and P. D. Christofides, "Machine learning-based predictive control of nonlinear processes. Part I: Theory," *AIChE J.*, vol. 65, no. 11, Nov. 2019, doi: 10.1002/AIC.16729.

[23] A. K. Singh, H. P. Singh, and S. Mishra, "Validation of ANN-based model for binary distillation column," *Adv. Intell. Syst. Comput.*, vol. 479, pp. 235–242, 2017, doi: 10.1007/978-981-10-1708-7_27.

[24] Z. Wu, A. Tran, Y. M. Ren, C. S. Barnes, S. Chen, and P. D. Christofides, "Model predictive control of phthalic anhydride synthesis in a fixed-bed catalytic reactor via machine learning

modeling," *Chem. Eng. Res. Des.*, vol. 145, pp. 173–183, May 2019, doi: 10.1016/J.CHERD.2019.02.016.

[25] Z. Wu, D. Rincon, J. Luo, and P. D. Christofides, "Handling Noisy Data in Machine Learning Modeling and Predictive Control of Nonlinear Processes," *Proc. Am. Control Conf.*, vol. 2021-May, pp. 3345–3351, May 2021, doi: 10.23919/ACC50511.2021.9483103.

[26] M. S. Alhajeri, F. Abdullah, Z. Wu, and P. D. Christofides, "Physics-informed machine learning modeling for predictive control using noisy data," *Chem. Eng. Res. Des.*, vol. 186, pp. 34–49, Oct. 2022, doi: 10.1016/J.CHERD.2022.07.035.

[27] Y. L. Hsu and J. S. Wang, "A Wiener-type recurrent neural network and its control strategy for nonlinear dynamic applications," *J. Process Control*, vol. 19, no. 6, pp. 942–953, Jun. 2009, doi: 10.1016/J.JPROCONT.2008.12.002.

[28] M. S. Alhajeri, J. Luo, Z. Wu, F. Albalawi, and P. D. Christofides, "Process structure-based recurrent neural network modeling for predictive control: A comparative study," *Chem. Eng. Res. Des.*, vol. 179, pp. 77–89, Mar. 2022, doi: 10.1016/J.CHERD.2021.12.046.

[29] A. Alnajdi, A. Suryavanshi, M. S. Alhajeri, F. Abdullah, and P. D. Christofides, "Machine learning-based predictive control of nonlinear time-delay systems: Closed-loop stability and input delay compensation," *Digit. Chem. Eng.*, vol. 7, p. 100084, Jun. 2023, doi: 10.1016/J.DCHE.2023.100084.

[30] B. Çıtmacı, J. Luo, J. B. Jang, C. G. Morales-Guio, and P. D. Christofides, "Machine learning-based ethylene and carbon monoxide estimation, real-time optimization, and multivariable feedback control of an experimental electrochemical reactor," *Chem. Eng. Res. Des.*, vol. 191, pp. 658–681, Mar. 2023, doi: 10.1016/J.CHERD.2023.02.003.

[31] "sklearn.linear_model.LinearRegression — scikit-learn 1.3.0 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (accessed Jul. 18, 2023).

[32] "sklearn.ensemble.RandomForestRegressor — scikit-learn 1.3.0 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html (accessed Jul. 18, 2023).

[33] G. Edward and G. Meirion Jenkins, "SERIES TEMPORALES, MODELO ARIMA

METODOLOGÍA DE BOX-JENKINS."

Anexos

Anexo 1. Modelos entrenados

Linear Regression

Linear Regression ajusta a un modelo lineal con coeficientes $w = (w_1, \ldots, w_p)$ para minimizar la suma residual de cuadrados entre los observados en el conjunto de datos y los objetivos predichos por la aproximación lineal [31].

