



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Implementación de Control Predictivo en Hardware para aplicación en planta de biogás

Memoria para optar al título de
Ingeniero Civil en Electrónica

Alumno
Sebastián Rodrigo Rojas Castillo.

PROFESOR GUÍA
Dr. Juan Carlos Agüero.

CO-REFERENTE
Dr. Cesar Silva

Agosto 2026, Valparaíso, Chile



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

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

Título del trabajo: Implementación de control predictivo en Hardware para aplicación en planta de biogás

Nombre del candidato(a): Sebastián Rodrigo Rojas Castillo

Carrera / Grado: Ingeniería Civil electrónica

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

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Juan Carlos Agüero, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente

DEJO CONSTANCIA que:

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

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

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

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

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

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

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 13/10/2026 **Firma:** 

Estudiante o Candidato(a):

Fecha: 13/10/2026 **Firma:** 

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

Para mis seres queridos, pero por sobre todo a papá, mamá y Truenito.
-Sebastián Rojas C.

Dedicatoria

AGRADECIMIENTOS

ME gustaría expresar mis agradecimientos de todo corazón principalmente a mi papá y mamá: gracias por todas las oportunidades que me han brindado a lo largo de mi vida, por siempre darme lo que necesitara, por el apoyo incondicional en los momentos difíciles como el confinamiento o los últimos años de la carrera, gracias por confiar en que llegaría a esta última instancia y por sobre todas las cosas, gracias por estar presentes para mí. Gracias papá por darte el tiempo desde que era pequeño para enseñarme e inculcarme conocimientos, por proponerme desafíos y metas y por motivarme a ser mejor. Gracias mamá por ayudarme en las cosas básicas pero fundamentales de la vida, por sacrificar gran parte de tu vida para que yo estuviera bien y por todo lo que has hecho por mí. No saben lo agradecido que estoy con ustedes, los amo mucho.

Especial agradecimiento para Truenito, mi fiel compañero que me acompañó en los primeros años de la carrera y que estaba conmigo en las frías noches de Valparaíso mientras estudiaba para los certámenes. Sé que desde el cielo todavía me acompañas. Quiero decirte que te llevaste una parte de mi corazón que nadie va a poder llenar jamás. Todavía te extraño mucho, pero sé que ahora estás mejor.

Quiero dar un agradecimiento especial a Vanessa, por siempre estar a mi lado y otorgarme su ayuda cuando lo necesito. Te quiero mucho.

Agradecer a mi familia, a mis tíos, Marcerlo Rojas y Patricia Calderon y a mis primos por brindarme su apoyo. Además quisiera agradecer a mi grupo de amigos tanto del liceo como a los que conocí en la Universidad: Macc, Soba, Vicens, Fela, Deus, Papijou, Pandi, Guille, Jose, Seba Andrade, Gaspar y Edu. Muchas gracias por los buenos momentos y espero que hayan muchos más. Los quiero mucho a todos.

Por último agradecer al Centro Avanzado de Ingeniería Eléctrica y Electrónica ANID AC3E CIA250006, al proyecto Fondecyt ANID 1261290 del profesor Juan C. Agüero, y al Proyecto Interno USM 2026 PILIR.26.28

Sebastián Rodrigo Rojas Castillo

RESUMEN

Esta memoria se centra en la implementación en hardware de una arquitectura de Control Predictivo No Lineal (NMPC) con estimación de estados mediante un Filtro de Kalman Extendido (EKF), orientada a seguir una referencia de producción de metano en un proceso de digestión anaeróbica (DA). La planta de digestión anaeróbica se implementa usando el modelo ADM1 que se emplea como planta virtual junto al modelo reducido AM2 como predictor, ambos modelos ampliamente usados en la literatura. El EKF estima una perturbación que compensa el sesgo entre ambos modelos, otorgando acción integral libre de offset. La planta se simula en MATLAB/Simulink y el controlador se ejecuta en Python sobre una Raspberry Pi, comunicados mediante un servidor Modbus TCP.

Palabras Claves

NMPC, EKF, Digestión Anaeróbica, ADM1, Observadores, Hardware-in-the-Loop, Modbus TCP.

ABSTRACT

This Thesis focuses on the hardware implementation of a Nonlinear Model Predictive Control (NMPC) architecture with state estimation via an Extended Kalman Filter (EKF), aimed at tracking a methane production reference in an anaerobic digestion (AD) process. The anaerobic digestion plant is implemented using the ADM1 model, which is used as the virtual plant, together with the reduced AM2 model as the predictor, both models widely used in the literature. The EKF also estimates a disturbance that compensates for the bias between both models, giving the scheme an offset-free integral action. The plant is simulated in MATLAB/Simulink and the controller runs in Python on a Raspberry Pi, communicating through a Modbus TCP server.

Keywords

NMPC, EKF, Anaerobic Digestion, ADM1, Observers, Hardware-in-the-Loop, Modbus TCP.

ÍNDICE

AGRADECIMIENTOS	I
RESUMEN	II
ABSTRACT	III
ABREVIACIONES	VI
SIMBOLOGÍA	VIII
1. INTRODUCCIÓN	1
1.1. Estado del Arte	1
1.2. Hipótesis	4
1.3. Objetivo General, Específicos, Alcances y Limitaciones	4
1.4. Resumen del Capítulo	6
2. CONCEPTOS.	7
2.1. Algoritmo de Filtrado	7
2.1.1. Extended Kalman Filter.	8
2.2. Control Predictivo Basado en Modelos.	9
2.2.1. Control Predictivo Basado en Modelos No-Lineal.	10
2.2.2. Formulación Delta.	11
2.3. Modbus.	12
2.3.1. Esquema Cliente/Servidor.	14
2.4. Framework CasADi.	15
2.5. Modelos de digestión Anaeróbica.	15
2.5.1. Modelo ADM1.	16
2.5.2. Modelos reducidos y AM2.	16
2.5.3. Relación entre ambos enfoques.	17

3. PLANTEAMIENTO DEL PROBLEMA.	18
3.1. Problema simple.	19
3.2. Problema de Interés.	20
3.2.1. ADM1, modelo de la planta.	20
3.2.2. AM2, modelo del controlador.	21
3.3. Resultados Previos.	23
4. METODOLOGÍA.	28
4.1. Problema simple.	28
4.1.1. Planta.	30
4.1.2. Controlador en Python.	31
4.1.2.1. Sincronización mediante protocolo de handshake.	32
4.1.2.2. MPC.	32
4.1.2.3. Observador.	34
4.2. Problema de Interés.	36
4.2.1. Planta.	36
4.2.2. Controlador.	37
4.2.2.1. Python.	39
4.2.2.2. NMPC.	39
4.2.2.3. Extended Kalman Filter.	41
5. RESULTADOS.	43
5.1. Problema Simple.	43
5.2. Problema de interés.	45
6. CONCLUSIONES.	53
6.1. Conclusiones generales.	53
6.2. Trabajo Futuro	54
A. CÓDIGOS PROBLEMA DE INTERES	56
A.1. Codigo am2_model.py	56
A.2. Controlador NMPC.	60
A.3. Extended Kalman Filter.	66
B. CÓDIGOS PARA PRUEBA SIMPLE.	70
ÍNDICE DE FIGURAS	76
ÍNDICE DE TABLAS	77
BIBLIOGRAFÍA	78

ABREVIACIONES

Mayúsculas

DA	: Digestión Anaerobia
IWA	: International Water Association
ADM1	: Anaerobic Digestion Model No. 1
AM2	: Modelo reducido de digestión anaerobia
COD	: Demanda Química de Oxígeno (Chemical Oxygen Demand)
EKF	: Filtro de Kalman Extendido (Extended Kalman Filter)
HIL	: Hardware-in-the-Loop
IoT	: Internet of Things
IPOPT	: Interior Point OPTimizer
MBAP	: Modbus Application Protocol (encabezado de trama Modbus TCP)
MPC	: Control Predictivo Basado en Modelos (Model Predictive Control)
NLP	: Problema de Programación No Lineal (Nonlinear Programming)
NMPC	: Control Predictivo Basado en Modelos No-Lineal (Nonlinear MPC)
ODE	: Ecuación Diferencial Ordinaria (Ordinary Differential Equation)
PDU	: Protocol Data Unit
PLC	: Controlador Lógico Programable (Programmable Logic Controller)
RK4	: Método de Runge-Kutta de cuarto orden
RTU	: Remote Terminal Unit (modo de transmisión serial de Modbus)
SISO	: Sistema de una entrada y una salida (Single Input, Single Output)
SQP	: Programación Cuadrática Secuencial (Sequential Quadratic Programming)
SRT	: Tiempo de retención de sólidos (Solids Retention Time)
TCP/IP	: Transmission Control Protocol / Internet Protocol
VFA	: Ácidos Grasos Volátiles (Volatile Fatty Acids)

Códigos de función Modbus

FC3	: Read Holding Registers (lectura de registros de retención)
FC6	: Write Single Register (escritura de un único registro)
FC16	: Write Multiple Registers (escritura de múltiples registros)

Minúsculas

gCOD	: gramos de Demanda Química de Oxígeno
mmol	: milimol

SIMBOLOGÍA

Matrices.

$\mathbf{A}$	: matriz de estados continua del sistema, $\mathbf{A} \in \mathbb{R}^{n \times n}$
$\mathbf{B}$	: matriz de entrada continua del sistema, $\mathbf{B} \in \mathbb{R}^{n \times m}$
$\mathbf{C}$	: matriz de salida continua del sistema, $\mathbf{C} \in \mathbb{R}^{q \times n}$
$\mathbf{A}_d$	: matriz de estados discreta, $\mathbf{A}_d \in \mathbb{R}^{n \times n}$
$\mathbf{B}_d$	: matriz de entrada discreta, $\mathbf{B}_d \in \mathbb{R}^{n \times m}$
$\mathbf{C}_d$	: matriz de salida discreta, $\mathbf{C}_d \in \mathbb{R}^{q \times n}$
$\mathbf{A}_{aug}$	: matriz de estados del sistema aumentado
$\mathbf{B}_{aug}$	: matriz de entrada del sistema aumentado
$\mathbf{C}_{aug}$	: matriz de salida del sistema aumentado
$\mathbf{A}_t$	: Jacobiano de la función de transición f_Δ respecto al estado extendido
$\mathbf{C}_t$	: Jacobiano de la función de salida g_Δ respecto al estado extendido (EKF)
$\mathbf{Q}$	: matriz de covarianza del ruido de proceso (EKF) / matriz de ponderación de la salida (NMPC)
$\mathbf{R}$	: matriz de covarianza del ruido de medición (EKF) / matriz de ponderación de la entrada (MPC)
$\mathbf{P}$	: matriz de covarianza del error de estimación del EKF
$\mathbf{I}$	: matriz identidad, $\mathbf{I} \in \mathbb{R}^{n \times n}$
$\mathbf{K}$	: ganancia de Kalman
$\mathbf{L}_{aug}$	: ganancia del observador de Luenberger aumentado
$\mathbf{A}_{ineq}$	: matriz de restricciones lineales de desigualdad (tasa de cambio del control)

Vectores.

$\mathbf{x}$	: vector de estados de la planta o del modelo de predicción
$\mathbf{u}$	: vector/señal de entrada manipulada
$\mathbf{y}$	: salida medida o controlada del proceso
$\hat{\mathbf{y}}$	: salida estimada o predicha por el modelo

y_{sp}, y^{ref}	: perfil de referencia (<i>setpoint</i>) de la variable controlada
ΔU	: secuencia de incrementos de control sobre el horizonte de control
z	: vector de decisión del problema SQP (planta simple)
$\hat{x}$	: estimación del vector de estados

Parámetros del modelo.

X_1	: concentración de biomasa acidogénica [g/L]
X_2	: concentración de biomasa metanogénica [g/L]
S_1	: concentración de sustrato orgánico [gCOD/L]
S_2	: concentración de ácidos grasos volátiles, VFA [mmol/L]
Z	: alcalinidad total [mmol/L]
C	: concentración de carbono inorgánico total [mmol/L]
D	: tasa de dilución, variable manipulada [d^{-1}]
$S_{1,in}$	: Concentración sustrato orgánico [gCOD/L]
$S_{2,in}$	: Concentración ácidos grasos volátiles, VFA [mmol/L]
q_M	: flujo molar de metano
Y	: variable de salida controlada, $Y = q_M$ [gCOD/L d]
α	: fracción de bacterias en fase líquida
μ_1	: tasa de crecimiento de la biomasa acidogénica
μ_2	: tasa de crecimiento de la biomasa metanogénica
$\mu_{max,1}$	: tasa máxima de crecimiento acidogénico [d^{-1}]
μ_0	: tasa máxima de crecimiento metanogénico [d^{-1}]
K_{S1}	: constante de media saturación de S_1 [gCOD/L]
K_{S2}	: constante de media saturación de S_2 [mmol/L]
K_{i2}	: constante de inhibición por sustrato [mmol/L]
k_1	: Rendimiento de degradación del sustrato
k_2	: Rendimiento de Producción de VFA [mmol/g]
k_3	: Rendimiento de consumo de VFA [mmol/g]
k_4, k_5	: Rendimiento de Producción de CO2 [mmol/g]
k_6	: Rendimiento de producción de CH4 [mmol/g]
k_{la}	: constante de transferencia líquido-gas [d^{-1}]
K_h	: constante de Henry [mmol/L/atm]
P_t	: presión total [atm]
K_b	: constante de equilibrio [mol/L]
$d,$	: perturbación de entrada
$\hat{d}$	: estimación perturbación

Escalares del observador y el controlador

T_s, dt	: periodo de muestreo [d]
N	: horizonte de predicción
N_{ho}	: horizonte de predicción del NMPC
N_{hi}	: horizonte de control del NMPC
W_y	: peso del error de seguimiento en la función de costo del NMPC
$W_{\Delta D}$	: peso de la variación de control en la función de costo del NMPC
$J, cost$	: función de costo del problema de optimización
ν_k	: innovación del EKF
D_{min}	: límite mínimo admisibles de la tasa de dilución
D_{max}	: límite máximo admisibles de la tasa de dilución
ΔD_{min}	: límite mínimo admisibles del incremento de control
ΔD_{max}	: límite máximo admisibles del incremento de control

Subíndices

in	: valor de entrada
k	: instante de muestreo k
$j k$	: predicción en el paso j realizada en el instante k
sp, ref	: valor de referencia o <i>setpoint</i>

Superíndices

$*$	: valor óptimo
e	: estado extendido
$\hat{}$	: valor estimado, e.g., $\hat{x}, \hat{d}$
T	: transpuesta de un vector o matriz

INTRODUCCIÓN

1.1. Estado del Arte

LA DIGESTIÓN anaeróbica (DA) es una tecnología clave para el tratamiento de residuos orgánicos y la recuperación de energía renovable en forma de biogás. Su importancia ha crecido dentro de la economía energética emergente, no solo por su capacidad de transformar residuos en metano, sino también por su potencial en recuperación de nutrientes y valorización de corrientes orgánicas de bajo valor [1]. Sin embargo, la DA es un proceso biológico no lineal, multietapa y sensible a perturbaciones en composición del sustrato, carga orgánica, temperatura, pH, alcalinidad e inhibidores. Dichas características explican por qué la operación manual o basada en controladores convencionales puede conducir a oscilaciones en la producción de metano, baja eficiencia o incluso fallas del proceso [2, 3].

El modelado dinámico ha sido una herramienta central para comprender, simular y controlar digestores anaeróbicos. El modelo propuesto por la International Water Association (IWA), conocido como Anaerobic Digestion Model N°1 (ADM1), es una referencia ampliamente aceptada porque describe con detalle las etapas bioquímicas y fisicoquímicas del proceso [4]. No obstante, su alto número de estados y parámetros dificulta su uso directo en aplicaciones de control en tiempo real. Por esta razón se han desarrollado modelos reducidos, entre ellos el Anaerobic Model N°2 (AM2), que representa las etapas de acidogénesis y metanogénesis con una estructura más simple y adecuada para diseño de observadores y controladores [5]. También, se han propuesto procedimientos sistemáticos para derivar modelos simplificados desde ADM1, como AM2HN, manteniendo una relación entre el modelo detallado y el modelo usado para control [6]. En la misma

dirección, la linealización de ADM1 ha permitido reducir costos computacionales y facilitar su incorporación en esquemas MPC [7].

La estimación de estados es otro componente crítico, ya que muchas variables relevantes de DA no se miden en línea o requieren instrumentación costosa. Las revisiones sobre instrumentación y control destacan que variables como los ácidos grasos volátiles (VFA), alcalinidad, amonio, caudal y composición de biogás son fundamentales para mantener estabilidad; pero no siempre están disponibles con la frecuencia requerida para control avanzado [2]. Por ello se han utilizado sensores de software y observadores basados en modelos, como Dewasme *et al.* [8] estudian la estimación de estados y entradas desconocidas usando variantes del filtro de Kalman extendido, validando el enfoque con datos experimentales de un reactor anaeróbico. Por su parte Lara-Cisneros *et al.* [9] proponen estimar la concentración de VFA a partir del flujo de metano, lo que ilustra la posibilidad de inferir variables internas usando mediciones disponibles en planta. Estos antecedentes justifican el uso de observadores no lineales, como EKF o PF, para cerrar el lazo de control con información incompleta.

En cuanto al control, la literatura ha evolucionado desde estrategias de controladores proporcionales, integrativos y derivativos (PID) hacia controladores de lógica difusa, control adaptativo y control por modos deslizantes; hasta enfoques predictivos capaces de manejar restricciones y anticipar la dinámica del proceso. El control difuso ha sido aplicado a biorreactores anaerobicos para enfrentar incertidumbre y no-linealidad [10]; mientras que el control por modos deslizantes ha mostrado robustez y bajo costo computacional para regular producción de metano usando ADM1 como planta virtual y modelos reducidos para el controlador [11]. Aun así, el control predictivo basado en modelos (MPC) resulta especialmente atractivo porque permite optimizar acciones futuras sujetas a restricciones de entrada, estado y salida [12,13]. En plantas de biogás, el MPC se ha usado para producción guiada por la demanda energética [14], identificación de modelos en línea [15], control adaptativo de sistemas de potencia alimentados con biogás [16] y estrategias de arranque automático de digestores [17].

El control predictivo no-lineal basado en modelos (NMPC) amplía estas capacidades al considerar explícitamente modelos no-lineales. García-Gen *et al.* [18] aplican NMPC a la digestión de residuos fácilmente biodegradables usando el caudal de dilución como variable manipulada y la producción de metano como objetivo de control. Para procesos de dos etapas, Piceno-Díaz *et al.* [19] comparan NMPC estándar y robusto, mostrando que la formulación robusta mejora el desempeño frente a perturbaciones e incertidumbre. Kamesh y Rani [20] integran EKF iterativo en una formulación MPC basada en primeros principios para sistemas SISO, reforzando la idea de combinar estimación y control predictivo en

procesos no lineales. Estos trabajos muestran que NMPC puede mejorar la regulación de DA, pero también evidencian tres dificultades persistentes: dependencia de modelos simplificados, sensibilidad a sesgos de predicción y costo computacional del problema de optimización.

La tesis de Azúa-Poblete [21] aborda precisamente esta brecha al combinar un modelo AM2 discreto junto a un filtro de Kalman extendido y un controlador NMPC para optimizar la producción de biogás. Su propuesta usa ADM1 como emulador de planta y AM2 como modelo predictor, lo que permite estudiar el caso práctico en que el controlador opera con un modelo reducido. El trabajo reporta que el esquema NMPC-EKF mejora el seguimiento respecto a formulaciones más ideales y que la incorporación de acción integral, mediante la estimación de un estado de perturbación, reduce errores estacionarios desde valores cercanos al 12 % hasta valores menores al 1 %. En su versión de artículo, Azúa-Poblete et al. formalizan este enfoque como un NMPC libre de offset: el EKF estima los estados del sistema y un estado adicional de disturbio que representa el sesgo entre AM2 y ADM1; luego el NMPC usa dicha estimación para corregir la acción de control sobre la dilución y seguir una referencia de flujo de metano [22]. Los resultados indican que la acción integral y la estimación del disturbio son más determinantes que la calibración offline de parámetros, aunque esta última también mejora el desempeño.

Junto con los enfoques mecanísticos, también han aparecido métodos basados en datos. La predicción del desempeño de DA mediante aprendizaje automático permite identificar parámetros operacionales críticos y apoyar decisiones de operación [23]. En digestiones de alto contenido de sólidos, los modelos de machine learning han sido usados para predicción y optimización del proceso [24]. Estos enfoques no reemplazan necesariamente a los modelos de primeros principios, pero pueden complementar la identificación de parámetros, la detección de condiciones anómalas y la aceleración de controladores predictivos mediante aproximaciones aprendidas.

Finalmente, la implementación práctica de control avanzado exige considerar hardware, sensores y portabilidad del código. En el ámbito de instrumentación, se han propuesto esquemas usando *Internet of Things* (IoT) de bajo costo para monitoreo de gases de efecto invernadero, presión, temperatura y flujo de biogás en reactores anaeróbicos, facilitando adquisición remota de datos y reduciendo barreras de monitoreo en ambientes rurales o de laboratorio [25]. En el ámbito de cómputo embebido, distintas investigaciones han implementado NMPC en microcontroladores ARM, FPGA, Raspberry Pi y STM32, comparando solvers y restricciones de memoria/tiempo de muestreo [26–28]. Asimismo, la implementación de MPC en PLCs industriales muestra una ruta relevante para transferir algoritmos desde simulación hacia entornos de planta [29, 30].