- **Código en Python:**

```
# Importar librerías
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Configurar el estilo de las gráficas
plt.style.use('fivethirtyeight')

# Ruta del archivo
rutaArchivo = './Datos.xlsx'

# Carga los datos desde el archivo Excel en un DataFrame de pandas
datos = pd.read_excel(rutaArchivo, decimal=',')

# Tratar valores faltantes
datos = datos.dropna()

# Definir las columnas de entrada y salida:
input_columns = ['hPV [cm]', 'hSP [cm]', 'PPV [mbar]',
                 'PSP [mbar]', 'Frecuencia de la Bomba [Hz]']
output_columns = ['% Apertura Presion', '% Apertura Nivel']

def Escalizado(datos, columna, rango_original, rango_deseado):
    datos[columna] = (datos[columna] - rango_original[0]) / (rango_original[1]
-

rango_original[0]) * (rango_deseado[1] - rango_deseado[0]) + rango_deseado[0]
```

```
    return datos[columna]

# Escalizar los datos de las variables de entrada
datos['hPV [cm]'] = Escalizado(datos, 'hPV [cm]', [0, 60], [0, 1])
datos['hSP [cm]'] = Escalizado(datos, 'hSP [cm]', [0, 60], [0, 1])
datos['PPV [mbar]'] = Escalizado(datos, 'PPV [mbar]', [0, 600], [0, 1])
datos['PSP [mbar]'] = Escalizado(datos, 'PSP [mbar]', [0, 600], [0, 1])
datos['Frecuencia de la Bomba [Hz]'] = Escalizado(
    datos, 'Frecuencia de la Bomba [Hz]', [0, 60], [0, 1])

# Escalizar los datos de las variables de salida
datos['% Apertura Presion'] = Escalizado(
    datos, '% Apertura Presion', [0, 100], [0, 1])
datos['% Apertura Nivel'] = Escalizado(
    datos, '% Apertura Nivel', [0, 100], [0, 1])

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(datos[input_columns],
datos[output_columns], test_size=0.2, random_state=200)

# Transformar datos en tensores
X_train_tensor = tf.convert_to_tensor(X_train, dtype=tf.float32)
y_train_tensor = tf.convert_to_tensor(y_train, dtype=tf.float32)

X_test_tensor = tf.convert_to_tensor(X_test, dtype=tf.float32)
y_test_tensor = tf.convert_to_tensor(y_test, dtype=tf.float32)

# Crear el modelo de regresión multivariable
model = LinearRegression()

# Entrenar el modelo
model.fit(X_train_tensor, y_train_tensor)

# Obtener las predicciones del modelo en el conjunto de prueba
y_pred = model.predict(X_test_tensor)

# Evaluar el modelo en el conjunto de prueba
score = model.score(X_test_tensor, y_pred)
print("Coeficiente de determinación R^2:", score)

# Calcular el RMSE en el conjunto de prueba
rmse = np.sqrt(mean_squared_error(y_test_tensor, y_pred))
print("RMSE en el conjunto de prueba:", rmse*100)

# Calcular el RMSE para cada salida
rmse_1 = np.sqrt(mean_squared_error(y_test_tensor[:, 0], y_pred[:, 0]))
rmse_2 = np.sqrt(mean_squared_error(y_test_tensor[:, 1], y_pred[:, 1]))

# Imprimir los resultados
```

```
print("RMSE para Apertura VC de Presión:", rmse_1*100, "%.")
print("RMSE para Apertura VC de Nivel:", rmse_2*100, "%.")

# Desnormalizar las predicciones y los datos de prueba. Limitar a valores
entre 0 y 100
y_pred = y_pred * 100
y_test = y_test_tensor * 100

# Crear un DataFrame con los datos predichos y los datos de prueba
df_pred = pd.DataFrame({'Esperado_1': y_test[:, 0], 'Predicho_1': y_pred[:,
0],
                        'Esperado_2': y_test[:, 1], 'Predicho_2': y_pred[:,
1]})

# Guardar los datos en un archivo Excel
df_pred.to_excel('datos_predichos.xlsx', index=False)

# Graficar los resultados de las salidas
# Obtener las salidas predichas
pred_1 = y_pred[:, 0]
pred_2 = y_pred[:, 1]

# Obtener las salidas reales
real_1 = y_test[:, 0]
real_2 = y_test[:, 1]

# Crear una figura con dos subtramas
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10))

# Graficar la salida 1
ax1.plot(pred_1, label='Predicho', linewidth=1)
ax1.plot(real_1, label='Real', linewidth=1)
ax1.set_xlabel('Tiempo [s]')
ax1.set_ylabel('Apertura VC de Presión [%]')
ax1.legend()

# Graficar la salida 2
ax2.plot(pred_2, label='Predicho', linewidth=1)
ax2.plot(real_2, label='Real', linewidth=1)
ax2.set_xlabel('Tiempo [s]')
ax2.set_ylabel('Apertura VC de Nivel [%]')
ax2.legend()

# Ajustar los márgenes y espaciado
fig.tight_layout()

plt.savefig('grafica_predicciones.png')
# Mostrar la figura
plt.show()
```

Random Forest Regressor

Random Forest Regressor es un meta estimador que ajusta una serie de árboles de decisión clasificatorios en varias submuestras del conjunto de datos y utiliza el promedio para mejorar la precisión predictiva y controlar el sobreajuste [32].