En síntesis, el estado del arte converge hacia controladores predictivos que combinan modelos reducidos, estimación de estados y mecanismos de compensación de sesgo para operar procesos anaerobios no lineales bajo restricciones. El avance representado por el trabajo de Azúa-Poblete es especialmente pertinente para esta memoria porque propone una arquitectura EKF-NMPC con acción integral basada en disturbio, validada frente a un emulador ADM1 y orientada al seguimiento de producción de metano. La brecha que permanece abierta es la transición desde validaciones en simulación hacia implementaciones en hardware de control real, con mediciones disponibles, tiempos de cálculo acotados e integración con una planta de biogás experimental.

1.2. Hipótesis

En base al estado del arte, se desprende de varios trabajos que se tiene la planta y el controlador trabajando en la misma plataforma, lo cual es útil en el ámbito de la simulación y experimentación. Sin embargo, si se quiere escalar en el desarrollo de controladores es necesario separar ambos procesos, de manera tal que el algoritmo NMPC-EKF opere fuera del entorno de simulación de la planta y se comunique con ella a través de un enlace de datos real; en lugar de asumir (como ocurre en las validaciones puramente simuladas de la literatura) que controlador y planta comparten el mismo espacio de cómputo y de tiempo.

La hipótesis principal de esta memoria postula implementar una arquitectura de control Hardware-in-the-Loop en la que un controlador predictivo no-lineal y un observador de estados EKF, ambos ejecutados en una plataforma de cómputo embebido distinta a la planta, logren que la producción de metano de un digestor anaeróbico siga una referencia de interés de producción, cerrando de esta forma el lazo de control.

La señal de salida entregada por la planta simulada corresponde a la producción de metano, mientras que la señal de entrada manipulada por el controlador es la tasa de dilución D . Ambas señales se intercambian entre plataformas a través de una arquitectura de comunicación Modbus TCP, la cual debe ser capaz de sostener el lazo de control cerrado sin introducir desincronización ni pérdida de información entre la planta y el controlador.

Se espera que esta propuesta permita validar el desempeño del esquema HIL y lograr implementar de manera efectiva el controlador NMPC-EKF desarrollado por [21] en hardware embebido y que se logre un seguimiento a referencia correcto.

1.3. Objetivo General, Específicos, Alcances y Limitaciones

El objetivo general de esta memoria es implementar una arquitectura de control predictivo no-lineal para un proceso de digestión anaeróbica, orientada al seguimiento de una referencia de producción de metano implementando un controlador NMPC y un observador

de estados en una plataforma de cómputo embebido, siendo esta una Raspberry-Pi model 4b.

Para cumplir este objetivo se plantean los siguientes objetivos específicos:

- Estudiar los fundamentos de la digestión anaeróbica, los modelos dinámicos empleados para su representación y las principales estrategias de estimación y control aplicables al proceso.
- Revisar la formulación del controlador NMPC y del filtro de Kalman extendido desarrollado por [21] y utilizado como base para el seguimiento de la producción de metano en presencia de estados no medidos y diferencias entre el modelo predictor y la planta.
- Diseñar una arquitectura de comunicación cliente-servidor basada en Modbus TCP que permita intercambiar mediciones y acciones de control entre una planta simulada y un controlador ejecutado externamente.
- Validar la arquitectura propuesta mediante un problema de control simple; siendo este la implementación de una planta lineal en el Software MATLAB/Simulink y un controlador MPC junto a un observador ejecutados sobre Hardware embebido.
- Extender la implementación hacia el problema de interés, correspondiente al control de una planta de biogás simulada, considerando la ejecución del algoritmo de control en hardware embebido y la comunicación con el entorno de simulación.
- Analizar el desempeño de la implementación en términos de seguimiento de referencia, coherencia de las señales intercambiadas, restricciones operacionales y factibilidad computacional.

El alcance de esta memoria considera el estudio, implementación y validación experimental en simulación HIL de una arquitectura de control para digestión anaeróbica. La planta de proceso se representa en MATLAB/Simulink, mientras que el controlador y el observador se ejecutan en Python sobre una Raspberry Pi model 4b. La comunicación entre ambos componentes se realiza mediante registros Modbus TCP, lo que permite separar la simulación de la planta del cálculo de la acción de control. Además, se considera una etapa previa de validación con una planta lineal simple, con el fin de comprobar el intercambio de datos, la sincronización básica y la ejecución del ciclo de control antes de abordar el sistema de biogás.

Las limitaciones del trabajo se relacionan principalmente con el carácter simulado de la planta y con las restricciones propias del hardware utilizado. No se considera la implementación sobre un digestor anaerobio físico ni la validación con instrumentación

industrial real. Por lo tanto, los resultados obtenidos dependen de la fidelidad del modelo empleado como planta virtual y de las condiciones de simulación definidas. Asimismo, el desempeño del controlador puede verse afectado por el tiempo de cálculo disponible en la Raspberry Pi, por la configuración del solver de optimización, por la discretización del modelo y por los retardos asociados a la comunicación Modbus TCP. En consecuencia, esta memoria constituye una etapa de validación y acercamiento a una implementación real, pero no reemplaza las pruebas necesarias en una planta experimental o industrial.

1.4. Resumen del Capítulo

En este capítulo se presentó la motivación general de la memoria y se revisó el estado del arte asociado al modelado, estimación e implementación de control avanzado en procesos de digestión anaeróbica. La revisión mostró que la digestión anaeróbica es un proceso no lineal, sensible a perturbaciones y con variables internas difíciles de medir en línea, lo que justifica el uso de modelos reducidos, observadores de estado y controladores predictivos capaces de operar bajo restricciones.

También se identificó como antecedente principal el uso de arquitecturas EKF-NMPC con acción integral basada en la estimación de perturbaciones, donde un modelo reducido AM2 se utiliza para control y un modelo ADM1 puede actuar como planta virtual de mayor fidelidad. A partir de esta revisión se estableció la brecha que aborda esta memoria: avanzar desde validaciones puramente simuladas hacia una implementación distribuida en hardware de control, considerando comunicación, tiempos de cálculo y ejecución del algoritmo fuera del entorno de simulación principal.

Finalmente, se definieron el objetivo general, los objetivos específicos, los alcances y las limitaciones del trabajo. En los capítulos siguientes se introducen los conceptos necesarios de filtrado, control predictivo, Modbus TCP, CasADi y los modelos de digestión; luego se plantea el problema de implementación mediante una validación inicial con una planta simple y, posteriormente, mediante el problema de interés asociado a biogás. Después se presentan los resultados obtenidos y, por último, las conclusiones y posibles líneas de trabajo futuro.

CONCEPTOS.

2.1. Algoritmo de Filtrado

Un algoritmo de filtrado es una técnica utilizada en la teoría de control y en estadística para estimar el estado de un sistema dinámico en tiempo real. El propósito de esto es obtener una estimación precisa del estado, incluso con ruido en las mediciones. El filtrado es esencial para el procesamiento de señales y sistemas de control, ya que permite obtener mediciones precisas de variables que no pueden ser directamente observadas o que están contaminadas con ruido. Este ruido puede ser originado desde varias fuentes, tales como entradas desconocidas, ruido en los sensores, ruido en los canales de transmisión, entre otros [21].

La meta del filtrado es obtener/recuperar una estimación de interés, denotada como $\hat{x}(t)$, a partir de mediciones ruidosas ($y(t)$ y $u(t)$) cuando estas están disponibles. El filtrado es usado para distinguir un tipo de información específica relacionada con dos tipos específicos: el suavizado y la predicción. Con este contexto, filtrar es el proceso de recuperar información sobre $\hat{x}(t)$ en el instante t utilizando las mediciones hasta el instante t [21].

Uno de los algoritmos de filtrado más conocidos es el Filtro de Kalman, que es un algoritmo de filtrado óptimo para sistemas lineales que cuentan con ruido Gaussiano. Este filtro se basa en un modelo de estado del sistema y utiliza las mediciones para actualizar la estimación del estado. El Filtro de Kalman es ampliamente utilizado en aplicaciones como navegación, seguimiento de objetos, control de sistemas y procesamiento de señales [21].

En este contexto, como se cuenta con un sistema no lineal, es necesario un algoritmo de filtrado que sea capaz de manejar esta no-linealidad. Una de estas opciones es el Filtro de

Kalman Extendido (EKF), que es una extensión del filtro de Kalman pero para sistemas no-lineales [21].

Se vera el detalle de este algoritmo de filtrado a continuación.

2.1.1. Extended Kalman Filter.

El filtro de Kalman Extendido (EKF) es una extensión del filtro de Kalman diseñado para estimar sistemas no-lineales mediante la linealización local alrededor del estado estimado. Este filtro combina las predicciones del modelo no-lineal con las mediciones actuales utilizando las ecuaciones del filtro de Kalman para obtener una estimación optima del estado del sistema [21].

Esto se ve mediante el siguiente sistema no-lineal definido por las siguientes ecuaciones:

$$x_{t+1} = f(x_t, u_t) + w_t \quad (2.1)$$

$$y_t = g(x_t, u_t) + v_t \quad (2.2)$$

Donde $f(x_t, u_t)$ y $g(x_t, u_t)$ son funciones conocidas, w_t y v_t es ruido Gaussiano que no esta correlacionado, y por ultimo x_t y y_t son señales conocidas.

En este contexto, se tiene lo siguiente:

$$p(x_1) = \mathcal{N}(x_1; \mu_1, \Sigma_1) \quad (2.3)$$

$$p(x_t|y_{1:t}) \approx \mathcal{N}(x_t; \hat{x}_{t|t}, \Sigma_{t|t}) \quad (2.4)$$

$$p(x_{t+1}|y_{1:t+1}) \approx \mathcal{N}(x_{t+1}; \hat{x}_{t+1|t}, \Sigma_{t+1|t}) \quad (2.5)$$

Donde $p(x_1)$ representa la distribución inicial del estado x_1 , $p(x_t|y_{1:t})$ representa la distribución de probabilidad del estado x_t dado todas las mediciones hasta el tiempo t , y $p(x_{t+1}|y_{1:t+1})$ representa la distribución predictiva del estado x_{t+1} dado las mediciones hasta el tiempo t .

Por lo que las ecuaciones del filtro extendido de Kalman quedan:

$$K_t = \Sigma_{t|t-1} C_t^T (R + C_t \Sigma_{t|t-1} C_t^T)^{-1} \quad (2.6)$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t (y_t - g_\Delta(\hat{x}_{t|t-1}, u_t)) \quad (2.7)$$

$$\Sigma_{t|t} = (I - K_t C_t) \cdot \Sigma_{t|t-1} \quad (2.8)$$

$$\hat{x}_{t+1|t} = f_\Delta(\hat{x}_{t|t}, u_t) \quad (2.9)$$

$$\Sigma_{t+1|t} = Q + A_t \Sigma_{t|t} A_t^T \quad (2.10)$$

Con condiciones iniciales $\hat{x}_{1|0} = \mu_1$ y $\Sigma_{1|0} = \Sigma_1$.

Donde se tiene que K_t es la ganancia de Kalman que depende del calculo en cada tiempo

de C_t , $\hat{x}_{t|t}$ es la estimación actualizada del estado, $\sigma_{t|t}$ es la estimación actualizada de la covarianza, $\hat{x}_{t+1|t}$ es la estimación predicha del estado siguiente y por ultimo $\Sigma_{t+1|t}$ es la estimación de la covarianza predicha siguiente. Además en cada iteración del algoritmo se tienen que calcular las matrices de transición de estados y observacion que se obtienen calculando los siguientes Jacobianos:

$$A_t = \left. \frac{\partial f_{\Delta}(x_t, u_t)}{\partial x_t} \right|_{x_t = \hat{x}_{t|t}} \quad (2.11)$$

$$C_t = \left. \frac{\partial g_{\Delta}(x_t, u_t)}{\partial x_t} \right|_{x_t = \hat{x}_{t|t-1}} \quad (2.12)$$

Resultando en matrices que varían en el tiempo. Esta linealización local es importante para aproximar el comportamiento no-lineal del sistema a un modelo lineal alrededor del punto de operación del sistema.

2.2. Control Predictivo Basado en Modelos.

El control predictivo basado en modelos (MPC) es una técnica avanzada de control automático que usa un modelo matemático del sistema a controlar y los valores actuales de los estados del sistema en un momento dado para determinar de forma óptima las entradas del sistema a lo largo de una secuencia de muestras futuras. Esto se realiza respetando las restricciones que se pueden definir como valores máximos y mínimos permitidos para las entradas, salidas y entados [12, 13, 21].

El control predictivo basado en modelos tiene presente los siguientes componentes:

- Un modelo matemático del sistema a controlar: Se usa el espacio de estado del sistema representado en tiempo discreto para estimar el sistema dinámico.
- Horizonte de control de N muestras: Es el número de acciones de control futuras que se calculan en cada paso de tiempo para controlar el sistema.
- Conjunto de Restricciones: Limitaciones para el rango de las variables de entrada, salida y estados del sistema.
- Función de Costo: Es el problema de optimización que penaliza los errores entre las entradas y estados deseados y reales.

Para asegurar un buen rendimiento de esta técnica de control automático es necesario que el modelo matemático del sistema a controlar este bien implementado, ya que las señales de control se derivan de predicciones basadas en dichos modelos. Si se tiene un modelo impreciso las señales de control podrían no ser las óptimas e incluso podrían provocar inestabilidad en el sistema. Por el contrario, si el modelo representa con precisión el sistema real, las

predicciones serán más exactas y acordes a lo que se quiere controlar, lo que permitira que el control predictivo genere señales de control efectivas.

Otro factor importante para la efectividad/rendimiento del controlador es el metodo utilizado para optimiza la función de costos, ya que algunos métodos pueden tardar mucho tiempo en converger o pueden consumir muchos recursos computacionales para completar la optimización.

2.2.1. Control Predictivo Basado en Modelos No-Lineal.

El control predictivo basado en modelos no-lineal (NMPC) es una extensión del MPC orientada a sistemas cuya dinámica no puede ser representada adecuadamente mediante un modelo lineal. Mientras que el MPC lineal usa matrices constantes para predecir la evolución del sistema, el NMPC utiliza directamente funciones no-lineales para describir tanto la dinámica de los estados como la relación entre estados, entradas y salidas. Esto permite tratar procesos con interacciones complejas entre variables, restricciones físicas y cambios de operación donde una aproximación lineal puede resultar insuficiente [21, 31].

En tiempo discreto, el sistema no-lineal considerado se puede escribir como:

$$x_{t+1} = f_{\Delta}(x_t, u_t) \quad (2.13)$$

$$y_t = g_{\Delta}(x_t, u_t) \quad (2.14)$$

Donde x_t corresponde al vector de estados, u_t corresponde a la entrada de control, y_t corresponde a la salida medida o controlada, $f_{\Delta}(\cdot)$ representa la dinámica no-lineal discretizada del sistema y $g_{\Delta}(\cdot)$ representa la función no-lineal de salida. El subíndice Δ indica que el modelo se encuentra expresado en tiempo discreto, con un periodo de muestreo definido [21].

El principio de funcionamiento del NMPC consiste en resolver, en cada instante de muestreo, un problema de optimización sobre un horizonte de predicción finito N . A partir del estado actual, usualmente estimado mediante un observador como el EKF, el controlador predice la evolución futura del sistema y calcula una secuencia óptima de entradas. Sin embargo, siguiendo la estrategia de horizonte deslizante, solo se aplica la primera acción de control y el problema se vuelve a resolver cuando llega una nueva medición. De esta forma, el controlador incorpora retroalimentación y puede corregir el efecto de perturbaciones, ruido de medición y diferencias entre el modelo y la planta real [13, 21].

Una formulación general del problema NMPC puede expresarse como:

$$\min_{u_{t|t}, \dots, u_{t+N-1|t}} \sum_{k=0}^{N-1} \left(\|y_{t+k|t} - y_{t+k}^{ref}\|_Q^2 + \|u_{t+k|t}\|_R^2 \right) \quad (2.15)$$

$$\text{s.a. } x_{t+k+1|t} = f_{\Delta}(x_{t+k|t}, u_{t+k|t}) \quad (2.16)$$

$$y_{t+k|t} = g_{\Delta}(x_{t+k|t}, u_{t+k|t}) \quad (2.17)$$

$$x_{t+k|t} \in \mathcal{X}, \quad u_{t+k|t} \in \mathcal{U}, \quad k = 0, \dots, N-1 \quad (2.18)$$

$$x_{t+N|t} \in \mathcal{X}_f \quad (2.19)$$

$$x_{t|t} = \hat{x}_{t|t} \quad (2.20)$$

Donde y_{t+k}^{ref} es la referencia futura a seguir, Q y R son matrices de ponderación, $\mathcal{X}$ y $\mathcal{U}$ representan los conjuntos admisibles para estados y entradas, respectivamente, y $\mathcal{X}_f$ representa un conjunto terminal. Esta formulación permite penalizar el error de seguimiento y el esfuerzo de control, respetando al mismo tiempo las restricciones del sistema [21, 31].

En aplicaciones como la digestión anaeróbica, el NMPC resulta especialmente útil debido a la naturaleza no-lineal del proceso, la existencia de restricciones operacionales y la necesidad de optimizar una variable de interés, como la producción de biogás, sin llevar el sistema a zonas de operación no deseadas [18, 22].

2.2.2. Formulación Delta.

La formulación delta del NMPC modifica la variable de decisión del problema de optimización. En vez de optimizar directamente la entrada absoluta u_t , se optimiza el incremento de la entrada, denotado como δu_t . Por lo tanto, la entrada aplicada al sistema se actualiza como:

$$u_t = u_{t-1} + \delta u_t \quad (2.21)$$

Esta formulación es útil cuando se busca obtener acciones de control más suaves, limitar cambios bruscos en actuadores y mejorar el comportamiento del controlador frente a incertidumbres o diferencias entre el modelo usado por el controlador y la planta real [21, 31].

Usando esta idea, el modelo de predicción queda definido por:

$$x_{t+1} = f(x_t, u_t) \quad (2.22)$$

$$u_t = u_{t-1} + \delta u_t \quad (2.23)$$

$$y_t = g(x_t, u_t) \quad (2.24)$$

Donde δu_t pasa a ser la variable que el optimizador calcula en cada instante de muestreo. Luego, el controlador reconstruye la entrada real mediante la suma del incremento óptimo y la entrada aplicada en el instante anterior [21].

El problema de optimización para el NMPC con formulación delta se puede escribir como:

$$J = \min_{\delta u_0, \dots, \delta u_{N-1}} (\|y_t - y_{ref}\|_Q^2 + \|\delta u_t\|_R^2) \quad (2.25)$$

$$\text{s.t. } x_{t+1} = f(x_t, u_t), \quad t \geq 0 \quad (2.26)$$

$$y_t = g(x_t, u_t), \quad t \geq 0 \quad (2.27)$$

$$u_t = u_{t-1} + \delta u_t, \quad t \geq 0 \quad (2.28)$$

$$x_t \in \mathcal{X}, \quad u_t \in \mathcal{U}, \quad t = 0, \dots, N-1 \quad (2.29)$$

$$x_N \in \mathcal{X}_f \quad (2.30)$$

Donde y_t es la señal de medicion, y_{ref} es la señal de referencia, u_t es la señal de control, las variables Q y R son las matrices de ponderación asociadas con la señal de salida y la señal de entrada, respectivamente [21].

Finalmente, al resolver el problema en el instante t , solo se aplica el primer incremento óptimo:

$$u_t = u_{t-1} + \delta u_0^* \quad (2.31)$$

Luego se desplaza el horizonte de predicción, se incorpora la nueva medición disponible y se resuelve nuevamente el problema de optimización. Esta estrategia mantiene el carácter retroalimentado del NMPC y permite corregir desviaciones provocadas por perturbaciones, errores de estimación o desajustes del modelo [21, 22].

La formulación delta garantiza la convergencia de un sistema de control de lazo cerrado a un estado estacionario $(u_t, x_t, y_t) \rightarrow (u_\infty, x_\infty, y_\infty)$, caracterizado por $\delta u_\infty = 0$. Esta propiedad, inherente en la formulación delta, asegura que las variaciones de entrada se anulen en el estado estacionario. Si la referencia y_{ref} es alcanzable y el sistema es estable, un controlador NMPC con una funcion de coste adecuada llevará el sistema a la referencia deseada. En consecuencia, usando esta estrategia se logra un seguimiento a referencia preciso de forma natural, sin calculos adicionales.

Esta formulación es la que adopta el NMPC del problema de interés (Sección 4.2.2.2), donde la perturbación estimada por el EKF se incorpora como acción integral. El controlador MPC del problema simple (Sección 4.1.2.2), en cambio, opera directamente sobre la entrada absoluta, ya que la compensación de sesgo se logra mediante el observador de perturbación de salida en lugar de la formulación delta.