- **Código en Python:**

```
# Importar librerías
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Configurar el estilo de las gráficas
plt.style.use('fivethirtyeight')

# Ruta del archivo
rutaArchivo = './Datos.xlsx'

# Carga los datos desde el archivo Excel en un DataFrame de pandas
datos = pd.read_excel(rutaArchivo, decimal=',')

# Tratar valores faltantes
datos = datos.dropna()

# Definir las columnas de entrada y salida:
input_columns = ['hPV [cm]', 'hSP [cm]', 'PPV [mbar]',
                 'PSP [mbar]', 'Frecuencia de la Bomba [Hz]']
output_columns = ['% Apertura Presion', '% Apertura Nivel']


def Escalizado(datos, columna, rango_original, rango_deseado):
    datos[columna] = (datos[columna] - rango_original[0]) / (rango_original[1]
-

rango_original[0]) * (rango_deseado[1] - rango_deseado[0]) + rango_deseado[0]

    return datos[columna]

# Escalizar los datos de las variables de entrada
datos['hPV [cm]'] = Escalizado(datos, 'hPV [cm]', [0, 60], [0, 1])
datos['hSP [cm]'] = Escalizado(datos, 'hSP [cm]', [0, 60], [0, 1])
datos['PPV [mbar]'] = Escalizado(datos, 'PPV [mbar]', [0, 600], [0, 1])
datos['PSP [mbar]'] = Escalizado(datos, 'PSP [mbar]', [0, 600], [0, 1])
```

```
datos['Frecuencia de la Bomba [Hz]'] = Escalizado(
    datos, 'Frecuencia de la Bomba [Hz]', [0, 60], [0, 1])

# Escalizar los datos de las variables de salida
datos['% Apertura Presion'] = Escalizado(
    datos, '% Apertura Presion', [0, 100], [0, 1])
datos['% Apertura Nivel'] = Escalizado(
    datos, '% Apertura Nivel', [0, 100], [0, 1])

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(datos[input_columns],
datos[output_columns], test_size=0.2, random_state=200)

# Transformar datos en tensores
X_train_tensor = tf.convert_to_tensor(X_train, dtype=tf.float32)
y_train_tensor = tf.convert_to_tensor(y_train, dtype=tf.float32)

X_test_tensor = tf.convert_to_tensor(X_test, dtype=tf.float32)
y_test_tensor = tf.convert_to_tensor(y_test, dtype=tf.float32)

# Crear el modelo de Bosque Aleatorio
model = RandomForestRegressor(n_estimators=100, random_state=200)

# Entrenar el modelo
model.fit(X_train_tensor, y_train_tensor)

# Obtener las predicciones del modelo en el conjunto de prueba
y_pred = model.predict(X_test_tensor)

# Evaluar el modelo en el conjunto de prueba
score = model.score(X_test_tensor, y_pred)
print("Coeficiente de determinación R^2:", score)

# Calcular el RMSE en el conjunto de prueba
rmse = np.sqrt(mean_squared_error(y_test_tensor, y_pred))
print("RMSE en el conjunto de prueba:", rmse*100, "%.")

# Calcular el RMSE para cada salida
rmse_1 = np.sqrt(mean_squared_error(y_test_tensor[:, 0], y_pred[:, 0]))
rmse_2 = np.sqrt(mean_squared_error(y_test_tensor[:, 1], y_pred[:, 1]))

# Imprimir los resultados
print("RMSE para Apertura VC de Presión:", rmse_1*100, "%.")
print("RMSE para Apertura VC de Nivel:", rmse_2*100, "%.")

# Desnormalizar las predicciones y los datos de prueba. Limitar a valores
entre 0 y 100
y_pred = y_pred * 100
y_test = y_test_tensor * 100
```

```
# Crear un DataFrame con los datos predichos y los datos de prueba
df_pred = pd.DataFrame({'Esperado_1': y_test[:, 0], 'Predicho_1': y_pred[:,
0],
                        'Esperado_2': y_test[:, 1], 'Predicho_2': y_pred[:,
1]})

# Guardar los datos en un archivo Excel
df_pred.to_excel('datos_predichos.xlsx', index=False)

# Graficar los resultados de las salidas
# Obtener las salidas predichas
pred_1 = y_pred[:, 0]
pred_2 = y_pred[:, 1]

# Obtener las salidas reales
real_1 = y_test[:, 0]
real_2 = y_test[:, 1]

# Crear una figura con dos subtramas
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10))

# Graficar la salida 1
ax1.plot(pred_1, label='Predicho', linewidth=1)
ax1.plot(real_1, label='Real', linewidth=1)
ax1.set_xlabel('Tiempo [s]')
ax1.set_ylabel('Apertura VC de Presión [%]')
ax1.legend()

# Graficar la salida 2
ax2.plot(pred_2, label='Predicho', linewidth=1)
ax2.plot(real_2, label='Real', linewidth=1)
ax2.set_xlabel('Tiempo [s]')
ax2.set_ylabel('Apertura VC de Nivel [%]')
ax2.legend()

# Ajustar los márgenes y espaciado
fig.tight_layout()

plt.savefig('grafica_predicciones.png')
# Mostrar la figura
plt.show()
```

ARIMA

El modelo ARIMA permite describir un valor como una función lineal de datos anteriores y errores debidos al azar, además, puede incluir un componente cíclico o estacional. Es decir, debe contener todos los elementos necesarios para describir el fenómeno. Box y Jenkins recomiendan como mínimo 50 observaciones en la serie temporal [33].

- **Código en Python:**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.model_selection import train_test_split, TimeSeriesSplit

# Configurar el estilo de las gráficas
plt.style.use('fivethirtyeight')

# Ruta del archivo
rutaArchivo = './Datos.xlsx'

# Carga los datos desde el archivo Excel en un DataFrame de pandas
datos = pd.read_excel(rutaArchivo, decimal=',')

# Tratar valores faltantes
datos = datos.dropna()

# Definir las columnas de entrada y salida:
input_columns = ['hPV [cm]', 'hSP [cm]', 'PPV [mbar]', 'PSP [mbar]',
'Frecuencia de la Bomba [Hz]']
output_columns = ['% Apertura Presion', '% Apertura Nivel']

def Escalizado(datos, columna, rango_original, rango_deseado):
    datos[columna] = (datos[columna] - rango_original[0]) / (rango_original[1]
- rango_original[0]) * (
            rango_deseado[1] - rango_deseado[0]) + rango_deseado[0]
    return datos[columna]

# Escalizar los datos de las variables de entrada
datos['hPV [cm]'] = Escalizado(datos, 'hPV [cm]', [0, 60], [0, 1])
datos['hSP [cm]'] = Escalizado(datos, 'hSP [cm]', [0, 60], [0, 1])
datos['PPV [mbar]'] = Escalizado(datos, 'PPV [mbar]', [0, 600], [0, 1])
datos['PSP [mbar]'] = Escalizado(datos, 'PSP [mbar]', [0, 600], [0, 1])
datos['Frecuencia de la Bomba [Hz]'] = Escalizado(datos, 'Frecuencia de la
Bomba [Hz]', [0, 60], [0, 1])

# Escalizar los datos de las variables de salida
datos['% Apertura Presion'] = Escalizado(datos, '% Apertura Presion', [0,
100], [0, 1])
```

```
datos['% Apertura Nivel'] = Escalizado(datos, '% Apertura Nivel', [0, 100],
[0, 1])

# División de conjuntos de datos con TimeSeriesSplit
X_train_list = []
X_val_test_list = []

# Dividir el conjunto de datos en 5 partes
tscv = TimeSeriesSplit(n_splits=5)  # Número de divisiones
for train_index, test_index in tscv.split(datos):
    X_train_list.append(datos.iloc[train_index])
    X_val_test_list.append(datos.iloc[test_index])

# Unir el conjunto de entrenamiento.
X_train = pd.concat(X_train_list)
X_val_test = pd.concat(X_val_test_list)

# Dividir el conjunto de validación y prueba de forma equitativa y sin
barajear.
X_val, X_test = train_test_split(X_val_test, test_size=0.5, shuffle=False)

# Definir la longitud de la secuencia de tiempo
sequence_length = 10  # Ajusta este valor según tus necesidades

# Definir una función para crear secuencias de longitud sequence_length
def create_sequences(data, sequence_length):
    X = []
    y = []
    for i in range(len(data) - sequence_length):
        X.append(data.iloc[i:i + sequence_length][input_columns].values)
        y.append(data.iloc[i + sequence_length][output_columns].values)
    return np.array(X), np.array(y)

# Crear secuencias de entrenamiento
X_train_input, y_train = create_sequences(X_train, sequence_length)

# Crear secuencias de validación
X_val_input, y_val = create_sequences(X_val, sequence_length)

# Crear secuencias de prueba
X_test_input, y_test = create_sequences(X_test, sequence_length)

# Crear un DataFrame con los datos de entrenamiento
df_train = pd.DataFrame(X_train_input[:, -1, :], columns=input_columns)

# Crear un DataFrame con los datos de validación
df_val = pd.DataFrame(X_val_input[:, -1, :], columns=input_columns)

# Crear un DataFrame con los datos de prueba
df_test = pd.DataFrame(X_test_input[:, -1, :], columns=input_columns)
```

```
# Entrenar y evaluar el modelo ARIMA para cada columna de salida
for column in output_columns:
    print(f"Entrenando y evaluando el modelo ARIMA para '{column}'...")

    # Obtener los datos de entrenamiento y validación para la columna actual
    y_train_column = y_train[:, output_columns.index(column)]
    y_val_column = y_val[:, output_columns.index(column)]

    # Crear el modelo ARIMA
    model = ARIMA(y_train_column, order=(1, 1, 1))  # Ajusta los parámetros
del modelo ARIMA según tus necesidades

    # Ajustar el modelo ARIMA
    model_fit = model.fit()

    # Realizar predicciones en los datos de validación
    predictions_val = model_fit.predict(start=len(y_train_column),
end=len(y_train_column) + len(y_val_column) - 1)

    # Realizar predicciones en los datos de prueba
    predictions_test = model_fit.predict(start=len(y_train_column) +
len(y_val_column),
                                         end=len(y_train_column) +
len(y_val_column) + len(y_test[:, 0]) - 1)*100


    # Calcular el error RMSE en los datos de prueba
    rmse_test = np.sqrt(np.mean((predictions_test - 100*y_test[:,
output_columns.index(column)]) ** 2))
    print(f"Error RMSE en los datos de prueba para '{column}': {rmse_test}")