2.3. Modbus.

Modbus es un protocolo de comunicación serial desarrollado originalmente por Modicon en 1979 para la comunicación entre controladores lógicos programables (PLC). Con el

tiempo se convirtió en un estándar en la industria de automatización y control, gracias a su simplicidad, robustez y carácter abierto [32]. El protocolo define cómo los dispositivos intercambian datos organizados en registros de 16 bits, y opera bajo un modelo maestro-esclavo (RTU) o cliente-servidor (TCP/IP) en donde los clientes son los encargados de iniciar todas las transacciones, el servidor (esclavo) nunca inicia comunicación por cuenta propia, solo responde a las solicitudes que recibe.

Modbus admite varios modos de transmisión: Modbus RTU y Modbus ASCII para comunicaciones seriales (RS-232/RS-485), y Modbus TCP/IP para redes Ethernet. En este proyecto se emplea **Modbus TCP**, que encapsula las tramas Modbus en paquetes TCP/IP y permite la comunicación entre dispositivos distribuidos en una red local.

Cada paquete Modbus TCP se divide de la siguiente manera, en donde se divide en un encabezado MBAP (*Modbus Application Protocol Header*) de 6 bytes [32], seguido por la PDU (*Protocol Data Unit*), la trama Modbus se divide de la siguiente manera:

■ **Encabezado MBAP (6 Bytes):**

- Transaction ID (2 Bytes): Identificador para emparejar request/response.
- Protocol ID (2 Bytes): Siempre 0x0000 para Modbus.
- Length (2 Bytes): Indica la cantidad de Bytes que siguen después de Length (PDU) para saber de qué porte es el paquete.

■ **Protocol Data Unit (PDU):**

- Unit ID (1 Byte): Identificador del esclavo (heredado de Modbus serial), en el protocolo TCP no se utiliza.
- Function Code (1 Byte): Indica la operación que va a realizar el paquete, estas pueden ser FC3, FC6, etc.
- Data (Bytes Variables.): Son los datos a usar, la cantidad depende de la Function Code.

Las funciones Modbus más usadas por los clientes escritores y lectores para la implementación de este proyecto son las siguientes:

- **FC3 — Read Holding Registers:** permite al cliente leer uno o más registros de retención del servidor.
- **FC6 — Write Single Register:** permite al cliente escribir un único registro en el servidor.
- **FC16 — Write Multiple Registers:** permite al cliente escribir un bloque con varios registros.

Los datos intercambiados son enteros sin signos de 16 bits (0-65535). Para transmitir valores con una mejor resolución se aplica un factor de escalamiento, esto es que el valor real se multiplica por este factor antes de escribir y se divide por el mismo factor al momento de leer, por lo que hay que tener consistencia y conocer el valor de este factor.

2.3.1. Esquema Cliente/Servidor.

En el esquema implementado, el servidor Modbus TCP actúa como memoria compartida entre los dos agentes principales del sistema Hardware-in-the-Loop (HIL): la planta ADM1 ejecutada en MATLAB/Simulink y el controlador NMPC ejecutado en Python sobre una Raspberry Pi.

El servidor está implementado usando la herramienta de **Node.js** en donde se puede crear un servidor TCP nativo (módulo `net`) que escucha el puerto 502. Internamente este servidor mantiene un arreglo de 100 registros de retención (`Uint16Array`) que representan la memoria compartida, el valor de la cantidad de registros se puede variar dependiendo de lo que se necesita. El servidor implementa manualmente el *parsing* del protocolo Modbus TCP, respondiendo a las funciones FC3, FC6 y FC16 sin depender de bibliotecas externas [32]. Adicionalmente, integra un servidor HTTP Express para un panel de monitoreo en tiempo real accesible desde el navegador en caso de que se quiera hacer monitoreo remoto.

Los roles de cada cliente en este esquema son:

- **MATLAB (cliente escritor):** Al inicio de cada paso de simulación se escribe en el registro `REG[0, 1, 2...n]` las señales necesarias que necesite el otro cliente para su funcionamiento. La escritura de estas señales se puede hacer usando la función `modbus_write_reg`.
- **Python/Raspberry Pi (cliente lector-escritor):** Lee los registros que se mandaron desde el cliente-escritor y se escribieron en el servidor, ejecuta el ciclo del controlador y escribe la actuación de control a aplicar en algún registro `REG` dentro del servidor.
- **MATLAB (cliente lector):** Después de que el cliente-escritor escribe los datos en los registros, el cliente-lector lee la respuesta del controlador, esto es que se leen los registros escritos por Python para obtener la nueva acción de control y aplicarla a la planta simulada.

Esta arquitectura desacopla temporalmente ambos procesos y permite que operen en distintas plataformas sin requerir comunicación directa entre ellas.

2.4. Framework CasADi.

CasADi es un framework de código abierto orientado a la optimización numérica y el control óptimo, con especial énfasis en la diferenciación automática (*algorithmic differentiation*, AD) [33]. Fue desarrollado originalmente en el KU Leuven y actualmente mantiene una comunidad activa de desarrollo. Provee interfaces para Python, MATLAB y C++, y está diseñado para construir de manera eficiente problemas de optimización no-lineal a gran escala.

La capacidad central de CasADi es la representación simbólica de expresiones matemáticas mediante grafos computacionales, lo que permite calcular derivadas exactas (gradientes, Jacobianos, Hessianos) de forma automática, sin aproximaciones numéricas por diferencias finitas. Esto es fundamental para el NMPC, donde el solver de optimización requiere el gradiente y la Hessiana de la función de costo para converger de forma eficiente [33].

Para resolver el problema de optimización no-lineal resultante del NMPC, CasADi actúa como interfaz hacia solvers de terceros. En este proyecto se utiliza **IPOPT** (*Interior Point OPTimizer*), un solver de programación no-lineal de código abierto basado en métodos de punto interior que puede resolver eficientemente problemas con restricciones de igualdad y desigualdad [34]. La integración se realiza directamente desde Python mediante la API de CasADi:

1. Se definen las variables de decisión (la secuencia de controles $\{u_0, \dots, u_{N-1}\}$) como símbolos CasADi.
2. Se construye la función de costo y las restricciones como expresiones simbólicas.
3. CasADi compila el problema y lo pasa a IPOPT, que itera hasta converger a un óptimo local.
4. El resultado óptimo se extrae numéricamente y se aplica la primera acción de control.

En el contexto del NMPC implementado, CasADi permite formular el problema de optimización de horizonte deslizante de forma compacta y eficiente, aprovechando la estructura del modelo AM2 para calcular las sensibilidades necesarias sin costo computacional adicional [21, 33].

2.5. Modelos de digestión Anaeróbica.

Los procesos de digestión anaeróbica pueden representarse mediante modelos dinámicos que describen cómo la materia orgánica se transforma en biogás mediante reacciones bioquímicas y fenómenos fisicoquímicos. En la literatura, estos modelos se clasifican entre

enfoques de alto detalle, orientados a la simulación y al análisis del proceso, y modelos reducidos, orientados al diseño de estimadores y controladores en línea. En ambos casos, la dinámica del sistema suele escribirse de forma general como

$$\dot{x}(t) = f(x(t), u(t)), \quad y(t) = g(x(t), u(t)), \quad (2.32)$$

donde x representa los estados del digestor, u la entrada manipulada y y la variable medida, como el flujo de metano.

2.5.1. Modelo ADM1.

El modelo ADM1, propuesto por la IWA [4], constituye una referencia ampliamente aceptada para la modelación de digestores anaerobios. Este modelo describe con detalle las etapas de hidrólisis, acidogénesis, acetogénesis y metanogénesis, además de incorporar equilibrios fisicoquímicos de carbono, nitrógeno y alcalinidad. Debido a su nivel de detalle, ADM1 permite reproducir con mayor fidelidad el comportamiento de una planta real, pero también exige un número elevado de estados, parámetros y ecuaciones, lo que lo vuelve costoso desde el punto de vista computacional. Por esta razón, suele emplearse como planta virtual o emulador en estudios de control, mientras que el controlador opera sobre una representación más simple.

2.5.2. Modelos reducidos y AM2.

Para el diseño de controladores predictivos, es habitual utilizar modelos reducidos que preserven las dinámicas dominantes del proceso. En este trabajo se emplea el modelo matemático AM2 que es un modelo dinámico que describe la digestión anaeróbica de residuos fácilmente biodegradables [35]. Aunque inicialmente se calibró para simular la digestión anaeróbica (DA) de destilación de vino, el conjunto de parámetros cinéticos y estequiométricos del modelo se pueden ajustar para simular cualquier otro residuo orgánico, incluidos los sistemas de codigestión. El modelo AM2 ha sido propuesto en la literatura como una aproximación mínima de la digestión anaeróbica, capaz de representar las etapas principales de acidogénesis y metanogénesis con un número reducido de estados.

Basandonos en el modelo descrito en [18], el modelo AM2 asume que el proceso puede ser reducir a dos etapas principales: la acidogénesis, donde el sustrato S_1 es convertido mediante la acidogénesis (X_1) en ácidos grasos volátiles S_2 y CO_2 ; y la etapa de metanogénesis, donde los ácidos grasos volátiles S_2 son convertidos en CH_4 y CO_2 mediante la arquea metanogénica (X_2). En su formulación para control, el estado del sistema reducido se define como

$$x = \begin{bmatrix} X_1 & X_2 & S_1 & S_2 \end{bmatrix}^T \quad (2.33)$$

donde X_1 y X_2 representan biomasa acidogénica y metanogénica, y S_1 , S_2 denotan concentraciones de sustrato y ácidos grasos volátiles. La entrada manipulada corresponde a la tasa de dilución D , mientras que la salida de interés es el flujo de metano q_M . Esta estructura permite formular problemas de estimación y optimización con menor costo computacional que el modelo ADM1, lo que resulta adecuado para un controlador NMPC implementado en tiempo real.

2.5.3. Relación entre ambos enfoques.

La relación entre ADM1 y AM2 es central para este proyecto. ADM1 ofrece una descripción más completa de la planta, mientras que AM2 sirve como predictor y modelo de observación para el lazo de control. Sin embargo, esta simplificación introduce un desajuste entre modelo y planta, que debe ser compensado mediante estimación de estados y perturbaciones. En la estrategia propuesta, el EKF reconstruye los estados internos del modelo AM2 y además estima un término de perturbación que representa el sesgo entre la predicción del modelo reducido y la respuesta observada de la planta ADM1. De esta forma, el controlador puede seguir una referencia de metano sin depender de un modelo de alta complejidad en cada iteración de optimización.

PLANTEAMIENTO DEL PROBLEMA.

La implementación de un controlador predictivo no lineal para un proceso de digestión anaeróbica no corresponde solamente al diseño de una ley de control. El problema completo involucra una planta no lineal, estados internos que no se miden directamente, restricciones de operación, diferencias entre el modelo usado por el controlador y la planta que se desea controlar, además de una arquitectura de comunicación que permita ejecutar el algoritmo fuera del entorno de simulación principal. Por esta razón, antes de abordar directamente el proceso de biogás, se plantea una etapa de validación con una planta simple y posteriormente el problema de interés basado en la literatura de digestión anaeróbica y en el trabajo desarrollado por Azúa-Poblete [21, 22].

El antecedente principal de esta memoria es la estrategia EKF-NMPC libre de offset propuesta para controlar la producción de metano en digestión anaeróbica [22]. En dicha estrategia, ADM1 se utiliza como emulador de planta debido a su representación detallada del proceso, mientras que AM2 se utiliza como modelo reducido para estimación y predicción. Esta separación reproduce una situación práctica: el modelo más completo puede ser demasiado costoso para el cálculo en línea, pero el modelo reducido introduce sesgos que deben ser compensados mediante observación y acción integral.

En consecuencia, el problema general de esta memoria puede resumirse como la implementación y validación de un lazo de control distribuido capaz de calcular acciones de control en una plataforma externa, comunicarse con una planta simulada y mantener seguimiento de referencia bajo restricciones. La variable controlada final es la producción de metano, la variable manipulada es la tasa de dilución y la medición disponible para cerrar

el lazo es limitada. Sin embargo, para reducir el riesgo de implementación, el problema se divide en dos niveles: un problema simple para validar la arquitectura y un problema de interés asociado al proceso anaeróbico.

3.1. Problema simple.

El primer problema consiste en cerrar el lazo de control de una planta lineal de bajo orden ejecutada en MATLAB mediante un controlador MPC y un observador implementados en Python. Esta etapa no busca representar la digestión anaeróbica, sino validar los elementos mínimos que también serán necesarios en el problema de interés: intercambio de mediciones, escritura de acciones de control, escalamiento de variables, sincronización entre procesos y ejecución del optimizador fuera de MATLAB.

La planta elegida es un sistema lineal estable de tercer orden:

$$G(s) = \frac{6}{(s+1)(s+2)(s+3)}. \quad (3.1)$$

Su ganancia estática es unitaria y sus polos se encuentran en el semiplano izquierdo, por lo que permite estudiar seguimiento de referencias escalonadas sin introducir inestabilidades propias de la planta. Aun así, al ser de tercer orden, el controlador requiere una representación de estados y el observador debe reconstruir variables que no se entregan directamente como medición.

El problema de control se plantea en tiempo discreto como:

$$x_{k+1} = A_d x_k + B_d u_k, \quad (3.2)$$

$$y_k = C_d x_k, \quad (3.3)$$

donde u_k es la acción calculada por el controlador, y_k es la salida escrita por la planta simulada y x_k es el vector de estados. El objetivo es que y_k siga una referencia r_k mediante una arquitectura distribuida en la que MATLAB no calcula la acción de control, sino que la recibe desde Python a través de Modbus TCP.

Este caso simple permite aislar errores de implementación que serían más difíciles de diagnosticar directamente en el digestor anaeróbico. Si el seguimiento falla en esta etapa, la causa probablemente está en la comunicación, el escalamiento, la sincronización, el observador o la formulación MPC, y no en la complejidad bioquímica de ADM1. Por lo tanto, el problema simple actúa como una prueba de integración previa a la validación del controlador NMPC-EKF.

El desafío específico de esta sección es entonces diseñar una prueba donde:

- MATLAB ejecute la planta y entregue la salida.
- Python lea la medición y la referencia desde el servidor Modbus.
- El observador estime los estados requeridos por el controlador.
- El MPC calcule una acción de control.
- La acción calculada se escriba en un registro Modbus y sea aplicada por la planta en el paso siguiente.

Una vez validado este ciclo, la misma arquitectura puede trasladarse al problema no-lineal. La diferencia principal será que el controlador lineal será reemplazado por un NMPC, el observador por un EKF y la planta simple por un emulador ADM1 del proceso de digestión anaeróbica.

3.2. Problema de Interés.

El problema de interés corresponde al control de un proceso de digestión anaeróbica orientado al seguimiento de una referencia de producción de metano. La digestión anaeróbica transforma materia orgánica en biogás mediante una cadena de reacciones microbiológicas que incluye hidrólisis, acidogénesis, acetogénesis y metanogénesis [1, 4]. El biogás contiene principalmente metano y dióxido de carbono, por lo que su producción tiene valor energético y ambiental. No obstante, el proceso es lento, no-lineal y sensible a perturbaciones en composición del sustrato, carga orgánica, pH, temperatura e inhibidores [2, 3].

Para abordar este problema se distinguen dos modelos con roles distintos dentro del lazo de control: ADM1, utilizado como emulador de la planta real, y AM2, el modelo reducido sobre el cual opera el controlador. A continuación se describe cada uno.

3.2.1. ADM1, modelo de la planta.

Para estudiar este tipo de sistemas existe el modelo ADM1 [4], propuesto por la IWA, que entrega una representación detallada de las etapas bioquímicas y fisicoquímicas de la digestión anaeróbica [4]. La tesis de Azúa-Poblete utiliza ADM1 como sustituto de una planta real, debido a que permite simular el comportamiento de un digestor con mayor fidelidad que un modelo reducido y sin requerir de una instalación experimental [21]. Sin embargo, esa misma riqueza del modelo introduce una dificultad importante: ADM1 posee una cantidad elevada de estados, entradas, parámetros y ecuaciones, lo que vuelve poco conveniente su uso directo dentro de un controlador predictivo que debe resolver un problema de optimización en cada instante de muestreo.

En consecuencia, en esta memoria ADM1 se utiliza exclusivamente como planta virtual: se ejecuta en MATLAB/Simulink para generar las mediciones que el controlador debe seguir,

mientras que el cálculo de la acción de control y la estimación de estados se realizan sobre un modelo reducido, descrito a continuación.

3.2.2. AM2, modelo del controlador.

Por esta razón, el controlador se basa en AM2, un modelo reducido de digestión anaeróbica que conserva las etapas principales de acidogénesis y metanogénesis. Por lo que basados en el modelo descrito en [18], el modelo sistema puede ser representado por un sistema en espacio de estados continuo no-lineal, donde los estados son los degradadores microbiales X_1 y X_2 , el sustrato organico S_1 y S_2 , alcalinidad (Z) y carbono inorganico (C).

Este espacio de estados esta descritos por las siguientes ecuaciones:

$$\dot{X}_1 = \mu_1 X_1 - \alpha D X_1 \quad (3.4)$$

$$\dot{X}_2 = \mu_2 X_2 - \alpha D X_2 \quad (3.5)$$

$$\dot{S}_1 = D(S_{1in} - S_1) - k_1 \mu_1 X_1 \quad (3.6)$$

$$\dot{S}_2 = D(S_{2in} - S_2) + k_2 \mu_1 X_1 - k_3 \mu_2 X_2 \quad (3.7)$$

$$\dot{Z} = D(Z_{in} - Z) \quad (3.8)$$

$$\dot{C} = D(C_{in} - C) - q_C + k_4 \mu_1 X_1 + k_5 \mu_2 X_2 \quad (3.9)$$

Donde se tiene además:

$$\varphi = C + S_2 - Z + k_H P_T + \frac{k_6}{k_{La}} \mu_2 X_2 \quad (3.10)$$

$$q_M = k_6 \mu_2 X_2 \quad (3.11)$$

$$pH = -\log_{10} \left(k_b \frac{C - Z + S_2}{Z - S_2} \right) \quad (3.12)$$

$$\mu_1 = \frac{\mu_1^{max} S_1}{K_{S1} + S_1} \quad (3.13)$$

$$\mu_2 = \frac{\mu_2^{max} S_2}{k_{S2} + S_2 + (S_2^2/k_{I2})} \quad (3.14)$$

Los valores de estas constantes estan en la tabla 3.1

El modelo AM2 permite formular el problema de predicción con un costo computacional mucho menor que ADM1, pero introduce una diferencia estructural entre modelo y planta. Este punto es central en el planteamiento del problema: se desea controlar una planta representada por ADM1 usando un predictor AM2. En trabajos previos, cuando el NMPC utiliza AM2 para controlar datos generados por AM2, el seguimiento de referencia es adecuado; en cambio, cuando AM2 se usa para controlar una planta ADM1, aparece un

Parámetro	Valor	Condición Inicial	Valor
k_1	42.14	C_0	65
k_2	116.5	$C_{in,0}$	60
k_3	268	D_0	0.34
k_4	50.6	$S_{1,0}$	1.8
k_5	343.6	$S_{1in,0}$	9.5
k_6	457.73	$S_{2,0}$	3.0
k_b	$6,5 \cdot 10^{-7}$	$S_{2in,0}$	93.6
k_H	16	$X_{1,0}$	0.8
k_{La}	19.8	$X_{2,0}$	0.8
k_{I2}	256	Z_0	60
k_{S1}	7.1	$Z_{in,0}$	62.5
k_{S2}	9.28	α	0.5
μ_1^{max}	1.2	P_T	1
μ_2^{max}	0.2688		

Tabla 3.1: Parámetros del modelo AM2, adaptado de [21].

sesgo persistente en la salida debido al desajuste entre modelos [21, 22]. Esta diferencia puede producir error estacionario incluso si el controlador estabiliza el sistema.

Además, en una aplicación real no se dispone de todos los estados internos del digestor. Variables como biomasa, concentración de sustrato o VFA pueden requerir instrumentación costosa, análisis de laboratorio o mediciones indirectas [2, 9]. Por ello, el lazo de control se plantea usando como medición principal el flujo de metano. A partir de esta medición y del modelo AM2 se debe estimar el estado mediante un Filtro de Kalman Extendido. El observador también estima un estado adicional de perturbación, cuyo propósito es representar el sesgo de predicción entre AM2 y ADM1.

El problema de control puede describirse entonces como un problema SISO no lineal: calcular la tasa de dilución D para que el flujo de metano q_M siga una referencia $q_{M,ref}$, respetando restricciones de operación y usando estados estimados. El NMPC resuelve, en cada instante de muestreo, una optimización sobre un horizonte finito; luego aplica solo la primera acción de control y repite el procedimiento cuando llega una nueva medición [12, 13]. En la estrategia base, la formulación delta penaliza cambios de entrada y la perturbación estimada por el EKF introduce una acción integral que reduce el error estacionario [22].

La dificultad adicional abordada en esta memoria es que el controlador no se ejecuta dentro del mismo entorno que la planta. La planta ADM1 permanece en MATLAB/Simulink, mientras que el controlador NMPC y el EKF se ejecutaran en Python sobre una Raspberry Pi. La comunicación entre ambos se realiza mediante Modbus TCP, usando registros

para transmitir mediciones, referencias, acciones de control, banderas de sincronización y estimaciones internas. Por lo tanto, el problema no se limita a reproducir una simulación de escritorio, sino a verificar si la estrategia EKF-NMPC puede operar en una arquitectura Hardware-in-the-Loop.

En síntesis, el problema de interés queda definido por las siguientes condiciones:

- La planta a controlar se representa mediante ADM1, usado como emulador de un digestor anaeróbico.
- El controlador y el observador usan AM2, un modelo reducido adecuado para cálculo en línea.
- La variable manipulada es la tasa de dilución D .
- La variable controlada es la producción de metano q_M .
- Los estados internos no se miden directamente y deben ser estimados mediante EKF.
- Existe desajuste entre ADM1 y AM2, por lo que se requiere compensación de sesgo mediante una perturbación estimada.
- La implementación debe cerrar el lazo mediante comunicación Modbus TCP entre MATLAB y Python (ejecutado en Raspberry Pi).

Con este planteamiento, la contribución de la memoria se ubica en la transición desde una estrategia EKF-NMPC validada en simulación hacia una implementación distribuida que considera comunicación, sincronización, escalamiento de datos y cómputo embebido. El resultado esperado no es reemplazar la validación experimental en una planta real, sino demostrar que el lazo completo puede ejecutarse de forma funcional en una plataforma más cercana a una implementación industrial o experimental.

3.3. Resultados Previos.

Los resultados previos que sirven como base directa para esta memoria corresponden al trabajo desarrollado por Azúa-Poblete [21, 22]. En dicho trabajo se estudia el control de la producción de metano de un digestor anaeróbico usando ADM1 como emulador de planta y AM2 como modelo reducido para predicción, estimación y cálculo de la acción de control. Estos resultados son relevantes porque establecen el desempeño esperado del controlador EKF-NMPC antes de trasladarlo a una arquitectura distribuida con comunicación Modbus TCP y ejecución externa del algoritmo.

En una primera etapa, el controlador NMPC fue evaluado usando AM2 tanto como modelo de predicción como planta simulada. Bajo esta condición, el seguimiento de referencia

fue adecuado, lo que permitió verificar que la formulación del controlador era capaz de llevar la salida hacia los valores deseados cuando no existe desajuste estructural entre el modelo usado por el controlador y la planta. Posteriormente, se consideró un escenario más exigente, donde el controlador seguía usando AM2 para predecir, pero la planta correspondía a ADM1. En este caso el controlador logró estabilizar el proceso, aunque apareció un sesgo estacionario asociado a la diferencia entre ambos modelos. En particular, para referencias altas se reportaron errores de seguimiento que podían llegar a aproximadamente un 18 %, lo que evidenció la necesidad de incorporar estimación de estados y compensación de offset [21].

La estrategia final propuesta en [21, 22] combina un Filtro de Kalman Extendido con un controlador predictivo no-lineal. El EKF estima los estados internos del modelo AM2 y, además, un estado adicional de perturbación que representa principalmente el sesgo de predicción entre AM2 y ADM1. Esta perturbación estimada se incorpora al NMPC como una acción integral, permitiendo reducir el error estacionario producido por el desajuste entre modelos. La simulación final considera un horizonte de 20 días, con cuatro referencias de producción de metano que cambian cada cinco días, e incluye ruido de medición para evaluar la robustez de la estrategia, los resultados de esta estrategia se verán en los siguientes graficos: El resultado principal se observa en la Fig. 3.1. La salida presenta un transitorio

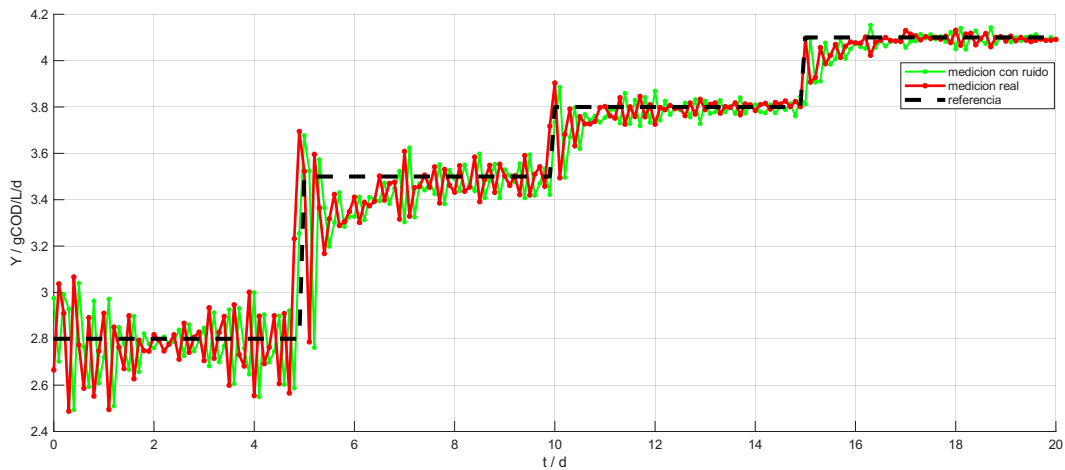


Figura 3.1: Seguimiento de referencia de la producción de metano obtenido por la estrategia EKF-NMPC en los resultados previos.

inicial debido a la diferencia entre las condiciones iniciales estimadas y las condiciones reales de la planta ADM1. Sin embargo, luego de los primeros ciclos de muestreo, el controlador logra llevar la producción de metano hacia la referencia. En los cambios bruscos de referencia aparecen aumentos temporales del error, comportamiento esperable en un sistema lento y no lineal como la digestión anaeróbica. No obstante, la acción integral generada a partir

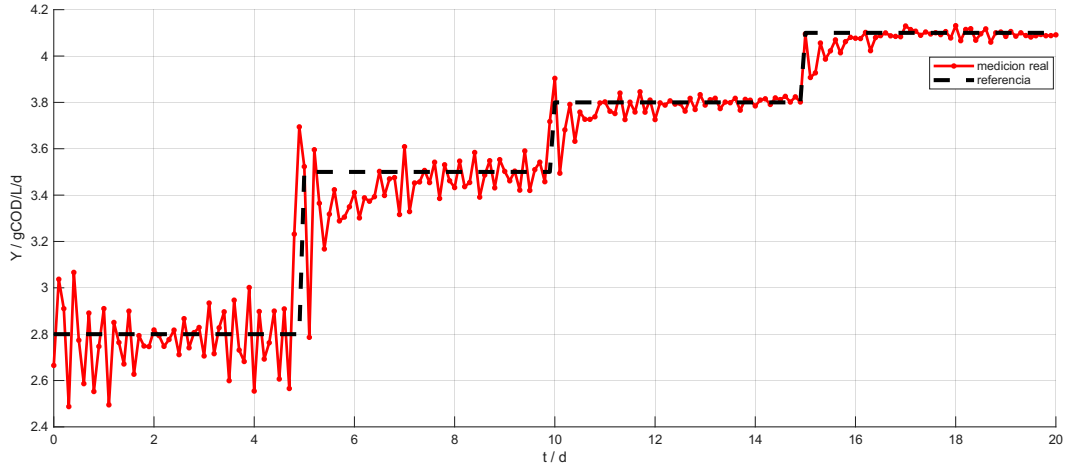


Figura 3.2: Seguimiento de referencia de la producción de metano obtenido por la estrategia EKF-NMPC, señal real en los resultados previos.

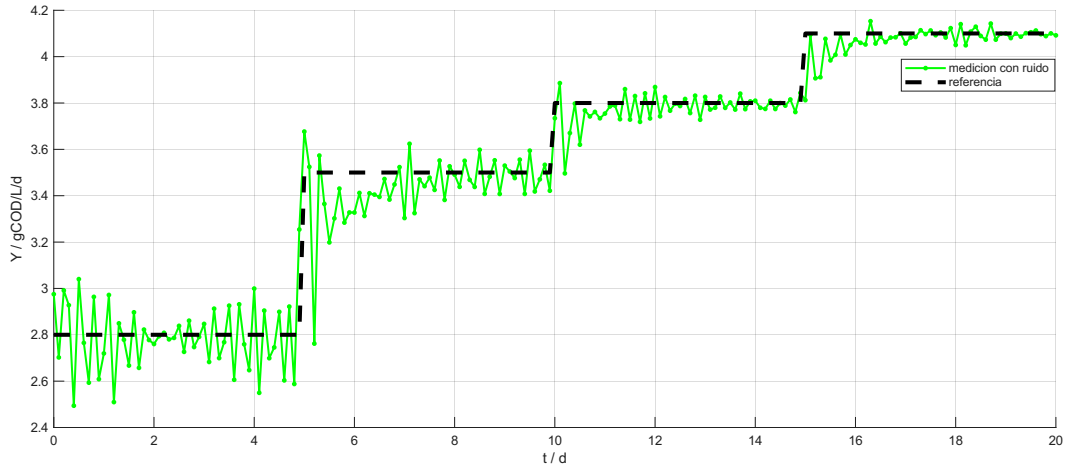


Figura 3.3: Seguimiento de referencia de la producción de metano obtenido por la estrategia EKF-NMPC, señal con ruido de medición en los resultados previos.

de la perturbación estimada permite que el error vuelva a valores bajos después de cada transitorio. Además en las figuras 3.2, 3.3 se muestran por separado la señal real y la señal con ruido de medición, para facilitar su visualización individual. La Fig. 3.4 muestra la actuación calculada por el controlador. La entrada manipulada corresponde a la tasa de dilución D , que se ajusta ante cada cambio de referencia para modificar la producción de metano. Además, la estimación de la perturbación aumenta especialmente en las referencias más exigentes, lo que coincide con el mayor desajuste entre AM2 y ADM1 en esas zonas de operación. Este comportamiento es importante, ya que muestra que el EKF no solo reconstruye estados,

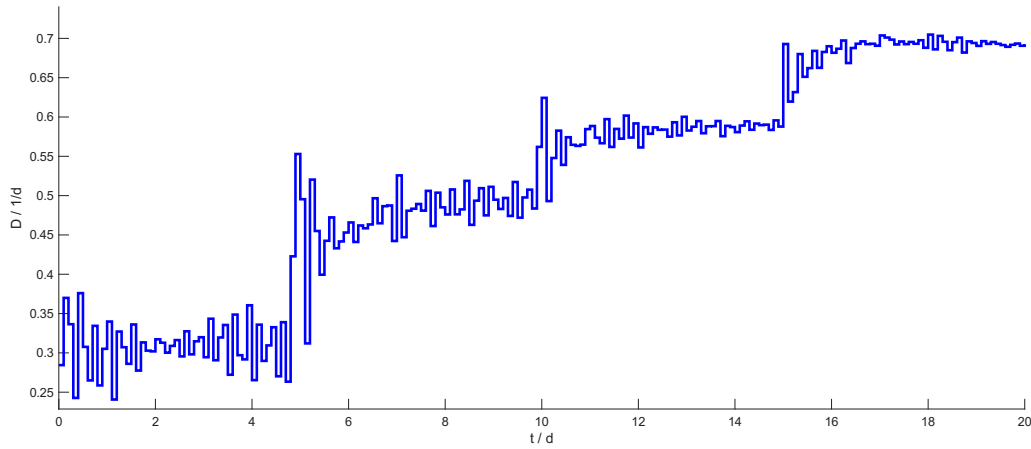


Figura 3.4: Tasa de dilución aplicada y perturbación estimada en los resultados previos.

sino que también entrega una variable que permite compensar el sesgo de modelo dentro del lazo de control. El error de seguimiento se presenta en la Fig. 3.5. En esta se aprecia que

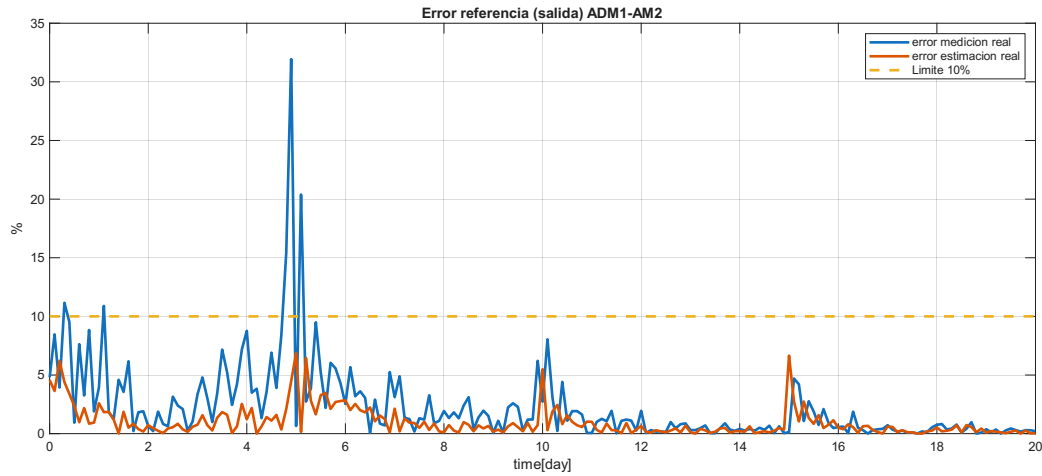


Figura 3.5: Error de seguimiento respecto de la referencia en los resultados previos.

los mayores errores ocurren durante los transitorios generados por los cambios de referencia. Una vez superados dichos transitorios, el error disminuye y se mantiene cercano a cero. Según el análisis reportado en [21], luego del primer día de simulación los errores se mantienen bajo el 2% en la simulación final, mientras que la incorporación de acción integral reduce errores estacionarios que antes podían estar alrededor del 12% hasta valores inferiores al 1%. La Fig. 3.6 complementa el análisis mostrando la evolución de los estados estimados. Dado que en una planta real no se dispone directamente de todas las concentraciones internas del digestor, esta estimación es una parte esencial de la estrategia. Los estados estimados

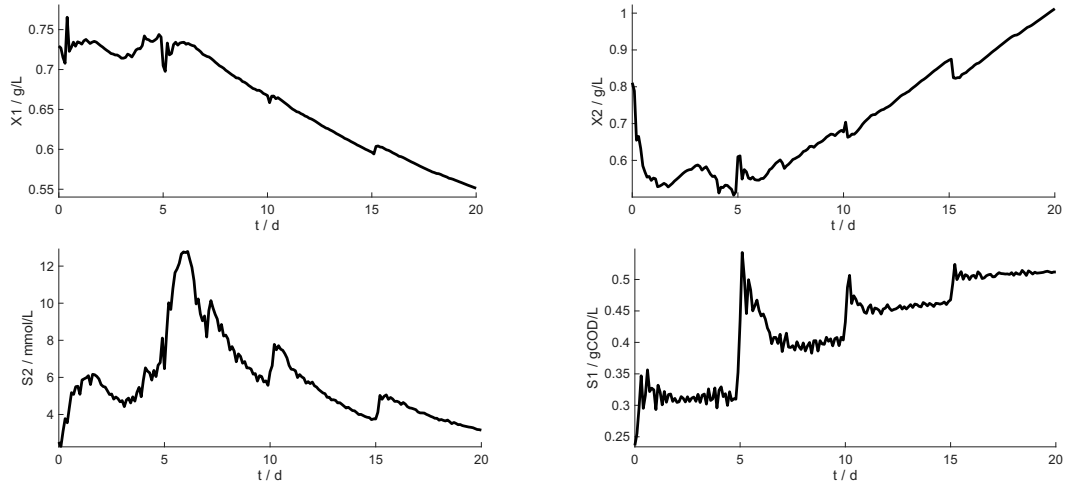


Figura 3.6: Estados en los resultados previos.

permiten inicializar el problema de optimización del NMPC en cada instante de muestreo y mantener el lazo cerrado usando una cantidad limitada de mediciones.

Para comparar distintas configuraciones, Azúa-Poblete evaluó seis casos: NMPC ideal con medición completa, NMPC con EKF, NMPC con EKF e identificación de parámetros, NMPC con EKF e integración, NMPC con EKF, identificación e integración, y finalmente esta última configuración con ruido de medición [21]. No obstante en esta memoria se trabajara con los ultimos dos, con un enfoque más profundo en el ultimo caso. La Tabla 3.2 resume las métricas más relevantes para las configuraciones finales. De estas comparaciones

Tabla 3.2: Métricas de desempeño reportadas para las estrategias finales en [21].

Caso	Estrategia	RMSE	MAPE [%]
5	EKF-NMPC + identificación + integración	0.1390	1.9106
6	Caso 5 + ruido de medición	0.1503	2.1122

se concluye que la mejor alternativa corresponde a la estrategia que integra estimación de estados y acción integral. Al agregar ruido de medición, las métricas se degradan levemente, pero se mantienen cercanas a las del caso sin ruido, lo que indica robustez frente a incertidumbres de medición. Por lo tanto, estos resultados previos constituyen el punto de referencia de esta memoria: el controlador EKF-NMPC ya fue validado en simulación, y el desafío posterior consiste en reproducir su funcionamiento en una implementación distribuida, donde MATLAB/Simulink ejecuta la planta ADM1 y Python calcula el EKF-NMPC en una Raspberry pi.

METODOLOGÍA.

Dentro de las buenas practicas de la ingenieria, se encuentra la de hacer pruebas con problemas más simples, para luego una vez validada esa solucion poder escalar al problema deseado, que comunmente es de una complejidad mayor. Por lo que antes de implementar el controlador NMPC deseado en la raspberry pi, lo que se hizo fue la implementacion de un controlador MPC para una planta de interes simple que reemplazaria la planta de biogás.

La arquitectura de comunicación propuesta inicialmente para el proyecto consideraba una conexión directa entre la Raspberry Pi y la planta (utilizando protocolos directos como UDP), tal como se muestra en la Figura 4.1.

Sin embargo, durante el desarrollo se determinó que era necesario incorporar un servidor Modbus TCP intermedio, encargado de centralizar la comunicación entre ambos clientes (planta y controlador) mediante registros. La arquitectura resultante, mostrada en la Figura 4.2, es la que finalmente se valida en esta memoria.

Una vez validada esta implementación y asegurandose que la arquitectura cliente-servidor 2.3.1 esta funcionando correctamente, se pasara al problema de Biogas deseado [36].

4.1. Problema simple.

Para comprobar la validación de la arquitectura propuesta entre el servidor Modbus y los clientes (controlador/planta), lo primero que se tiene que hacer es definir la planta o sistema simple con el que trabajar, luego hay que diseñar un controlador similar al NMPC pero más simple para lograr hacer control de esta planta, en este caso se hara un controlador MPC, que es la version lineal del controlador NMPC; además tambien hay que implementar un observador que sirva de reemplazo para el EKF. Por ultimo lo que se hara

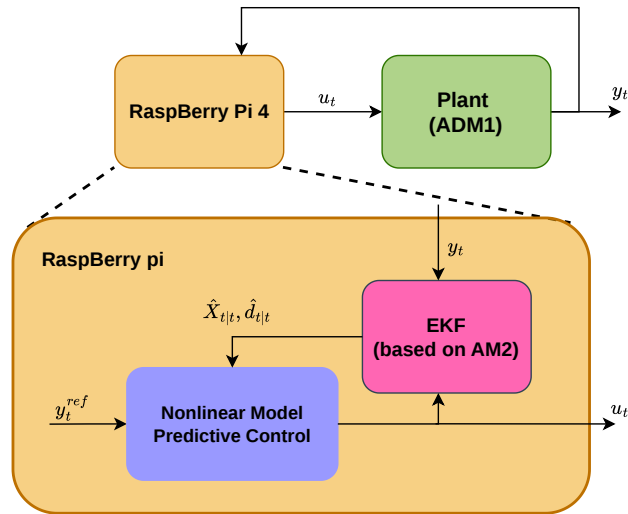


Figura 4.1: Esquema inicial de la arquitectura de comunicación propuesta.

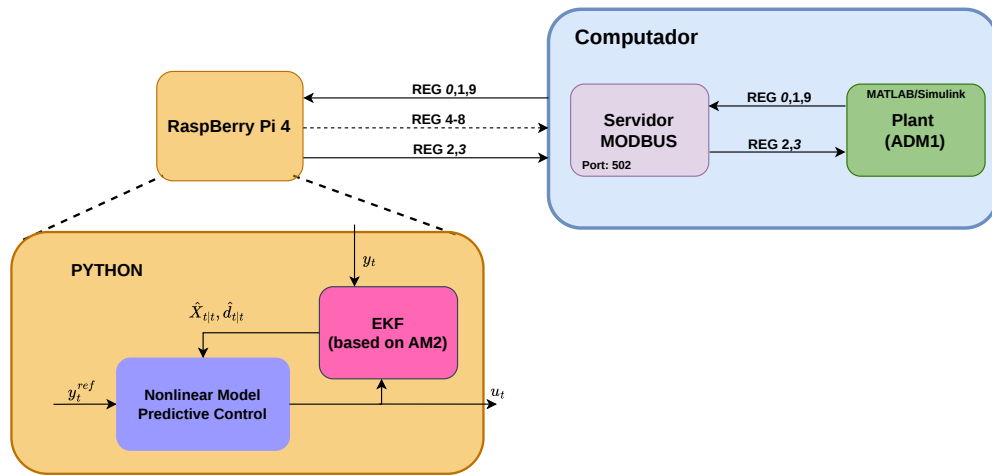


Figura 4.2: Arquitectura final de comunicación, incorporando el servidor Modbus TCP.

sera establecer la comunicación entre la planta implementada en MATLAB y el controlador junto al observador implementado en Python sobre la Raspberry Pi.