    # Graficar las predicciones y los valores reales para los datos de prueba.
    plt.figure(figsize=(8, 6))
    plt.plot(range(len(y_test[:, output_columns.index(column)])),
100*y_test[:, output_columns.index(column)], label='Real')
    plt.plot(range(len(y_test[:, output_columns.index(column)])),
predictions_test, label='Predicción')
    plt.title(f"Pred. ARIMA para '{column}' - Prueb.")
    plt.xlabel('Tiempo')
    plt.ylabel(column)
    plt.legend()
    plt.savefig(f'grafica_predicciones_{column}_test.png')
    plt.show()
```

Redes neuronales recurrentes (LSTM)

Las redes neuronales recurrentes son una forma de codificar esta naturaleza secuencial. Cuando los datos secuenciales tienen relaciones a largo plazo en secuencias largas y es necesario hacer predicciones de series temporales, es recomendado usar las redes neuronales recurrentes LSTM (Long-Short-Term Memory). Este tipo de redes neuronales tiene tres componentes principales: una celda de memoria, una puerta de entrada y una puerta de salida[2].

Las redes neuronales LSTM utilizan datos secuenciales y/o datos temporales como los que se recolectan, por ejemplo, en un proceso que involucra el control multivariable por seguimiento en la ingeniería química, usando la automatización de procesos y los controladores lógicos programables (PLC).

- **Código en Python:**

```
# Importar librerías
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split, TimeSeriesSplit
from sklearn.metrics import mean_squared_error

# Configurar el estilo de las gráficas
plt.style.use('fivethirtyeight')

# Ruta del archivo
rutaArchivo = './Datos.xlsx'

# Carga los datos desde el archivo Excel en un DataFrame de pandas
datos = pd.read_excel(rutaArchivo, decimal=',')

# Tratar valores faltantes
datos = datos.dropna()

# Definir las columnas de entrada y salida:
input_columns = ['hPV [cm]', 'hSP [cm]', 'PPV [mbar]',
                 'PSP [mbar]', 'Frecuencia de la Bomba [Hz]']
output_columns = ['% Apertura Presion', '% Apertura Nivel']


def Escalizado(datos, columna, rango_original, rango_deseado):
    datos[columna] = (datos[columna] - rango_original[0]) / (rango_original[1]
-
```

```
rango_original[0]) * (rango_deseado[1] - rango_deseado[0]) + rango_deseado[0]

    return datos[columna]


# Escalizar los datos de las variables de entrada
datos['hPV [cm]'] = Escalizado(datos, 'hPV [cm]', [0, 60], [0, 1])
datos['hSP [cm]'] = Escalizado(datos, 'hSP [cm]', [0, 60], [0, 1])
datos['PPV [mbar]'] = Escalizado(datos, 'PPV [mbar]', [0, 600], [0, 1])
datos['PSP [mbar]'] = Escalizado(datos, 'PSP [mbar]', [0, 600], [0, 1])
datos['Frecuencia de la Bomba [Hz]'] = Escalizado(
    datos, 'Frecuencia de la Bomba [Hz]', [0, 60], [0, 1])

# Escalizar los datos de las variables de salida
datos['% Apertura Presion'] = Escalizado(
    datos, '% Apertura Presion', [0, 100], [0, 1])
datos['% Apertura Nivel'] = Escalizado(
    datos, '% Apertura Nivel', [0, 100], [0, 1])

# División de conjuntos de datos con TimeSeriesSplit
X_train_list = []
X_val_test_list = []


# Dividir el conjunto de datos en 5 partes
tscv = TimeSeriesSplit(n_splits=5)  # Número de divisiones
for train_index, test_index in tscv.split(datos):
    X_train_list.append(datos.iloc[train_index])
    X_val_test_list.append(datos.iloc[test_index])

# Escoger el conjunto de entrenamiento. En este caso, el último.
X_train = X_train_list[4]
X_val_test = X_val_test_list[4]

# Dividir el conjunto de validación y prueba de forma equitativa y sin
barajear.
X_val, X_test = train_test_split(X_val_test, test_size=0.5, shuffle=False)

# Definir la longitud de la secuencia de tiempo
sequence_length = 25  # Ajusta este valor según tus necesidades

# Definir una función para crear secuencias de longitud sequence_length
def create_sequences(data, sequence_length):
    X = []
    y = []
    for i in range(len(data) - sequence_length):
        X.append(data.iloc[i:i+sequence_length][input_columns].values)
        y.append(data.iloc[i+sequence_length][output_columns].values)
    return np.array(X), np.array(y)
```

```
# Crear secuencias de entrenamiento
X_train_input, y_train = create_sequences(X_train, sequence_length)

# Crear secuencias de validación
X_val_input, y_val = create_sequences(X_val, sequence_length)

# Crear secuencias de prueba
X_test_input, y_test = create_sequences(X_test, sequence_length)

# Transformar datos en tensores
X_train_tensor = tf.convert_to_tensor(X_train_input, dtype=tf.float32)
y_train_tensor = tf.convert_to_tensor(y_train, dtype=tf.float32)