4.1.1. Planta.

La planta simple fue implementada directamente en MATLAB mediante el archivo `planta_matlab_handshake.m`. Esta implementación permite ejecutar el lazo de prueba de forma explícita: MATLAB escribe el setpoint vigente en el servidor Modbus, lee la acción de control calculada en Python, integra la planta un paso de muestreo y escribe nuevamente la salida para que el controlador pueda leerla.

La planta utilizada es la siguiente:

$$G(s) = \frac{6}{(s+1)(s+2)(s+3)} \quad (4.1)$$

Esta planta de tercer orden consta de 3 polos ubicados en -1 , -2 y -3 , por lo que corresponde a un sistema lineal estable. Además, su ganancia estática es unitaria, ya que:

$$G(0) = \frac{6}{(1)(2)(3)} = 1 \quad (4.2)$$

Esto permite que, en régimen permanente, la salida pueda seguir referencias constantes sin requerir un escalamiento adicional de la señal de entrada. Por esta razón, la planta resulta adecuada como primer caso de validación, ya que mantiene una dinámica suficientemente simple para verificar la comunicación y el algoritmo de control, pero conserva un orden mayor a uno que obliga al controlador y al observador a trabajar con una representación de estados.

Al expandir el denominador de la función de transferencia se obtiene:

$$G(s) = \frac{6}{s^3 + 6s^2 + 11s + 6} \quad (4.3)$$

A partir de esta expresión, una representación en espacio de estados en forma canónica controlable está dada por:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6 & -11 & -6 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(t) \quad (4.4)$$

$$y(t) = \begin{bmatrix} 6 & 0 & 0 \end{bmatrix} x(t) \quad (4.5)$$

Donde $x(t) \in \mathbb{R}^3$ corresponde al vector de estados de la planta, $u(t)$ es la entrada manipulada por el controlador MPC y $y(t)$ es la salida medida que se utiliza para cerrar el lazo de control.

Para la implementación en MATLAB se definen las matrices continuas A_c , B_c , C_c y D_c , se construye el sistema en espacio de estados y luego se discretiza mediante la función `c2d` disponible con `Control System Toolbox`. Recordar que para obtener el modelo discreto se

hace uso de un retentor de orden cero y un periodo de muestreo $T_s = 0,1s$, por lo que el modelo discreto resultante finalmente queda:

$$x_{k+1} = A_d x_k + B_d u_k \quad (4.6)$$

$$y_k = C_d x_k \quad (4.7)$$

con:

$$A_d = e^{A_c T_s} \quad (4.8)$$

$$B_d = \int_0^{T_s} (e^{A_c \tau} d\tau) B_c \quad (4.9)$$

$$C_d = C_c \quad (4.10)$$

La señal de entrada corresponde a la acción de control recibida desde el controlador implementado en Python, mientras que la salida generada por la planta se escribe en el servidor Modbus para ser leída por el observador y el controlador.

En la implementación se usa $T_s = 0,1$ s, valor que debe coincidir con el periodo usado en el controlador en Python. Además este modelo discreto es el que también se utiliza para el MPC y el observador; en Python se obtiene el modelo mediante la función `cont2discrete`.

La comunicación con el controlador se realiza mediante registros Modbus TCP. MatLab escribe el setpoint vigente en Reg_2, lee desde Reg_1 la señal de control u_k , hace el cálculo de la dinámica discreta de la planta y escribe la salida y_k en Reg_0. Los valores se transmiten como enteros de 16 bits usando un factor de escala igual a 100 para mejorar la resolución. Esto se puede ver en la implementación de código puesta en el Apéndice B.1.

4.1.2. Controlador en Python.

El controlador de la planta simple fue implementado en Python en el archivo `mpc.disturbance.handshake.py`. Este programa se comunica con el servidor Modbus TCP, de donde lee la salida de la planta y la referencia, estima los estados no medidos junto con una perturbación de salida y resuelve un problema MPC para calcular la acción de control que será aplicada por la planta programada en MATLAB. El código completo se encuentra en el Apéndice B.2

El modelo usado por el controlador corresponde al mismo sistema discreto descrito anteriormente. En Python se definen las matrices A_c , B_c y C_c , y luego se discretizan con `cont2discrete` usando $T_s = 0,1$ s y retención de orden cero. Así se obtienen las matrices A_d , B_d y C_d , utilizadas tanto por el observador aumentado como por el modelo de predicción del MPC.

4.1.2.1. Sincronización mediante protocolo de handshake.

En una implementación de tasa fija, donde tanto la planta (MATLAB) como el controlador (Python) avanzan según su propio reloj interno, la sincronización imperfecta entre ambos procesos introduce un desfase efectivo entre el instante en que se calcula una acción de control u_k y el instante en que dicha acción es efectivamente aplicada por la planta. Para eliminar esta fuente de error de modelado, se implementó un protocolo de sincronización explícita (*handshake*) entre ambos procesos, en lugar de compensar el desfase dentro del modelo del controlador.

El protocolo utiliza un registro Modbus adicional, Reg_3, denominado `pi_ready`, que actúa como bandera de sincronización:

- `pi_ready = 1`: el controlador Python escribió una acción de control nueva en Reg_1, por lo tanto la planta MATLAB puede avanzar un paso de simulación.
- `pi_ready = 0`: la planta MATLAB ya avanzó un paso y publicó una nueva medición en Reg_0 y un nuevo setpoint en Reg_2, por lo que el controlador puede leerlos y calcular la siguiente acción.

El ciclo de operación es el siguiente: el controlador espera a que `pi_ready` esté en 0, lee la medición y el setpoint vigentes, resuelve el problema de optimización, escribe la acción de control y finalmente levanta la bandera a 1. La planta, por su parte, espera a que la bandera esté en 1, lee la acción de control, avanza un paso su modelo discreto, publica la nueva medición y el nuevo setpoint, y finalmente baja la bandera a 0, señalizando que está lista para recibir la siguiente acción. Ambos procesos sondean la bandera mediante *polling* de baja latencia, con un tiempo máximo de espera (*timeout=5 s*) para detectar fallas de comunicación.

Bajo este esquema, cada transición de estado de la planta ocurre exactamente con la acción de control correspondiente, sin importar cuánto tiempo de reloj real haya tomado el cómputo o la transmisión Modbus.

4.1.2.2. MPC.

Gracias al protocolo de sincronización descrito en 4.1.2.1, la acción de control calculada en un ciclo es exactamente la que la planta aplica en la transición inmediatamente siguiente, por lo que el estado corresponde al estado de la planta, quedando de la siguiente forma:

$$z_k = x_k \quad (4.11)$$

y el modelo de predicción es el modelo nominal discreto, quedando:

$$z_{j+1|k} = A_d z_{j|k} + B_d u_{j|k} \quad (4.12)$$

$$\hat{y}_{j|k}^{(0)} = C_d z_{j|k} \quad (4.13)$$

Además, para eliminar el error estacionario frente a sesgos entre la planta y el modelo, el controlador incorpora una perturbación de salida estimada $\hat{d}_k$. Esta perturbación no forma parte del estado del MPC, sino que entra como parámetro externo actualizado en cada ciclo por el observador. Así, la salida predicha por el MPC se calcula como:

$$\hat{y}_{j|k} = C_d z_{j|k} + \hat{d}_k \quad (4.14)$$

De esta forma, el optimizador predice explícitamente el desplazamiento entre la salida del modelo y la medición real, y calcula la acción de control necesaria para compensarlo. Cabe destacar que $\hat{d}_k$ se mantiene *constante a lo largo de todo el horizonte de predicción*: no se le asigna una dinámica propia dentro del problema de optimización, sino que se asume estática en su valor estimado al inicio del ciclo.

El problema de optimización se formula en CasADi y se resuelve mediante IPOPT. La variable de decisión corresponde a la secuencia futura de acciones de control:

$$U = \begin{bmatrix} u_{0|k} & u_{1|k} & \cdots & u_{N-1|k} \end{bmatrix}^T \quad (4.15)$$

El horizonte de predicción utilizado es $N = 15$, lo que equivale a 1,5 segundos (virtuales) de predicción para $T_s = 0,1$ s. La función de costo implementada penaliza el error de seguimiento y el esfuerzo de control:

$$J = \sum_{j=0}^{N-1} \left(Q(r_k - \hat{y}_{j|k})^2 + R u_{j|k}^2 \right) + Q_T(r_k - \hat{y}_{N|k})^2 \quad (4.16)$$

En la implementación se usan los pesos $Q = 80$ y $R = 5$. Además, se agrega una penalización terminal equivalente a $Q_T = 15Q$ para favorecer que la predicción final se acerque a la referencia.

La señal de control se restringe al rango:

$$-6 \leq u_{j|k} \leq 6, \quad j = 0, \dots, N-1 \quad (4.17)$$

Luego de resolver el problema MPC, solamente se aplica la primera acción óptima $u_{0|k}^*$, siguiendo la estrategia de horizonte deslizante. Esta acción se escribe en Reg_1 y, mediante el protocolo de *handshake*, se garantiza que sea exactamente la acción que la planta aplicará en el siguiente paso. En el ciclo siguiente se lee una nueva medición, se actualiza el observador y se resuelve nuevamente el problema. Para mejorar el tiempo de convergencia del solver, el programa usa *warm start*, desplazando la solución óptima anterior como condición inicial de la siguiente iteración:

$$U_{\text{warm}} = \begin{bmatrix} u_{1|k}^* & u_{2|k}^* & \cdots & u_{N-1|k}^* & u_{N-1|k}^* \end{bmatrix}^T \quad (4.18)$$

4.1.2.3. Observador.

Como la planta solo entrega la salida y_k , pero el MPC requiere una estimación del vector de estados x_k , por lo que se implementa un observador de Luenberger discreto. Este observador no estima solamente los estados de la planta, sino que también estima una perturbación constante de salida. Para esto se define el siguiente estado extendido:

$$\xi_k = \begin{bmatrix} x_k \\ d_k \end{bmatrix} \quad (4.19)$$

donde d_k representa una perturbación o sesgo desconocido que se suma a la salida medida. Se asume que esta perturbación cambia lentamente, por lo que se modela como constante:

$$x_{k+1} = A_d x_k + B_d u_k \quad (4.20)$$

$$d_{k+1} = d_k \quad (4.21)$$

$$y_k = C_d x_k + d_k \quad (4.22)$$

Esta representación se escribe en forma matricial como:

$$\xi_{k+1} = A_{aug} \xi_k + B_{aug} u_k \quad (4.23)$$

$$y_k = C_{aug} \xi_k \quad (4.24)$$

con:

$$A_{aug} = \begin{bmatrix} A_d & 0 \\ 0 & 1 \end{bmatrix}, \quad B_{aug} = \begin{bmatrix} B_d \\ 0 \end{bmatrix}, \quad C_{aug} = \begin{bmatrix} C_d & 1 \end{bmatrix}, \quad \xi_k = \begin{bmatrix} x_k \\ d_k \end{bmatrix} \quad (4.25)$$

Además el vector de estados extendido es $\xi_k \in R^4$. Para que d_k sea efectivamente estimable, el par (A_{aug}, C_{aug}) debe ser observable. Esto se cumple en este caso porque (A_d, C_d) ya es observable, condición necesaria para lograr error nulo en estado estacionario con este tipo de esquema de estimación de perturbación constante.

El observador aumentado predice primero el estado extendido:

$$\hat{\xi}_{k|k-1} = A_{aug} \hat{\xi}_{k-1|k-1} + B_{aug} u_{k-1} \quad (4.26)$$

$$\hat{y}_{k|k-1} = C_{aug} \hat{\xi}_{k|k-1} \quad (4.27)$$

En donde u_{k-1} corresponde exactamente a la última acción de control enviada por el controlador, la cual se sabe con certeza que fue la aplicada por la planta en la transición correspondiente gracias al protocolo de *handshake* descrito en 4.1.2.1.

Luego se calcula la innovación entre la medición recibida desde MATLAB y la salida estimada:

$$\nu_k = y_k - \hat{y}_{k|k-1} \quad (4.28)$$

Finalmente, el estado extendido estimado se corrige mediante:

$$\hat{\xi}_{k|k} = \hat{\xi}_{k|k-1} + L_{aug} \nu_k \quad (4.29)$$

De esta estimación se extraen tanto los estados de la planta $\hat{x}_k$ como la perturbación estimada $\hat{d}_k$.

La ganancia L_{aug} se calcula por asignación de polos usando la funcion `place_poles` disponible en la libreria `scipy`, aplicando la propiedad de dualidad: se diseña una ganancia de realimentación de estado para el par (A_{aug}^T, C_{aug}^T) , lo cual es equivalente a diseñar la ganancia de inyección de salida L_{aug} del observador, ya que

$$\text{eig}(A_{aug} - L_{aug}C_{aug}) = \text{eig}(A_{aug}^T - C_{aug}^T L_{aug}^T). \quad (4.30)$$

En el código se seleccionan los polos continuos $-3 \pm j$, $-4,5$ y -2 . Los tres primeros se diseñaron para ser más rápidos que el polo más lento de la planta, mientras que el último se elige más lento y se asigna a la dinámica de la perturbación, evitando que su estimación amplifique ruido de cuantización o pequeñas variaciones de comunicación, evitando que la estimación de d_k amplifique ruido de cuantización o pequeñas variaciones de comunicación. Estos polos se llevan al dominio discreto mediante:

$$p_d = e^{p_c T_s} \quad (4.31)$$

En la práctica, la corrección se aplica en dos etapas (ecs. 4.26–4.29), por lo que la dinámica de error resultante corresponde a $(I - L_{aug}C_{aug})A_{aug}$ en lugar de $(A_{aug} - L_{aug}C_{aug})$ usado como objetivo de diseño en la ecuación (4.30). Para que el error de estimación tienda a cero, los autovalores de $(I - L_{aug}C_{aug})A_{aug}$ deben quedar dentro del círculo unitario, condición que se verifica numéricamente en la implementación.

Con la estimación $\hat{x}_k$ y la perturbación estimada $\hat{d}_k$, el controlador construye los parámetros que se entregan al optimizador:

$$p = \begin{bmatrix} x_k \\ r_k \\ \hat{d}_k \end{bmatrix} \quad (4.32)$$

La acción de control calculada por el MPC se escribe en `Reg_1`, y mediante la bandera `pi_ready` (`Reg_3`) se señala a la planta MATLAB que puede avanzar el siguiente paso de simulación con dicha acción.

Los resultados de la prueba simple se pueden ver en el capítulo de resultados 5, específicamente en la sección 5.1, además que el código del controlador realizado en python se puede ver completamente en el anexo B.2.

4.2. Problema de Interés.

Luego de validar la arquitectura con una planta lineal simple, el siguiente paso consiste en aplicar la misma idea al problema de interés de esta memoria, siendo este el control de una planta de digestión anaeróbica orientada al seguimiento de una referencia de producción de metano [36]. En este caso la dificultad aumenta porque el proceso es no lineal, los estados internos no están disponibles directamente y existe una diferencia estructural entre el modelo usado para controlar y la planta que se desea utilizar. La estrategia implementada

sigue el enfoque presentado en [36], esto es que la planta de mayor fidelidad se ejecuta en MATLAB usando el modelo ADM1 [4] como emulador del proceso, mientras que el controlador embebido en Python utiliza un modelo reducido AM2 [36] para estimación y predicción. De esta forma, se reproduce un escenario realista de control, donde el algoritmo no tiene acceso al modelo completo de la planta, sino a una representación simplificada que debe ser suficientemente informativa para calcular las acciones de control necesarias.

El objetivo del lazo de control es regular el flujo de metano q_M siguiendo una referencia definida por el operador. La variable manipulada corresponde a la tasa de dilución D . Como q_M es la única medición usada por el controlador, se incorpora un Filtro de Kalman Extendido (EKF) para estimar los estados internos del modelo AM2 y un estado adicional de perturbación. Luego, estas estimaciones son entregadas al controlador NMPC, el cual resuelve un problema de optimización no-lineal en cada instante de muestreo y escribe la nueva acción de control en el servidor Modbus, esta acción de control es la que luego lee la planta y de esta forma se termina el ciclo de control.

4.2.1. Planta.

La planta de interés corresponde al proceso de digestión anaeróbica basado en ADM1 [4], ejecutado en MATLAB. ADM1 es un modelo detallado del proceso anaeróbico y permite representar con mayor fidelidad la dinámica biológica y fisicoquímica del digestor. Sin embargo, por su complejidad, no se utiliza directamente dentro del controlador NMPC, sino que solo se utiliza como planta virtual para cerrar el lazo y evaluar si el controlador implementado en la Raspberry Pi puede operar correctamente frente a una dinámica más compleja que el modelo AM2 usado para predecir.

La planta MATLAB y el controlador Python se comunican a través del mismo servidor Modbus TCP validado en el problema simple. En este caso, MATLAB escribe el contador de paso, la medición de flujo de metano y la referencia deseada por el operador en los registros del servidor; mientras que Python escribe la tasa de dilución calculada, una señal de disponibilidad que actúa como flag para activar otro paso en la planta y las estimaciones del EKF.

El mapa de los registros usado es:

- Reg_0: contador de paso k escrito por MATLAB.
- Reg_1: medición q_M escrita por MATLAB.
- Reg_2: acción de control D escrita por Python.
- Reg_3: señal pi_ready escrita por Python.
- Reg_4 a Reg_7: estimaciones $\hat{X}_1$, $\hat{X}_2$, $\hat{S}_1$ y $\hat{S}_2$, escritas por Python.
- Reg_8: perturbación estimada $\hat{d}$.
- Reg_9: referencia de salida escrita por MATLAB.

Los valores transmitidos por Modbus se escalan antes de escribirse en los registros de 16 bits. Para la acción de control y la perturbación estimada se usa un factor de escala igual a 1000, mientras que para mediciones y estados estimados se usa un factor igual a 100. Esto permite representar magnitudes continuas con mayor resolución manteniendo una interfaz de comunicación simple. En el caso de los registros 4-8 estos se usan solo de monitoreo y llegan solo al servidor Modbus, en donde el operador los puede visualizar junto a las otras señales, pero el valor de estos registros no llega a la planta ADM1 ya que esta no los utiliza en ninguna parte del ciclo de operación. En la figura 4.2 se logra apreciar la dirección y flujo de cada uno de los registros usados en esta memoria.

4.2.2. Controlador.

Para el diseño del controlador y del observador se utiliza el modelo AM2 reducido [36]. Este modelo representa principalmente dos etapas del proceso: acidogénesis y metanogénesis. En la primera, el sustrato orgánico S_1 es consumido por la biomasa acidogénica X_1 y transformado en ácidos grasos volátiles (S_2) y CO_2 . En la segunda etapa, la biomasa metanogénica X_2 consume S_2 y produce CO_2 y metano (CH_4), que es nuestra variable de salida. La versión usada en este trabajo es el modelo AM2 reducido que se desarrolló en [36], en donde los estados de Alcalinidad (Z) y carbono inorgánico (C) son descartados ya que no afectan directamente la dinámica del sistema en términos de la salida estimada, ya que estos dos estados se usan principalmente para calcular el pH del sistema, por lo que los estados del modelo reducido quedan:

$$x = \begin{bmatrix} X_1 & X_2 & S_1 & S_2 \end{bmatrix}^T \quad (4.33)$$

donde X_1 y X_2 son concentraciones de biomasa, S_1 es la concentración de sustrato orgánico y S_2 corresponde a la concentración de ácidos grasos volátiles (VFA). La entrada manipulada

corresponde a la tasa de dilución, quedando de la siguiente forma:

$$u = D \quad (4.34)$$

y la salida controlada es el flujo de metano:

$$y = q_M \quad (4.35)$$

Como se puede notar, se paso de un sistema MIMO a un sistema SISO con cuatro estados presentes. El modelo reducido AM2 usado por el controlador está dado por las siguientes ecuaciones:

$$\dot{X}_1 = \mu_1(S_1)X_1 - \alpha DX_1 \quad (4.36)$$

$$\dot{X}_2 = \mu_2(S_2)X_2 - \alpha DX_2 \quad (4.37)$$

$$\dot{S}_1 = D(S_{1,in} - S_1) - k_1\mu_1(S_1)X_1 \quad (4.38)$$

$$\dot{S}_2 = D(S_{2,in} - S_2) + k_2\mu_1(S_1)X_1 - k_3\mu_2(S_2)X_2 \quad (4.39)$$

En donde $S_{1,in}$ y $S_{2,in}$ representan la concentracion inicial de los substratos en el sistema, se cuenta también con las siguientes funciones de crecimiento:

$$\mu_1(S_1) = \frac{\mu_{max,1}S_1}{K_{S1} + S_1} \quad (4.40)$$

$$\mu_2(S_2) = \frac{\mu_0 S_2}{K_{S2} + S_2 + \frac{S_2^2}{K_{I2}}} \quad (4.41)$$

Estas funciones de crecimiento se programaron en el archivo `am2_model.py`, en donde se puede ver la implementación de este en al anexo A.1.

Para la salida de metano se tiene la siguiente ecuación:

$$q_M = k_6\mu_2(S_2)X_2 \quad (4.42)$$

Esta implementación tambien se realiza en el archivo `am2_model.py` y se puede ver en el anexo A.1. En el controlador, esta salida se trabaja en unidades de $[\text{mmol/L/D}]$ pero la planta ADM1 entrega esta salida en unidades de $gCOD/L/d$ por lo que también es necesario hacer un cambio de unidades en el momento de mandar la señal, para dejar la salida de metano en unidades $gCOD/L/d$ se hace mediante el factor de conversión 64/1000, quedando de esta forma:

$$y_{gCOD} = q_M \frac{64}{1000} \quad (4.43)$$

El perfil de referencias deseado por el usuario tambien esta en unidades de $gCOD/L/d$, por lo que hay que tener cuidado en que unidades se trabaja.

Las valores de los siguientes parametros: α , k_1 , k_2 , k_3 , k_6 , $\mu_{max,1}$, μ_0 , K_{S1} , K_{S2} y K_{I2} se toman de [36] y se definen en el archivo `config.py`, archivo que despues sera utilizado para la configuracion de todo el sistema. Los valores se pueden ver en la tabla 3.1 que se reviso en los resultados previos.

4.2.2.1. Python.

El controlador del problema de interés fue implementado en Python y ejecutado en la Raspberry Pi. A diferencia del problema simple, donde el controlador estaba concentrado en un único archivo, esta implementación se separó en módulos para facilitar lectura, depuración y modificación. El archivo `main.py` coordina el lazo de control, `config.py` contiene parámetros de simulación y comunicación, el archivo `am2_model.py` define el modelo AM2 y los respectivos Jacobianos necesarios, en el archivo `ekf.py` se implementa el Filtro de Kalman Extendido, en el archivo `nmpc.py` se formula y resuelve el problema NMPC, y por ultimo en `modbus_comm.py` se encapsula la comunicación Modbus TCP.

En cada instante de muestreo, el programa principal espera a que MATLAB actualice el contador de paso en Modbus. Luego lee la medición de metano q_M , ejecuta el EKF para obtener las estimaciones de estado, calcula el perfil de referencia dentro del horizonte de predicción y resuelve el NMPC. Finalmente, escribe la nueva tasa de dilución D junto con las estimaciones del observador y activa la señal `pi_ready`, indicando a la planta en MATLAB que la acción de control está disponible.

El periodo de muestreo usado es $T_s = 0,1$ días, consistente con la dinámica lenta del proceso anaerobio. La simulación se configuro para hacer un total de 20 días, con una condición inicial de control $D_0 = 0,34 d^{-1}$. El perfil de referencia utilizado para validar el seguimiento de metano considera cambios escalonados en $t = 0, 5, 10$ y 15 días, con valores $2,8, 3,5, 3,8$ y $4,1 gCOD/L/d$, respectivamente.

4.2.2.2. NMPC.

El modelo utilizado para el NMPC para predecir la evolución del sistema corresponde al modelo AM2 presentado en la Sección 4.2.2, integrado numéricamente para obtener un paso discreto de predicción. De forma general, este modelo de predicción puede expresarse como:

$$x_{t+1} = f(x_t, u_t) \quad (4.44)$$

$$y_t = g(x_t, u_t) \quad (4.45)$$

donde f corresponde a la integración de un paso de muestreo del modelo AM2 continuo y g es la función de salida de metano, expresada en $gCOD/L/d$. Al igual que en el EKF (Sección 4.2.2.3), el uso de este modelo reducido responde a que la planta ADM1 solo entrega como medición el flujo de metano q_M , sin acceso directo a sus estados internos.

Con los estados estimados $\hat{x}_{k|k}$ entregados por el EKF como condición inicial, el NMPC

resuelve en cada instante de muestreo el siguiente problema de optimización:

$$J = \min_{\Delta U} \sum_{j=0}^{N_{ho}-1} \left[\|y_{sp,j|k} - \hat{y}_{j|k}\|_Q^2 + \|\Delta D_{j|k}\|_R^2 \right] \quad (4.46)$$

$$\text{s.a. } x_{j+1|k} = f(x_{j|k}, D_{eff,j|k}), \quad j = 0, \dots, N_{ho} - 1 \quad (4.47)$$

$$\hat{y}_{j|k} = g(x_{j|k}) \quad (4.48)$$

$$D_{eff,j|k} = D_{j|k} + \hat{d}_k \quad (4.49)$$

$$D_{j|k} = D_{j-1|k} + \Delta D_{j|k}, \quad D_{-1|k} = D_{k-1} \quad (4.50)$$

$$D_{min} \leq D_{j|k} \leq D_{max} \quad (4.51)$$

$$\Delta D_{min} \leq \Delta D_{j|k} \leq \Delta D_{max} \quad (4.52)$$

$$x_{min} \leq x_{j|k} \leq x_{max} \quad (4.53)$$

$$x_{0|k} = \hat{x}_{k|k} \quad (4.54)$$

donde $\Delta U = [\Delta D_{0|k} \ \Delta D_{1|k} \ \dots \ \Delta D_{N_{hi}-1|k}]^T$ es la variable de decisión y $\hat{d}_k$ es la perturbación estimada por el EKF, que se asume constante a lo largo de todo el horizonte de predicción.

En la implementación se utiliza un horizonte de predicción $N_{ho} = 4$ y un horizonte de control $N_{hi} = 4$. La tasa de dilución se restringe al intervalo $0 \leq D \leq 1$, el incremento de control se acota mediante $-0,35 \leq \Delta D \leq 0,35$, y los pesos seleccionados son $W_y = 70$ para el error de seguimiento del setpoint y $W_{\Delta D} = 20$ para la penalización de la tasa de cambio del control.

El modelo de predicción $f(x_{j|k}, D_{eff,j|k})$ es el modelo F_{RK4} que corresponde al AM2 continuo integrado numéricamente mediante Runge-Kutta de cuarto orden. En la implementación en Python, esta integración reemplaza a `ode45`, utilizado en la versión original de MATLAB. Además, de acuerdo con la formulación del trabajo base, el modelo incorpora el factor de mortalidad $0,96 = (1 - 0,04)$ en las ecuaciones de crecimiento de biomasa.

Siguiendo el enfoque de control predictivo libre de offset (*offset-free MPC*) propuesto en [22], el NMPC no aplica directamente la entrada calculada $D_{j|k}$ al modelo de predicción, sino la entrada efectiva $D_{eff,j|k} = D_{j|k} + \hat{d}_k$. Esta perturbación representa principalmente el error de predicción asociado a controlar una planta de mayor complejidad (ADM1) con un modelo reducido (AM2). Al incorporarla en el modelo de predicción, el controlador adquiere una acción de tipo integral que permite reducir el error estacionario frente a diferencias persistentes entre ambos modelos.

Tal como se planteó en el problema de optimización, la variable de decisión no es directamente la entrada absoluta D , sino la secuencia de incrementos futuros de la tasa de dilución. La entrada absoluta se reconstruye acumulando dichos incrementos a partir de la acción aplicada en el instante anterior, $D_{-1|k} = D_{k-1}$. Esta formulación permite penalizar

directamente los cambios bruscos de la tasa de dilución en la función de costo, en lugar de penalizar la magnitud absoluta de D , favoreciendo una operación más suave del actuador.

El problema no lineal se construye en CasADi y se resuelve mediante el solver IPOPT. Siguiendo la estrategia de horizonte deslizante, en cada ciclo se aplica únicamente la primera acción óptima de la secuencia calculada:

$$D_k = D_{k-1} + \Delta D_{0|k}^* \quad (4.55)$$

En el siguiente instante de muestreo se recibe una nueva medición, se actualiza el EKF y se resuelve nuevamente el problema de optimización. Para mejorar el tiempo de convergencia del solver, se utiliza *warm start*, desplazando la secuencia óptima de incrementos calculada en el ciclo anterior como condición inicial de la siguiente iteración.

Explicado todo el procedimiento, el código completo de esta implementación se encuentra en el apéndice A.2.

4.2.2.3. Extended Kalman Filter.

Como la medición disponible para cerrar el lazo es el flujo de metano q_M , el controlador no conoce directamente las concentraciones internas X_1 , X_2 , S_1 y S_2 . Para reconstruir estas variables se implementa un Filtro de Kalman Extendido sobre el modelo AM2. Además, siguiendo la estrategia libre de offset descrita en [36], el observador incluye un estado adicional de perturbación asociado a la entrada. El estado extendido se define como:

$$x_k^e = \begin{bmatrix} X_1 & X_2 & S_1 & S_2 & d \end{bmatrix}^T \quad (4.56)$$

El modelo discreto extendido usado por el EKF se obtiene mediante integración de Euler:

$$x_{k+1} = x_k + T_s f(x_k, D_k) \quad (4.57)$$

y se aumenta con el efecto de la perturbación d_k sobre la entrada:

$$x_{k+1}^e = f_a(x_k^e, D_k) = \begin{bmatrix} f_f(x_k, D_k) + \frac{\partial f_f}{\partial D} d_k \\ d_k \end{bmatrix} \quad (4.58)$$

donde $f_f(x_k, D_k)$ corresponde al paso discreto nominal del modelo AM2. La perturbación se modela como constante, es decir:

$$d_{k+1} = d_k \quad (4.59)$$

La función de salida del EKF es:

$$y_k = g_a(x_k^e, D_k) = k_6 \mu_2(S_{2,k}) X_{2,k} \quad (4.60)$$

Por lo tanto, el observador trabaja directamente con la medición de metano en $mmol/L/d$, y luego el controlador convierte dicha salida a $gCOD/L/d$ cuando se compara con la referencia.

En cada ciclo se ejecutan las dos etapas del EKF. Primero se realiza la corrección usando la medición disponible:

$$K_k = P_{k|k-1} C_k^T (R + C_k P_{k|k-1} C_k^T)^{-1} \quad (4.61)$$

$$\hat{x}_{k|k}^e = \hat{x}_{k|k-1}^e + K_k \left(y_k - g_a(\hat{x}_{k|k-1}^e, D_k) \right) \quad (4.62)$$

$$P_{k|k} = (I - K_k C_k) P_{k|k-1} \quad (4.63)$$

donde C_k es el Jacobiano de la función de salida respecto al estado extendido:

$$C_k = \left. \frac{\partial g_a}{\partial x^e} \right|_{\hat{x}_{k|k-1}^e, D_k} \quad (4.64)$$

Luego se realiza la predicción para el siguiente instante:

$$\hat{x}_{k+1|k}^e = f_a(\hat{x}_{k|k}^e, D_k) \quad (4.65)$$

$$P_{k+1|k} = Q + A_k P_{k|k} A_k^T \quad (4.66)$$

donde:

$$A_k = \left. \frac{\partial f_a}{\partial x^e} \right|_{\hat{x}_{k|k}^e, D_k} \quad (4.67)$$

Los Jacobianos A_k y C_k se implementan analíticamente en `am2_model.py`, evitando el cálculo por diferencias finitas durante la ejecución en la Raspberry Pi.

Las condiciones iniciales del EKF son:

$$\hat{x}_0^e = \begin{bmatrix} 0,7294 & 0,8109 & 0,2364 & 2,5144 & 0 \end{bmatrix}^T \quad (4.68)$$

La covarianza inicial se define como $P_0 = I$, el ruido de proceso como:

$$Q = \text{diag}(0,001, 0,001, 0,001, 0,001, 0,01) \quad (4.69)$$

y el ruido de medición como:

$$R = 50 \quad (4.70)$$

Finalmente, el vector entregado al NMPC queda compuesto por las estimaciones físicas $\hat{X}_1$, $\hat{X}_2$, $\hat{S}_1$ y $\hat{S}_2$, junto con la perturbación estimada $\hat{d}$. Así, el EKF cumple dos funciones simultáneas: reconstruir los estados no medidos necesarios para la predicción y compensar el sesgo persistente entre la planta ADM1 y el modelo reducido AM2 usado por el controlador.

El código completo del EKF se encuentra en el Apéndice A.3, y las funciones del modelo AM2 y sus Jacobianos analíticos en el Apéndice A.1.

RESULTADOS.

En este capítulo se presentan los resultados obtenidos para las dos etapas de validación desarrolladas en el trabajo. Primero se analiza el problema simple, cuyo objetivo principal fue comprobar que la arquitectura de comunicación Modbus TCP permite cerrar correctamente el lazo entre una planta simulada en MATLAB y un controlador ejecutado en Python sobre la Raspberry Pi. Luego se presentan los resultados del problema de interés, donde la misma arquitectura se utiliza para controlar la producción de metano de una planta ADM1 mediante un controlador NMPC basado en el modelo reducido AM2 y un EKF aumentado.

La lectura de los resultados debe considerar que ambas pruebas no buscan validar solamente el desempeño de los algoritmos de control, sino también la ejecución distribuida del lazo. Por lo tanto, las curvas obtenidas incorporan el efecto del intercambio de datos por registros Modbus, el escalamiento de variables, el tiempo de cómputo del optimizador y la sincronización entre MATLAB y Python.

5.1. Problema Simple.

La primera prueba corresponde al sistema lineal de tercer orden descrito en la sección 4.1. La planta fue ejecutada en MATLAB y el controlador MPC junto con el observador fueron ejecutados en Python. La referencia aplicada corresponde a un escalón unitario luego de un segundo inicial de reposo, seguido por una vuelta a cero alrededor de los 21 s. Con esto se busca verificar tanto el seguimiento de referencia como la capacidad del controlador para reaccionar ante cambios positivos y negativos del setpoint.

En la figura 5.1 se observa que la salida de la planta sigue correctamente la referencia aplicada. Durante el primer segundo la salida permanece cercana a cero, lo que confirma que el lazo parte desde la condición de reposo esperada. Al aplicar el escalón, el controlador

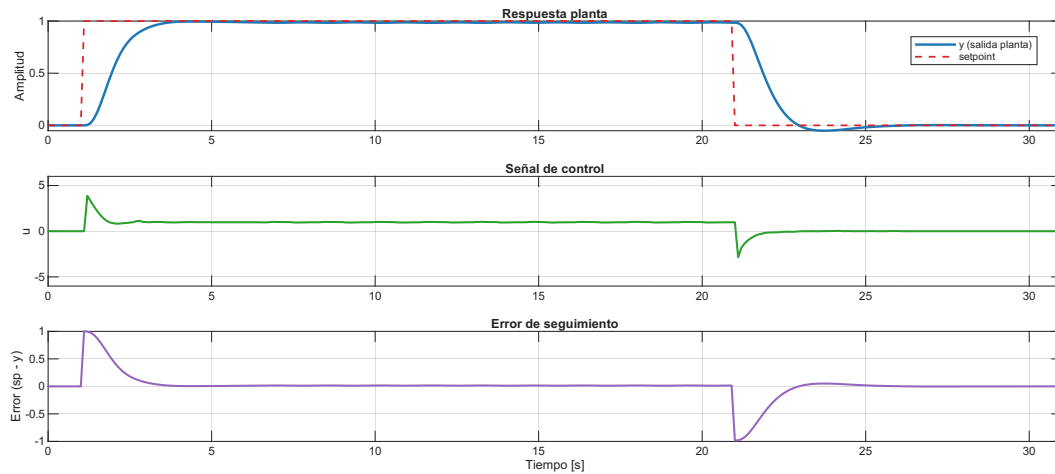


Figura 5.1: Respuesta de la planta simple ante una referencia escalonada.

incrementa rápidamente la señal de control y la salida comienza a crecer de manera suave hasta alcanzar el valor de referencia. La respuesta no presenta sobrepaso apreciable y el error de seguimiento converge a cero luego del transitorio inicial.

La señal de control muestra un incremento pronunciado al momento del cambio de referencia, seguido por una disminución progresiva hasta establecerse alrededor del valor requerido para mantener la salida en régimen permanente. Este comportamiento es consistente con la ganancia estática unitaria de la planta: una vez que la salida llega a la referencia, la entrada necesaria para sostener el punto de operación se mantiene cercana a un valor constante. Además, la acción de control permanece dentro de los límites impuestos en el MPC, por lo que no se observa una operación dominada por saturación.

Cuando la referencia vuelve a cero, el controlador aplica una acción negativa para descargar la dinámica acumulada de la planta. En este segundo transitorio se aprecia nuevamente una convergencia suave hacia el nuevo punto de operación, con un error que cambia de signo de forma esperada y luego tiende a cero. La pequeña oscilación alrededor del valor final es acotada y no compromete la estabilidad del lazo.

Estos resultados permiten validar la arquitectura cliente-servidor propuesta para el primer caso de prueba. En particular, se comprueba que MATLAB puede escribir la salida de planta y la referencia en el servidor Modbus, que Python puede leer dichas variables, calcular la acción de control y escribirla nuevamente, y que el ciclo completo se mantiene sincronizado con el periodo de muestreo utilizado. Por lo tanto, la prueba simple confirma que la comunicación, el escalamiento de datos y el cierre del lazo funcionan antes de pasar al caso no-lineal de biogás.

5.2. Problema de interés.

Luego de validar la arquitectura con el problema simple, se ejecutó el caso de interés asociado a la planta de digestión anaeróbica. En esta prueba, MATLAB implementa la planta ADM1 y Python ejecuta el EKF aumentado junto con el controlador NMPC. La variable controlada es la producción de metano expresada en $gCOD/L/d$, mientras que la variable manipulada es la tasa de dilución D .

Las figuras de esta sección comparan dos resultados. La curva denominada “Original” corresponde a la simulación base usada como referencia de comparación [36], mientras que la curva “HIL planta” corresponde al resultado obtenido con la implementación distribuida entre MATLAB, servidor Modbus y Raspberry Pi. Esta comparación permite evaluar si el controlador implementado en hardware reproduce el comportamiento esperado de la simulación, aun cuando aparecen efectos adicionales de comunicación, cuantización y tiempo de cómputo.

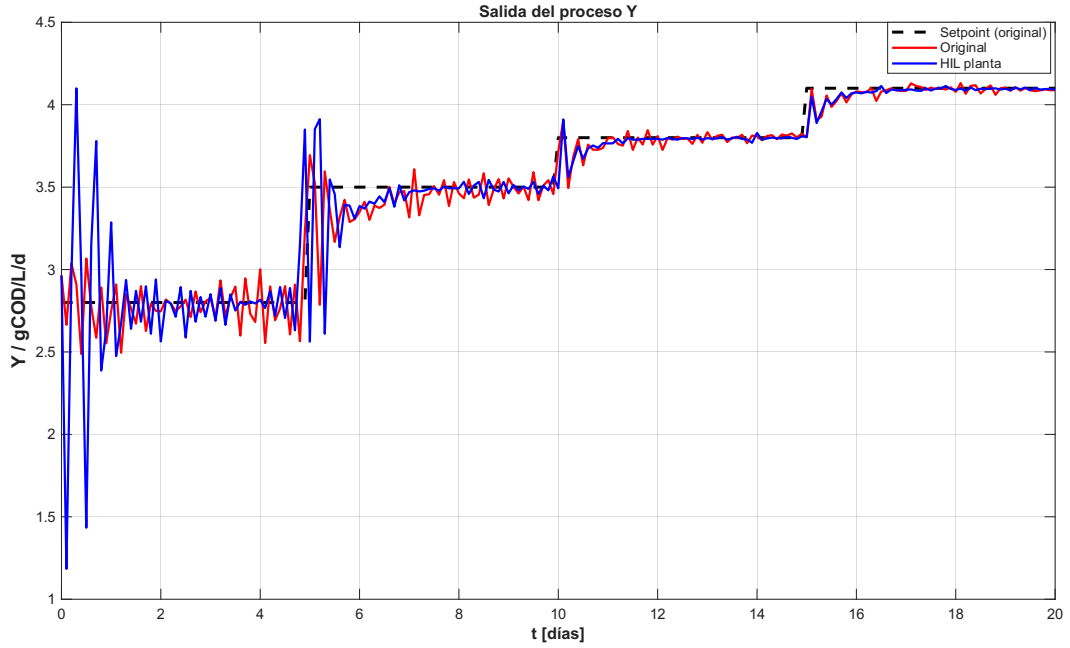


Figura 5.2: Seguimiento de referencia de la producción de metano para el problema de interés.

La Figura 5.2 muestra la respuesta de la salida controlada frente al perfil de referencia. La Tabla 5.1 cuantifica esta respuesta en cada uno de los cuatro segmentos de setpoint. En los primeros instantes se observa un transitorio marcado en ambas curvas, asociado a la inicialización del EKF, a la diferencia entre la condición inicial estimada y la dinámica de la planta ADM1, y al ajuste inicial de la perturbación estimada: al finalizar

el primer segmento ($t = 4,9$ días), el error de seguimiento alcanza un 31,9 % en la curva Original y un 37,5 % en la curva HIL, con un error cuadrático medio (RMSE) de 0,196 y 0,436 $gCOD/L/d$ respectivamente. El máximo error absoluto durante los primeros dos días es de 0,312 $gCOD/L/d$ en la simulación Original, frente a 1,616 $gCOD/L/d$ en la implementación HIL, evidenciando que el transitorio inicial es considerablemente más pronunciado en la implementación distribuida.