X_val_tensor = tf.convert_to_tensor(X_val_input, dtype=tf.float32)
y_val_tensor = tf.convert_to_tensor(y_val, dtype=tf.float32)

X_test_tensor = tf.convert_to_tensor(X_test_input, dtype=tf.float32)
y_test_tensor = tf.convert_to_tensor(y_test, dtype=tf.float32)

# Crear el modelo
model = Sequential()
neurons = 128

# Agregar una capa LSTM
model.add(LSTM(neurons, input_shape=(sequence_length, len(input_columns))))

# Agregar dos capas densas intermedias
model.add(Dense(neurons, activation='relu'))

# Agregar una capa densa para la salida
model.add(Dense(len(output_columns)))

# Definir función de pérdida RMSE
def rmse_loss(y_true, y_pred):
    return tf.sqrt(tf.reduce_mean(tf.square(y_pred - y_true)))


# Compilar el modelo
model.compile(loss=rmse_loss, optimizer='adam', metrics=['accuracy'])

# Imprimir un resumen del modelo
model.summary()

# Entrenar el modelo
history = model.fit(X_train_tensor, y_train_tensor, batch_size=128,
                    epochs=10, validation_data=(X_val_tensor, y_val_tensor))

# Evaluar el rendimiento del modelo en el conjunto de prueba
```

```
loss = model.evaluate(X_test_tensor, y_test_tensor)
print("Pérdida en el conjunto de prueba:", loss)

# Guardar el modelo entrenado
model.save('lstm_model.h5')

# Graficar los errores en función de las épocas
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.savefig('grafica_errores.png')
plt.xlabel('N° de Épocas')
plt.ylabel('RMSE')
plt.legend()
plt.show()

# Obtener las predicciones del modelo en el conjunto de prueba
y_pred = model.predict(X_test_tensor)

# Calcular el RMSE para cada salida
rmse_1 = np.sqrt(mean_squared_error(y_test_tensor[:, 0], y_pred[:, 0]))
rmse_2 = np.sqrt(mean_squared_error(y_test_tensor[:, 1], y_pred[:, 1]))

# Imprimir los resultados
print("RMSE para Apertura VC de Presión:", rmse_1*100, "%.")
print("RMSE para Apertura VC de Nivel:", rmse_2*100, "%.")

# Desnormalizar las predicciones y los datos de prueba. Limitar a valores
entre 0 y 100
y_pred = y_pred * 100
y_test = y_test_tensor * 100

# Crear un DataFrame con los datos predichos y los datos de prueba
df_pred = pd.DataFrame({'Esperado_1': y_test[:, 0], 'Predicho_1': y_pred[:,
0],
                        'Esperado_2': y_test[:, 1], 'Predicho_2': y_pred[:,
1]})

# Guardar los datos en un archivo Excel
df_pred.to_excel('datos_predichos.xlsx', index=False)

# Graficar los resultados de las salidas
# Obtener las salidas predichas
pred_1 = y_pred[:, 0]
pred_2 = y_pred[:, 1]

# Obtener las salidas reales
real_1 = y_test[:, 0]
real_2 = y_test[:, 1]

# Crear una figura con dos subtramas
```

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 8))

# Graficar la salida 1
ax1.plot(pred_1, label='Predicho', linewidth=2)
ax1.plot(real_1, label='Real', linewidth=2)
ax1.set_xlabel('Tiempo [s]')
ax1.set_ylabel('Apertura VC de Presión [%]')
ax1.legend()

# Graficar la salida 2
ax2.plot(pred_2, label='Predicho', linewidth=2)
ax2.plot(real_2, label='Real', linewidth=2)
ax2.set_xlabel('Tiempo [s]')
ax2.set_ylabel('Apertura VC de Nivel [%]')
ax2.legend()

# Ajustar los márgenes y espaciado
fig.tight_layout()

plt.savefig('grafica_predicciones.png')
# Mostrar la figura
plt.show()
```

Anexo 2. Aplicativo en Python para optimización del modelo basado en redes neuronales recurrentes con memoria a corto plazo (LSTM)

La optimización se realizó mediante un algoritmo en Python usando librerías como Tensorflow, Pandas, Numpy, Keras, Scikeras y Sklearn; además, se usaron las técnicas de Times Series Split con validación cruzada y una búsqueda de hiperparámetros por cuadrícula (GridSearchCV), estableciendo diferentes niveles para cada hiperparámetro escogido.

```
# Importar librerías
import tensorflow as tf
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.model_selection import TimeSeriesSplit
from scikeras.wrappers import KerasRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer, mean_squared_error

print("Estoy corriendo")

# Ruta del archivo
rutaArchivo = './Datos.xlsx'

# Carga los datos desde el archivo Excel en un DataFrame de pandas
datos = pd.read_excel(rutaArchivo, decimal=',')

# Tratar valores faltantes
datos = datos.dropna()
print("Ya leí datos")

# Definir las columnas de entrada y salida:
input_columns = ['hPV [cm]', 'hSP [cm]', 'PPV [mbar]',
                 'PSP [mbar]', 'Frecuencia de la Bomba [Hz]']
output_columns = ['% Apertura Presion', '% Apertura Nivel']


def Escalizado(datos, columna, rango_original, rango_deseado):
    datos[columna] = (datos[columna] - rango_original[0]) / (rango_original[1]
-

rango_original[0]) * (rango_deseado[1] - rango_deseado[0]) + rango_deseado[0]

    return datos[columna]
```

```
# Escalizar los datos de las variables de entrada
datos['hPV [cm]'] = Escalizado(datos, 'hPV [cm]', [0, 60], [0, 1])
datos['hSP [cm]'] = Escalizado(datos, 'hSP [cm]', [0, 60], [0, 1])
datos['PPV [mbar]'] = Escalizado(datos, 'PPV [mbar]', [0, 600], [0, 1])
datos['PSP [mbar]'] = Escalizado(datos, 'PSP [mbar]', [0, 600], [0, 1])
datos['Frecuencia de la Bomba [Hz]'] = Escalizado(
    datos, 'Frecuencia de la Bomba [Hz]', [0, 60], [0, 1])

# Escalizar los datos de las variables de salida
datos['% Apertura Presion'] = Escalizado(
    datos, '% Apertura Presion', [0, 100], [0, 1])
datos['% Apertura Nivel'] = Escalizado(
    datos, '% Apertura Nivel', [0, 100], [0, 1])

# Definir una función para crear secuencias de longitud sequence_length


def create_sequences(data, sequence_length):
    X = []
    y = []
    for i in range(len(data) - sequence_length):
        X.append(data.iloc[i:i+sequence_length][input_columns].values)
        y.append(data.iloc[i+sequence_length][output_columns].values)
    return np.array(X), np.array(y)

# Definir función de pérdida RMSE para modelo LSTM


def rmse(y_true, y_pred):
    return tf.sqrt(tf.reduce_mean(tf.square(y_pred - y_true)))

# Definir función de pérdida RMSE para GridSearchCV
def rmse2(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))


rmse_scorer = make_scorer(rmse2, greater_is_better=False)

# Crear el modelo
def create_model(neurons, epochs, batch_size):
    model = Sequential()
    model.add(LSTM(neurons, input_shape=(sequence_length,
len(input_columns))))
    model.add(Dense(neurons, activation='relu'))
    model.add(Dense(len(output_columns)))
    model.compile(loss=rmse, optimizer='adam', metrics=['accuracy'])

    return model
```

```
# Realizar la búsqueda en la cuadrícula para n_splits y sequence_length
best_score = float('inf')
best_params = {}

# Definir la lista de valores para n_splits y sequence_length
n_splits = 5
sequence_length_values = [10, 30, 90, 270]

for sequence_length in sequence_length_values:

    # Dividir el conjunto de datos en n partes
    tscv = TimeSeriesSplit(n_splits=n_splits)

    # Crear secuencias de entrenamiento
    X_train_input, y_train = create_sequences(datos, sequence_length)

    # Crear modelo con KerasRegressor
    model = KerasRegressor(model=create_model, verbose=0)

    # Definir los parámetros a probar (GRID)
    neurons = [16, 32, 64]
    epochs = [10, 20, 40]
    batch_size = [32, 64, 128]
    param_grid = dict(model__neurons=neurons,
                      model__epochs=epochs, model__batch_size=batch_size)

    grid = GridSearchCV(estimator=model, param_grid=param_grid,
                        scoring=rmse_scorer, n_jobs=-1, cv=tscv)
    grid_result = grid.fit(X_train_input, y_train)

    if ((-1) * grid_result.best_score_) < best_score:
        best_score = (-1) * grid_result.best_score_
        best_params = {
            'n_splits': n_splits,
            'sequence_length': sequence_length,
            'neurons': grid_result.best_params_['model__neurons'],
            'epochs': grid_result.best_params_['model__epochs'],
            'batch_size': grid_result.best_params_['model__batch_size']
        }

        print("El mejor resultado para el RSME hasta el momento es: %f" %
              (best_score*100))
        print("Los mejores hiperparámetros hasta el momento son: ",
best_params)

# Resumen de resultados
print("Best Score: %f" % (best_score*100))
print("Best Parameters: ", best_params)
```