Tabla 5.1: Error de seguimiento de Y por segmento de setpoint.

Segmento	Error final Orig.	Error final HIL	RMSE Orig.	RMSE HIL
sp= 2,8	+0,894 (+31,9 %)	+1,050 (+37,5 %)	0,196	0,436
sp= 3,5	+0,217 (+6,2 %)	+0,066 (+1,9 %)	0,149	0,213
sp= 3,8	+0,002 (+0,05 %)	+0,008 (+0,2 %)	0,062	0,067
sp= 4,1	-0,009 (-0,2 %)	-0,009 (-0,2 %)	0,048	0,060
Total			0,129	0,246

Cuando la referencia aumenta alrededor de los 5 días, el controlador eleva la producción de metano hacia el nuevo valor. En esta zona aparece un transitorio visible, pero la salida vuelve a acercarse a la referencia luego de algunos instantes de muestreo, reduciendo el error final de este segundo tramo a 6,2 % en la curva Original y 1,9 % en la curva HIL. Para los cambios posteriores, alrededor de los 10 y 15 días, la respuesta HIL se mantiene muy cercana a la simulación original: el error final cae por debajo de 0,3 % en ambas curvas desde el tercer segmento en adelante. En particular, desde aproximadamente el día 11 en adelante, ambas curvas presentan una diferencia pequeña y siguen la referencia sin error estacionario apreciable, tal como refleja el RMSE del último segmento (0,048 Original, 0,060 HIL, la brecha más estrecha de toda la prueba).

El resultado anterior es relevante porque el controlador no utiliza el modelo ADM1 completo, sino el modelo reducido AM2. Por lo tanto, la capacidad de seguimiento observada depende de dos elementos: la predicción no lineal del NMPC y la compensación del sesgo entre modelo y planta mediante el estado de perturbación estimado por el EKF. La cercanía entre las curvas Original y HIL, que se traduce en un RMSE total de 0,129 $gCOD/L/d$ para la primera y 0,246 $gCOD/L/d$ para la segunda, indica que la implementación en la Raspberry Pi conserva el comportamiento principal del controlador, con una penalización adicional concentrada casi en su totalidad en el transitorio inicial.

La Figura 5.3 presenta el error de seguimiento $Y - Y_{sp}$, cuyos valores numéricos por segmento ya se detallaron en la Tabla 5.1. Durante el inicio de la simulación se observan oscilaciones de mayor amplitud, especialmente en la curva HIL. Estas oscilaciones son coherentes con el transitorio de estimación inicial: el EKF debe corregir los estados internos no medidos usando solamente la medición de metano, mientras el NMPC comienza a ajustar

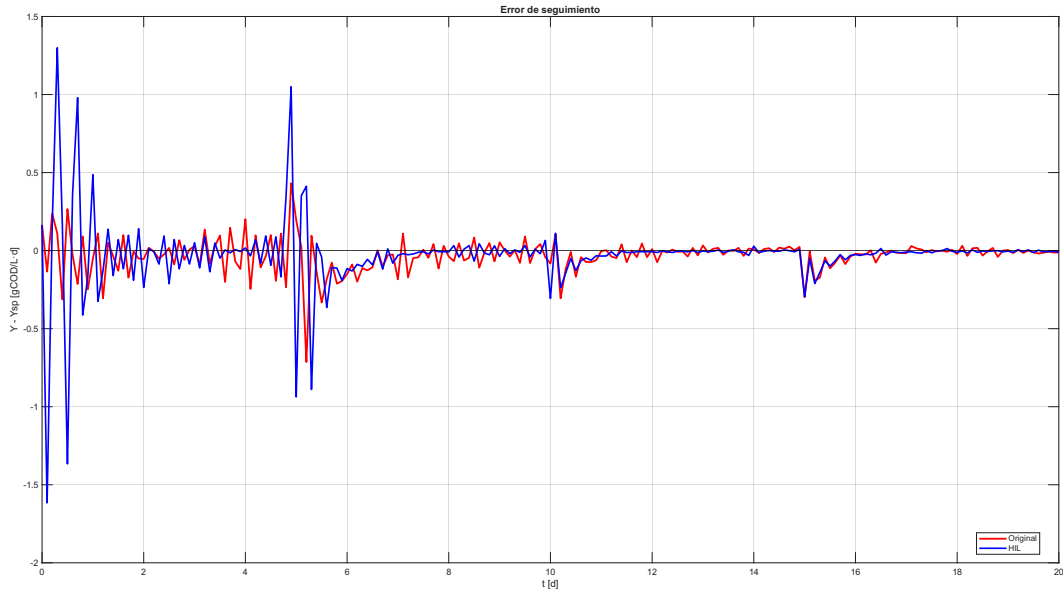


Figura 5.3: Error de seguimiento de la producción de metano.

la tasa de dilución.

Después de este periodo inicial, el error se mantiene cercano a cero. En los cambios de referencia se producen desviaciones transitorias, pero estas disminuyen rápidamente y no se observa acumulación de error permanente. Esto confirma que la acción integral indirecta introducida por la estimación del disturbio cumple su función principal: compensar diferencias persistentes entre el modelo AM2 usado por el controlador y la planta ADM1 usada como emulador.

La señal de control se muestra en la Figura 5.4, y la Tabla 5.2 resume la tasa de dilución alcanzada al final de cada segmento. La entrada HIL presenta variaciones importantes durante el primer día, lo que coincide con el periodo de mayor error y con la adaptación inicial del observador; en este primer segmento, D difiere en $-0,167 d^{-1}$ entre ambas implementaciones (0,423 Original frente a 0,256 HIL), siendo la mayor discrepancia de control observada en toda la prueba.

Tabla 5.2: Tasa de dilución D al final de cada segmento.

Segmento	D Original	D HIL	Diferencia
sp= 2,8	0,4228	0,2560	-0,1668
sp= 3,5	0,4836	0,4900	+0,0064
sp= 3,8	0,5957	0,5860	-0,0097
sp= 4,1	0,6906	0,6970	+0,0064

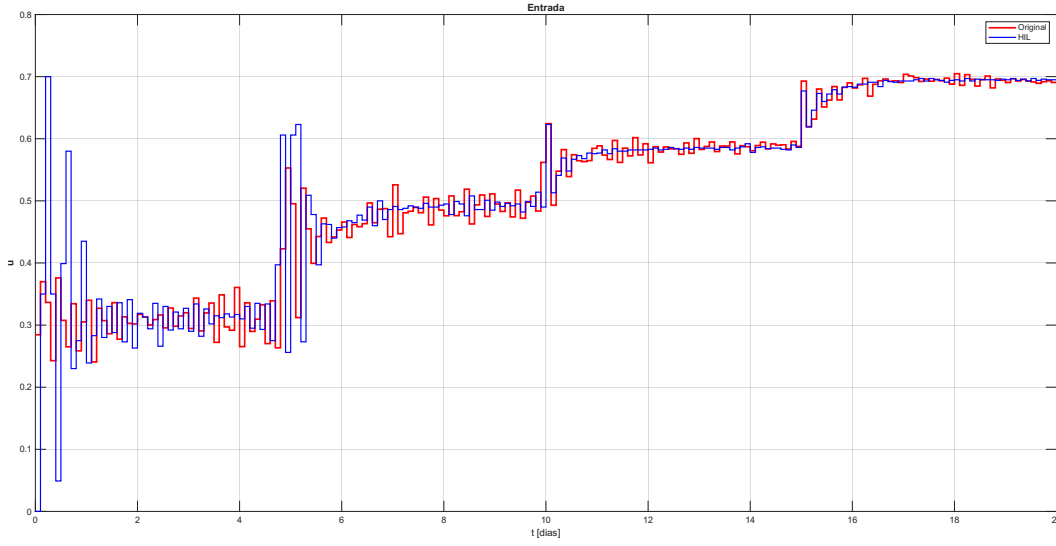


Figura 5.4: Tasa de dilución calculada por el controlador NMPC.

En los segmentos posteriores, en cambio, la diferencia entre ambas señales de control se reduce a menos de $0,01 d^{-1}$: en torno a $0,58 d^{-1}$ hacia el día 10 y $0,69 d^{-1}$ hacia el día 15, prácticamente idénticos entre Original y HIL. Esto confirma que la mayor parte de la discrepancia de control entre ambas implementaciones se concentra en el transitorio inicial, y no en el régimen ya establecido. La forma escalonada de la curva se debe a que el controlador opera en tiempo discreto y escribe una nueva acción en cada instante de muestreo. Además, la entrada permanece dentro del rango admisible $0 \leq D \leq 1$, respetando las restricciones impuestas en el problema NMPC.

Las Figuras 5.5 y 5.6 muestran la evolución de los estados estimados por el EKF, y la Tabla 5.3 cuantifica el error cuadrático medio de cada estimación respecto del estado real de la planta, separado por implementación. En el caso de las biomásas X_1 y X_2 , las curvas HIL siguen la tendencia general de la simulación Original, aunque con diferencias apreciables en algunos intervalos: el RMSE de X_1 frente al estado real crece de 0,010 (Original) y 0,056 (HIL) en el primer segmento hasta 0,164 y 0,217 respectivamente en el último segmento. Esto es esperable, ya que los estados internos no se miden directamente y el EKF debe reconstruirlos a partir de una única salida: el flujo de metano. Por esta razón, el objetivo principal del observador no es reproducir exactamente cada estado interno de ADM1, sino entregar al NMPC una estimación consistente con la salida medida y útil para la predicción.

El estado S_2 presenta transitorios mucho más notorios que el resto, tanto en la simulación Original como en la implementación HIL, con un RMSE respecto del estado real que supera 3 en el primer segmento de la simulación Original y llega a 7,3 en el primer segmento de la implementación HIL (el mayor error de estimación observado entre todos los estados y

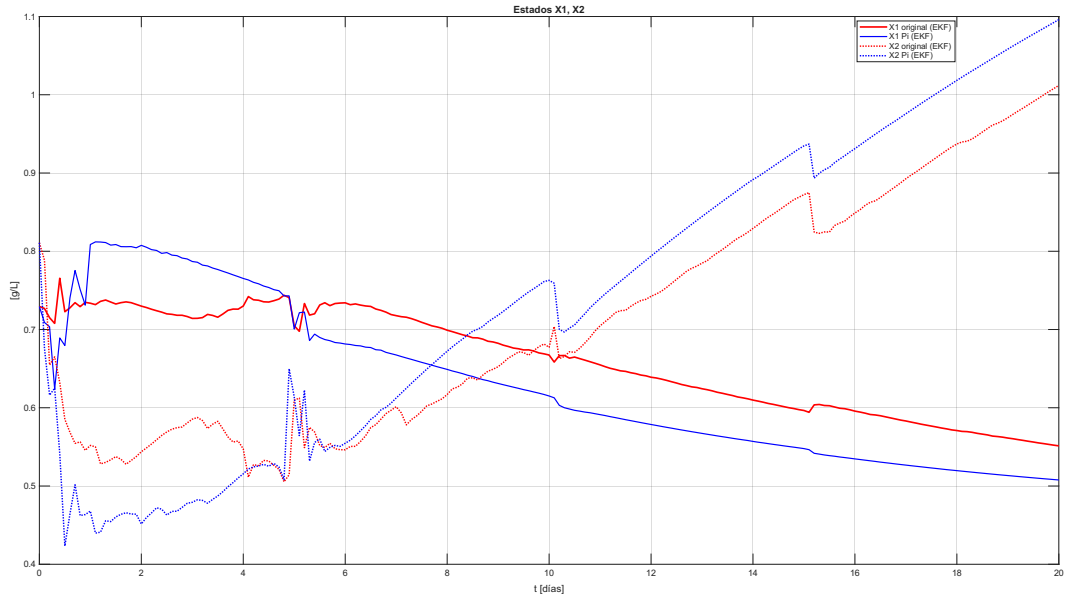


Figura 5.5: Estimaciones de biomasa X_1 y X_2 obtenidas por el EKF.

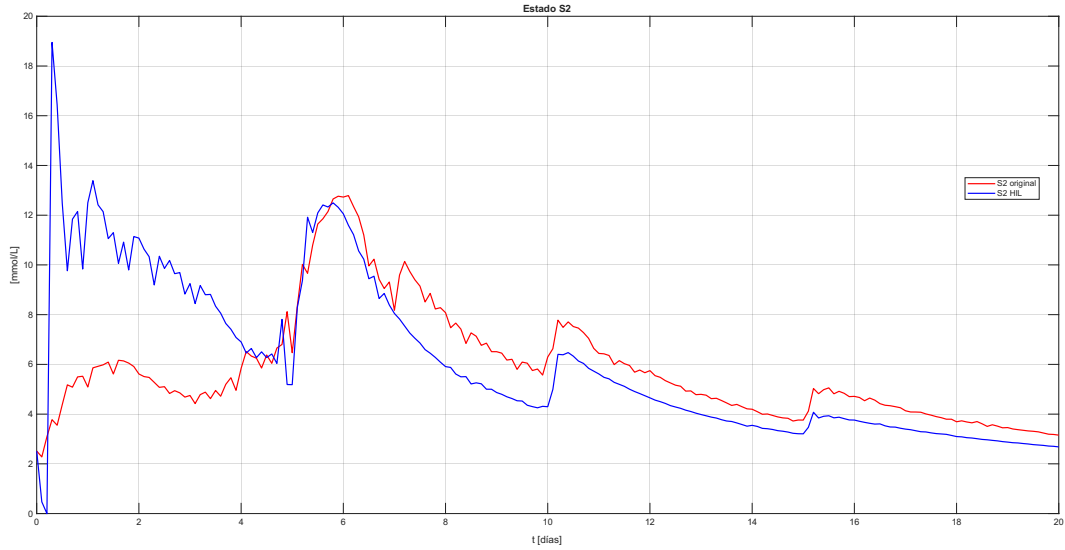


Figura 5.6: Estimación del estado S_2 obtenida por el EKF.

segmentos de esta prueba). Este comportamiento es importante porque S_2 está directamente relacionado con la etapa metanogénica del modelo AM2 y, por lo tanto, influye en el cálculo de q_M . Aun así, después de los transitorios principales, la estimación HIL mantiene una evolución coherente con la tendencia original y no presenta divergencia. Esto indica que el

Tabla 5.3: RMSE de la estimación EKF respecto del estado real de la planta, por implementación.

Segmento	Original				HIL			
	X_1	X_2	S_1	S_2	X_1	X_2	S_1	S_2
sp= 2,8	0.0099	0.2480	0.0758	3.017	0.0562	0.3142	0.0868	7.339
sp= 3,5	0.0364	0.2078	0.1354	5.738	0.0805	0.1806	0.1541	4.930
sp= 3,8	0.1075	0.0775	0.1432	1.680	0.1640	0.0697	0.1709	1.073
sp= 4,1	0.1643	0.1159	0.1658	1.277	0.2173	0.1931	0.1898	1.884
Total	0.1004	0.1759	0.1344	3.401	0.1451	0.2082	0.1555	4.542

EKF permanece acotado durante toda la simulación y entrega información suficiente para que el NMPC mantenga el seguimiento de la salida.

Como verificación adicional, la Tabla 5.4 compara directamente la estimación obtenida por el EKF de la simulación Original contra la del EKF de la implementación HIL. La concordancia entre ambas es sustancialmente mayor que la que cada una presenta por separado frente al estado real (por ejemplo, el RMSE cruzado total de S_2 es de 2,728, frente a 3,401 y 4,542 que cada implementación exhibe respecto de la verdad), lo que indica que ambos observadores convergen de manera consistente hacia una misma estimación, en lugar de divergir de forma independiente el uno del otro.

Tabla 5.4: RMSE de la estimación EKF entre la implementación Original y la HIL.

Segmento	X_1	X_2	S_1	S_2
sp= 2,8	0.0581	0.0817	0.1050	5.112
sp= 3,5	0.0486	0.0487	0.0358	1.531
sp= 3,8	0.0581	0.0534	0.0287	0.931
sp= 4,1	0.0539	0.0829	0.0243	0.749
Total	0.0548	0.0686	0.0584	2.728

La Figura 5.7 muestra la estimación del disturbio de entrada, y la Tabla 5.5 resume su valor al final de cada segmento. Las curvas asociadas a MATLAB y a la Raspberry Pi se superponen casi completamente: la diferencia entre el valor calculado en Python y el recibido en MATLAB vía Modbus nunca supera 0,0004 en ningún segmento, lo que valida que el valor estimado por el EKF se transmite correctamente mediante Modbus y que no existen diferencias relevantes entre lo calculado por el controlador y lo registrado por la planta. La estimación parte con un transitorio inicial, luego cambia de forma marcada cerca de los instantes donde se modifican las referencias y finalmente converge hacia valores negativos de mayor magnitud, pasando de +0,097 en el primer segmento a -0,320 en el último.

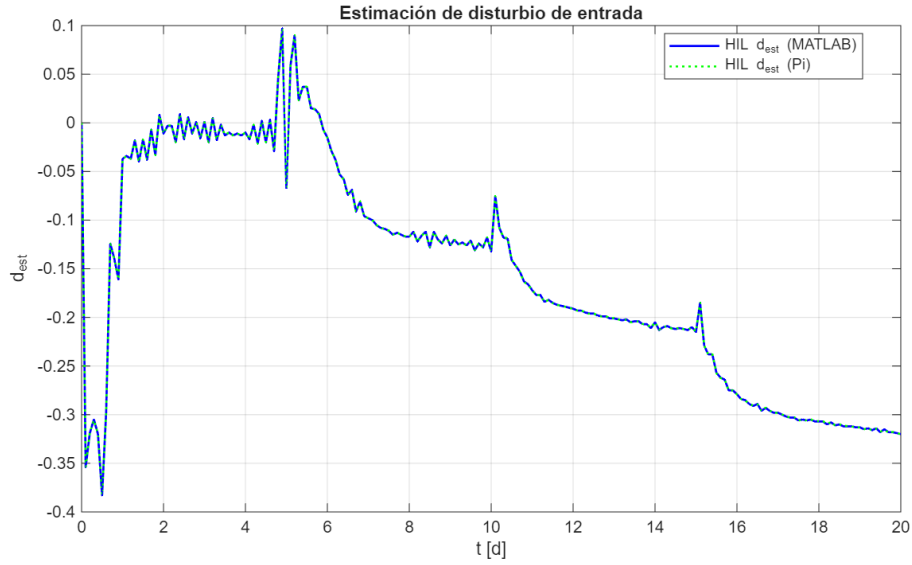


Figura 5.7: Estimación del disturbio de entrada utilizada para compensación libre de offset.

Tabla 5.5: Perturbación estimada $\hat{d}_k$ al final de cada segmento.

Segmento	$\hat{d}$ Original	$\hat{d}$ HIL (Python)
sp= 2,8	+0,0815	+0,0974
sp= 3,5	-0,1270	-0,1176
sp= 3,8	-0,1970	-0,2104
sp= 4,1	-0,2990	-0,3200

Este comportamiento confirma que la perturbación estimada no actúa como una variable extra del observador, sino como un término activo de compensación. Su valor ajusta la entrada efectiva usada por el modelo de predicción y permite reducir el sesgo producido por controlar ADM1 con AM2. En consecuencia, el seguimiento sin error estacionario apreciable observado en la Figura 5.2 y cuantificado en la Tabla 5.1 se explica en parte por la evolución de esta estimación.

En conjunto, los resultados del problema de interés muestran que el lazo NMPC-EKF implementado sobre la Raspberry Pi logra controlar la planta ADM1 simulada en MATLAB a través de la arquitectura Modbus TCP. La salida sigue el perfil de referencia con un RMSE total de $0,246 \text{ gCOD/L/d}$ (frente a $0,129 \text{ gCOD/L/d}$ de la simulación Original), la entrada respeta las restricciones de operación con diferencias de control inferiores a $0,01 \text{ d}^{-1}$ una vez superado el transitorio inicial, el error de seguimiento se mantiene por debajo de $0,3\%$ desde el tercer segmento en adelante, y el observador entrega estimaciones

acotadas de estados y disturbio. El estado S_2 concentra la mayor discrepancia de estimación en ambas implementaciones, constituyendo el principal problema abierto identificado en esta validación. Por lo tanto, la implementación HIL reproduce de manera satisfactoria el comportamiento de la simulación original y constituye una validación funcional de la arquitectura propuesta para el control predictivo no lineal del proceso de biogás.

CONCLUSIONES.

6.1. Conclusiones generales.

En esta memoria se implementó y validó una arquitectura de control predictivo no lineal para un proceso de digestión anaeróbica, orientada al seguimiento de una referencia de producción de metano. La propuesta consideró una separación física y lógica entre la planta y el controlador: la planta se ejecutó en MATLAB/Simulink usando ADM1 como emulador de mayor fidelidad, mientras que el controlador NMPC y el filtro de Kalman extendido se ejecutaron en Python sobre una Raspberry Pi. La comunicación entre ambos procesos se realizó mediante un servidor Modbus TCP, lo que permitió evaluar no solo el algoritmo de control, sino también su operación en una arquitectura Hardware-in-the-Loop.