Anexo 3. Aplicativo en Python para la comunicación del modelo entrenado y el servidor OPC

```
# -*- coding: utf-8 -*-

# Importación de librerías
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from asyncua import Client, ua
import h5py
import logging
import asyncio
import tensorflow as tf


# Definir la función de pérdida personalizada
def rmse_loss(y_true, y_pred):
    return tf.sqrt(tf.reduce_mean(tf.square(y_pred - y_true)))

# Registrar la función de pérdida personalizada
tf.keras.utils.get_custom_objects().update({'rmse_loss': rmse_loss})

# Configuración básica del registro de eventos
logging.basicConfig(level=logging.WARNING)

# Obtener el objeto logger para el módulo 'asyncua'
_logger = logging.getLogger('asyncua')

# URL para la conexión OPC UA
url = "opc.tcp://127.0.0.1:49320"

# Definir la ruta del modelo LSTM entrenado
ruta_modelo = lstm_model.h5'

# Cargar el modelo entrenado
with h5py.File(ruta_modelo, 'r') as file:
    model_cargado = tf.keras.models.load_model(file)

# Crea una función para procesar las variables de entrada y obtener las
variables de salida predichas
def process_variables(variables_input):

    # Transformar los datos en tensores
    variables_input_tensor = tf.convert_to_tensor(
        variables_input, dtype=tf.float32)

    # Obtener las variables de salida predichas
    variables_output = model_cargado.predict(variables_input_tensor)

    return variables_output
```

```
def Escalizado (datos,columna, rango_original, rango_deseado):
  datos[columna] = (datos[columna] - rango_original[0]) / (rango_original[1] -
rango_original[0]) * (rango_deseado[1] - rango_deseado[0]) + rango_deseado[0]

  """ datos[columna] = (datos[columna] - valor_minimo) / (valor_maximo -
valor_minimo) """
  return datos[columna]


# Definición de la función principal
async def main():
    # Crear una instancia del cliente OPC UA
    client = Client(url=url)

    try:
        # Intentar establecer la conexión al servidor OPC UA
        await client.connect()
    except Exception as e:
        # En caso de error al conectar, imprimir el mensaje de excepción y
salir de la función
        print(e)
        return

# ------------------------------ Cliente -------------------------------------
#
    # Buffer FIFO Sliding Window para almacenar las variables de entrada
    sequence_length = 10
    buffer = []

    while True:
        # Variables de entrada entregadas por el PLC
        nivel_sp = await client.get_node("ns=2;s=Nivel.PLC Nivel.SP
Nivel").get_value()
        nivel_pv = await client.get_node("ns=2;s=Nivel.PLC
Nivel.Nivel").get_value()
        presion_sp = await client.get_node("ns=2;s=Nivel.PLC Nivel.SP
Presión").get_value()
        presion_pv = await client.get_node("ns=2;s=Nivel.PLC
Nivel.Presión").get_value()
        frec_bomba = await client.get_node("ns=2;s=Nivel.PLC Nivel.Frec
Bomba").get_value()

        # Variables de entrada
        variables_input = [nivel_pv, nivel_sp,
                           presion_pv, presion_sp, frec_bomba]

        # Agregar variables de entrada al buffer
        buffer.append(variables_input)
```

```
        if len(buffer) == sequence_length:
            # Definir las columnas de entrada
            input_columns = ['hPV [cm]', 'hSP [cm]', 'PPV [mbar]',
                             'PSP [mbar]', 'Frecuencia de la Bomba [Hz]']

            # Crear el DataFrame
            df = pd.DataFrame(buffer, columns=input_columns)

            # Escalar los datos de entrada
            df['hPV [cm]'] = Escalizado(df,'hPV [cm]',[0,60],[0,1])
            df['hSP [cm]'] = Escalizado(df,'hSP [cm]',[0,60],[0,1])
            df['PPV [mbar]'] = Escalizado(df,'PPV [mbar]',[0,600],[0,1])
            df['PSP [mbar]'] = Escalizado(df,'PSP [mbar]',[0,600],[0,1])
            df['Frecuencia de la Bomba [Hz]'] = Escalizado(df,'Frecuencia de
la Bomba [Hz]',[0,60],[0,1])

            X_input = df.to_numpy()
            X_input = X_input.reshape(1, -1, 5)

            # Obtener las variables de salida predichas y desnormalizarlas
            variables_output = np.clip(
                process_variables(X_input)[0]*100, 0, 100)

            print(variables_output)


            # Guardar las variables de salida para su posterior escritura en
el PLC
            aperture_valve_pressure = ua.DataValue(
                ua.Variant(variables_output[0], ua.VariantType.Float))
            aperture_valve_level = ua.DataValue(
                ua.Variant(variables_output[1], ua.VariantType.Float))

            # Obtener los nodos de las variables de salida
            valve_nivel_PLC = client.get_node(
                "ns=2;s=Nivel.PLC Nivel.Valvula Nivel")
            valve_pressure_PLC = client.get_node(
                "ns=2;s=Nivel.PLC Nivel.Valvula Presión")

            # Escribir las variables de salida en el PLC
            await valve_nivel_PLC.set_value(aperture_valve_level)
            await valve_pressure_PLC.set_value(aperture_valve_pressure)

            # Eliminar el elemento más antiguo del buffer (FIFO)
            buffer.pop(0)

    try:
        # Intentar desconectar el cliente OPC UA
        await client.disconnect()
    except Exception as e:
```

```
        # En caso de error al desconectar, imprimir el mensaje de excepción y
salir de la función
        print(e)
        return

# Ejecutar la función `main()` utilizando `asyncio.run()`
if __name__ == "__main__":
    asyncio.run(main())
```