La primera etapa, basada en una planta lineal simple de tercer orden, permitió verificar los elementos básicos de la implementación distribuida. En particular, se comprobó que MATLAB podía escribir mediciones y referencias en el servidor Modbus, que Python podía leer dichas variables, calcular la acción de control y devolverla a la planta, y que el protocolo de sincronización implementado evitaba desfases entre el cálculo de la acción y su aplicación. Esta prueba fue relevante porque permitió aislar problemas de comunicación, escalamiento y sincronización antes de abordar la dinámica no lineal del proceso de biogás.

En el problema de interés se logró cerrar el lazo NMPC-EKF usando como medición principal la producción de metano y como variable manipulada la tasa de dilución D . El controlador utilizó el modelo reducido AM2 para predicción, mientras que la planta virtual correspondió a ADM1. Esta diferencia entre modelo y planta representa una condición exigente, ya que el controlador debe operar con un modelo más simple que el proceso que se desea regular. Los resultados muestran que la implementación HIL reproduce de forma satisfactoria el comportamiento de la simulación original: la salida sigue el perfil de

referencia, la entrada respeta las restricciones $0 \leq D \leq 1$ y el error de seguimiento se reduce significativamente luego de los transitorios iniciales.

Desde el punto de vista cuantitativo, la implementación HIL obtuvo un RMSE total de seguimiento de $0,246 \text{ gCOD/L/d}$, frente a $0,129 \text{ gCOD/L/d}$ de la simulación original. La mayor diferencia se concentró en el primer tramo, donde el EKF aún se encontraba ajustando las estimaciones iniciales de estados y perturbación. Sin embargo, desde el tercer segmento de referencia en adelante, el error final de seguimiento se mantuvo por debajo de $0,3\%$ en ambas implementaciones, lo que confirma que el controlador implementado en la Raspberry Pi conserva la capacidad de seguimiento del esquema original una vez superado el período inicial de convergencia.

La estimación de la perturbación cumplió un rol central en este desempeño. Al incorporar un estado adicional en el EKF, el controlador pudo compensar el sesgo persistente entre el modelo AM2 usado para predicción y la planta ADM1 usada como emulador. La perturbación estimada evolucionó de manera coherente con los cambios de referencia y permitió introducir una acción de tipo integral libre de offset.

También se verificó que el EKF entrega estimaciones acotadas de los estados internos requeridos por el NMPC. Las variables X_1 , X_2 y S_1 mantuvieron tendencias coherentes entre la implementación original y la HIL. El estado S_2 , asociado a los ácidos grasos volátiles y directamente relacionado con la etapa metanogénica, presentó las mayores discrepancias, especialmente durante el transitorio inicial. Este resultado no invalida la arquitectura, pero identifica un punto sensible del observador que debe considerarse en futuras mejoras, ya que S_2 influye directamente en la predicción de metano.

En consecuencia, se acepta la hipótesis planteada en esta memoria: es posible implementar una arquitectura Hardware-in-the-Loop en la que un controlador NMPC y un EKF, ejecutados fuera del entorno de simulación de la planta, logren cerrar el lazo de control y seguir una referencia de producción de metano mediante comunicación Modbus TCP. La memoria constituye así una etapa intermedia entre la validación puramente simulada y una futura implementación experimental, demostrando que el algoritmo puede operar en hardware embebido y comunicarse con una planta externa sin perder su comportamiento principal.

6.2. Trabajo Futuro

Como trabajo futuro, el paso más importante corresponde a implementar la arquitectura propuesta en una planta física de digestión anaeróbica. Esto permitiría validar el desempeño del controlador frente a sensores reales, ruido de medición, retardos de actuación, variaciones de temperatura, cambios en la composición del sustrato y perturbaciones que no están completamente representadas por el modelo ADM1. Para esta etapa será necesario integrar instrumentación de planta, actuadores para modificar la tasa de dilución y mecanismos de

seguridad que limiten acciones de control potencialmente riesgosas para el proceso biológico.

Otra línea relevante consiste en reemplazar o mejorar el controlador NMPC utilizado. Si bien el esquema implementado mostró buen desempeño, existen alternativas que podrían aumentar robustez o reducir costo computacional, tales como redes neuronales, MPC adaptativo, etc. También sería conveniente estudiar solvers más rápidos o técnicas de explicitación y aproximación del NMPC, especialmente si se desea migrar desde una Raspberry Pi hacia hardware industrial con restricciones más severas.

Asimismo, se recomienda mejorar el observador de estados, con especial atención a la estimación de S_2 . Algunas alternativas son reajustar las matrices de covarianza del EKF, usar un filtro de Kalman unscented, incorporar estimación conjunta de parámetros o aumentar la cantidad de mediciones disponibles, por ejemplo VFA, alcalinidad, pH o composición del biogás. Estas mejoras podrían reducir el transitorio inicial y aumentar la confiabilidad del controlador ante cambios bruscos de operación.

Finalmente, sería conveniente avanzar hacia una implementación más cercana a un entorno industrial. Esto incluye evaluar el uso de PLCs o controladores industriales, estudiar el comportamiento de la comunicación ante pérdidas de paquetes o retardos, incorporar rutinas de diagnóstico y fallas, y probar perfiles de referencia asociados a demandas energéticas reales. Con ello, la arquitectura podría evolucionar desde una validación HIL hacia una plataforma de control avanzada para operación flexible y segura de plantas de biogás.

CÓDIGOS PROBLEMA DE INTERES

A.1. Código am2_model.py

```
1 import numpy as np
2 from config import AM2.PARAMS, DT
3 # -----
4 # Funciones de tasa de crecimiento
5 # -----
6
7 def _mu1(S1: float, p: dict) -> float: # mu1 = mu_max1 * S1 / (Ks1 + S1)
8     linea 40 model_AM2.m
9     return (p["mu_max1"] * S1) / (p["Ks1"] + S1 + 1e-15)
10
11 def _mu2(S2: float, p: dict) -> float: # mu2 = mu0 * S2 / (Ks2 + S2 + S2^2/
12     Ki2) linea 41 model_AM2.m
13     return (p["mu0"] * S2) / (p["Ks2"] + S2 + (S2**2 / p["Ki2"])) + 1e-15)
14
15 def _dmu1_dS1(S1: float, p: dict) -> float:
16     '''
17     derivada de mu1 respecto a S1.
18     mu1 = mu_max1 * S1 / (ks1 + S1)
19     '''
20     return (p["mu_max1"] * p["Ks1"]) / (p["Ks1"] + S1 + 1e-15)**2
21
22 def _dmu2_dS2(S2: float, p: dict) -> float:
23     '''
24     derivada de mu2 respecto a S2.
```

```

23         mu2 = mu_max2 * S2 / (ks2 + S2 + (S2^2/Ki2))
24     Nota: En matlab se uso mu_max2 = mu0 , que es el nuevo valor calculado.
25     '''
26     Ki2 = p["Ki2"]
27     Ks2 = p["Ks2"]
28     mu0 = p["mu0"]
29
30     num = mu0 * Ki2 * (Ki2 * Ks2 - S2**2)
31     denom = (S2**2 + Ki2 * S2 + Ki2 * Ks2)**2
32     return num / denom
33
34 # -----
35 # RHS continuo f(x, u) - 4 estados
36 # -----
37
38 def _f_rhs(x: np.ndarray, u: float, p: dict) -> np.ndarray: # Linea 51-61
39     model_AM2.m
40     """
41     ODEs: Balance de masas de los estados.
42     """
43     X1, X2, S1, S2 = x
44     m1 = _mu1(S1, p)
45     m2 = _mu2(S2, p)
46     # Derivadas de los estados respecto a dt
47     dX1 = (m1 * X1) - (p["alpha"] * u * X1)
48     dX2 = (m2 * X2) - (p["alpha"] * u * X2)
49     dS1 = (u * (5.2845 - S1)) - (p["k1"] * m1 * X1)
50     dS2 = (u * (73.68 - S2)) + (p["k2"] * m1 * X1) - (p["k3"] * m2 * X2)
51     return np.array([dX1, dX2, dS1, dS2]) # dxNormal-dt = [ dX1dt ; dX2dt ;
52     dS1dt ; dS2dt];
53
54 # -----
55 # Modelo discreto extendido fa(x_ext, u)
56 # -----
57
58 def fa(x_ext: np.ndarray, u: float,
59       p: dict = AM2PARAMS, dt: float = DT) -> np.ndarray:
60     """
61     Modelo discreto extendido del AM2 para el EKF.
62     x_ext = [X1, X2, S1, S2, d]
63     Retorna x_ext_siguiente.
64     """
65     x = x_ext[:4] # estados normales X1 X2 S1 S2
66     d = x_ext[4] # estado extendido d
67     X1, X2, S1, S2 = x
68
69     # Euler: ff = x + dt * f(x, u)
70     f = _f_rhs(x, u, p)

```

```

69     ff = x + (dt * f) # linea 68 model_AM2.m
70
71     # Jacobiano dff/du = dt * dff/du
72     dff_du = dt * np.array([
73         -p["alpha"] * X1,
74         -p["alpha"] * X2,
75         5.2845 - S1,
76         73.68 - S2,
77     ])
78
79     # Estado extendido siguiente
80     ff_ext = np.concatenate([ff + dff_du * d, [d]]) #linea 74 model_AM2.m: fa
81     = [ff + jacobian(ff,u)*d ; d];
82     return ff_ext
83
84 # -----
85 # Funcion de salida ga(x_ext, u)
86 # -----
87 def ga(x_ext: np.ndarray, u: float,
88     p: dict = AM2PARAMS) -> float:
89     """ Flujo de CH4:
90         qM = k6 * mu2 * X2 [mmol CH4/L/d].
91     """
92     X2 = x_ext[1]
93     S2 = x_ext[3]
94     m2 = _mu2(S2, p)
95     return p["k6"] * m2 * X2
96
97 # -----
98 # Jacobiano At(x_ext, u) dfa/dx_ext (5x5)
99 # -----
100
101 def At(x_ext: np.ndarray, u: float,
102     p: dict = AM2PARAMS, dt: float = DT) -> np.ndarray:
103     """Jacobiano del modelo extendido discreto x_ext.
104         Derivado analiticamente de:
105         fa = [ff + dff_du*d ; d]
106         ff = x + dt*f(x,u)
107     """
108     x = x_ext[:4]
109     d = x_ext[4]
110     X1, X2, S1, S2 = x
111
112     m1 = _mu1(S1, p)
113     m2 = _mu2(S2, p)
114     dm1_dS1 = _dmu1_dS1(S1, p)
115     dm2_dS2 = _dmu2_dS2(S2, p)

```

```

116
117 # ——— Jacobiano Jf = df/dx (4x4)
118 Jf = np.zeros((4, 4)) # arreglo 4x4
119 # df1/dx -> f1 = mu1*X1 - alpha*u*X1
120 Jf[0, 0] = m1 - p["alpha"] * u # df1/dX1
121 Jf[0, 2] = dm1_dS1 * X1 # df1/dS1
122 # df2/dx -> f2 = mu2*X2 - alpha*u*X2
123 Jf[1, 1] = m2 - p["alpha"] * u # df2/dX2
124 Jf[1, 3] = dm2_dS2 * X2 # df2/dS2
125 # df3/dx -> f3 = u*(S1in - S1) - k1*mu1*X1
126 Jf[2, 0] = -p["k1"] * m1 # df3/dX1
127 Jf[2, 2] = -u - p["k1"] * dm1_dS1 * X1 # df3/dS1
128 # df4/dx -> f4 = u*(S2in - S2) + k2*mu1*X1 - k3*mu2*X2
129 Jf[3, 0] = p["k2"] * m1 # df4/dX1
130 Jf[3, 1] = -p["k3"] * m2 # df4/dX2
131 Jf[3, 2] = p["k2"] * dm1_dS1 * X1 # df4/dS1
132 Jf[3, 3] = -u - p["k3"] * dm2_dS2 * X2 # df4/dS2
133
134 # dff/dx = I + dt*Jf
135 dff_dx = np.eye(4) + dt * Jf # (4x4)
136
137 # ——— Jacobiano de dff_du (4x4)
138 d_dff_du_dx = np.diag([
139     dt * (-p["alpha"]), # (X1)
140     dt * (-p["alpha"]), # (X2)
141     dt * (-1.0), # (S1)
142     dt * (-1.0), # (S2)
143 ]) # (4x4) diagonal
144
145 # d(ff + dff_du*d)/dx = dff_dx + d * d_dff_du_dx
146 block_xx = dff_dx + d * d_dff_du_dx # (4x4)
147
148 # dff_du (4x1) = d(ff + dff_du*d)/dd
149 dff_du = dt * np.array([
150     -p["alpha"] * X1,
151     -p["alpha"] * X2,
152     5.2845 - S1,
153     73.68 - S2,
154 ])
155
156 # ——— Ensamblar At (5x5)
157 A = np.zeros((5, 5))
158 A[:4, :4] = block_xx # dfa[0:4]/dx[0:4]
159 A[:4, 4] = dff_du # dfa[0:4]/dd
160 A[4, 4] = 1.0 # dd_next/dd = 1 (d es estado constante)
161
162 return A
163

```

```

164 # -----
165 # Jacobiano Ct(x_ext, u) dga/dx_ext (1x5)
166 # -----
167
168 def Ct(x_ext: np.ndarray, u: float,
169       p: dict = AM2PARAMS) -> np.ndarray:
170     """
171     Jacobiano de la salida.
172     ga = k6 * mu2(S2) * X2
173     """
174     X2 = x_ext[1]
175     S2 = x_ext[3]
176     m2 = _mu2(S2, p)
177     dm2_dS2 = _dmu2_dS2(S2, p)
178
179     C = np.zeros((1, 5))
180     # dga/dX1 = dga/dS1 = dga/dd = 0
181     C[0, 1] = p["k6"] * m2 # dga/dX2
182     C[0, 3] = p["k6"] * dm2_dS2 * X2 # dga/dS2
183
184     return C

```

Listing A.1: Fragmento código am2_model.py.

A.2. Controlador NMPC.

```

1 import casadi as ca
2 import numpy as np
3 from config import (AM2PARAMS, DT, N_HO, N_HI,
4                     D_MIN, D_MAX, DELTA_D_MIN, DELTA_D_MAX,
5                     X_STATE_MIN, X_STATE_MAX,
6                     W_Y, W_DU)
7
8 class NMPC:
9     """
10     Controlador NMPC con formulacion delta sobre el modelo AM2.
11
12     Uso:
13         nmpc = NMPC()
14         D_new, du_opt, info = nmpc.solve(x_meas, u_prev, d_est, ysp_profile)
15     """
16
17     def __init__(
18         self,
19         p: dict = AM2PARAMS,
20         dt: float = DT,
21         nho: int = N_HO,

```

```

22     nhi: int    = N_HI,
23 ):
24     self._p    = p
25     self._dt   = dt
26     self._nho  = nho
27     self._nhi  = nhi
28
29     # Warm-start: guarda Du optimo del paso anterior
30     self._du_warm = np.zeros(nhi)
31
32     self._solver, self._lbg, self._ubg = self._build_solver()
33
34     # -----
35     # Modelo AM2 continuo para CasADi (igual a fun_AD_model.m)
36     # -----
37
38     def _rhs(self, x: ca.MX, D: ca.MX) -> ca.MX:
39         """
40         RHS continuo AM2.
41
42         Factor 0.96 = (1-0.04) -> linea 83 fun_AD_model.m
43         D ya es el D efectivo (D_cmd + d_est), igual a:
44         D = modeldata.ucontrol(1) + modeldata.uint -> linea 62
45         fun_AD_model.m
46         """
47         p = self._p
48         X1, X2, S1, S2 = x[0], x[1], x[2], x[3]
49
50         mu1 = p["mu_max1"] * S1 / (p["Ks1"] + S1 + 1e-15)
51         mu2 = p["mu0"] * S2 / (p["Ks2"] + S2 + (S2**2 / p["Ki2"]) + 1e-15)
52
53         dX1 = mu1 * X1 * 0.96 - p["alpha"] * D * X1 #Linea 81 fun_AD_model.m
54         dX2 = mu2 * X2 * 0.96 - p["alpha"] * D * X2 #Linea 82
55         dS1 = D * (5.2845 - S1) - p["k1"] * mu1 * X1 #Linea 83
56         dS2 = D * (73.68 - S2) + p["k2"] * mu1 * X1 - p["k3"] * mu2 * X2
57         return ca.vertcat(dX1, dX2, dS1, dS2)
58
59     def _rk4(self, x: ca.MX, D: ca.MX) -> ca.MX:
60         """
61         En Matlab esto se hace con ode45. Lineas 41-71 fun_NMPC_costf.m
62         Aca se hace runge kutta 4 pero dividido espacialmente en 4 subpasos
63         """
64         M = 4 # Numero de sub-pasos
65         h = self._dt / M
66         x_k = x
67         for _ in range(M):
68             k1 = self._rhs(x_k, D)
69             k2 = self._rhs(x_k + h / 2 * k1, D)

```

```

69         k3 = self._rhs(x_k + h / 2 * k2, D)
70         k4 = self._rhs(x_k + h * k3, D)
71         x_k = x_k + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4)
72     return x_k
73
74
75
76 def _output_gcod(self, x: ca.MX) -> ca.MX:
77     """
78     Salida: flujo CH4 en gCOD/L/d.
79
80     lineas 89-97 fun_NMPC_costf.m:
81         ypred = k6 * mu0 * S2/(Ks2+S2+S2^2/Ki2) * X2 * (64/1000)
82     """
83     p = self._p
84     X2 = x[1]
85     S2 = x[3]
86     mu2 = (p["mu0"] * S2) / (p["Ks2"] + S2 + (S2**2 / p["Ki2"])) + 1e-15)
87     return p["k6"] * mu2 * X2 * (64.0 / 1000.0)
88
89 # -----
90 # Construccion del NLP con CasADi
91 # -----
92
93 def _build_solver(self):
94     nho, nhi = self._nho, self._nhi
95
96     # Variables de decision: Du = [Du_0, ..., Du_{nhi-1}]
97     du = ca.MX.sym("du", nhi)
98
99     # Parametros del NLP:
100     # p_vec = [x_meas(4), u_prev(1), d_est(1), ysp(nho)]
101     n_params = 4 + 1 + 1 + nho
102     p_vec = ca.MX.sym("p", n_params)
103
104     x0_p = p_vec[:4]
105     u_prev_p = p_vec[4]
106     d_est_p = p_vec[5]
107     ysp_p = p_vec[6: 6 + nho]
108
109     # Construccion simbolica del costo y restricciones
110     cost = ca.MX(0)
111     g = []
112     lb_g = []
113     ub_g = []
114
115     x = x0_p
116     u = u_prev_p

```

```

117
118     for k in range(nho):
119         # Actualizar u solo dentro del horizonte de control
120         if k < nhi:
121             u = u + du[k]
122             #Restriccion sobre control abs: D_MIN <= u_k <= D_MAX
123             g.append(u)
124             lbг.append(D_MIN)
125             ubг.append(D_MAX)
126             # costo tasa de cambio
127             cost = cost + WDU * du[k] ** 2 # Linea 161 fun_NMPC_costf.m
128
129         D_eff = u + d_est_p
130
131         # Integrar un paso con RK4
132         x = self._rk4(x, D_eff)
133
134         # — Restricciones de estados: xmin <= x_pred <= xmax —
135         # fun_NMPC_nonlcon.m
136         for i in range(4):
137             g.append(x[i])
138             lbг.append(X.STATE_MIN)
139             ubг.append(X.STATE_MAX)
140
141         # Salida predicha [gCOD/L/d]
142         y_k = self._output_gcod(x)
143
144         # — Costo de seguimiento del setpoint —
145         cost = cost + WY * (ysp_p[k] - y_k) ** 2 # Lineas 100,102,161
146         fun_NMPC_costf.m
147
148     # NLP
149     nlp = {
150         "x": du,
151         "f": cost,
152         "g": ca.vertcat(*g),
153         "p": p_vec,
154     }
155
156     opts = {
157         "ipopt.print_level" : 0,
158         "ipopt.max_iter" : 1000,
159         "ipopt.tol" : 1e-5,
160         "ipopt.acceptable_tol" : 1e-4,
161         "ipopt.warm_start_init_point" : "yes",
162         "print_time" : 0,
163         "ipopt.sb" : "yes",

```

```

164     }
165
166     solver = ca.nlpsol("nmpc_solver", "ipopt", nlp, opts)
167     return solver, np.array(lbg, dtype=float), np.array(ubg, dtype=float)
168
169     # -----
170     # Resolver el NMPC en cada paso
171     # -----
172
173     def solve(
174         self,
175         x_meas: np.ndarray,
176         u_prev: float,
177         d_est: float,
178         ysp: list | np.ndarray,
179     ) -> tuple[float, np.ndarray, dict]:
180         """
181         Resuelve el NMPC para el paso actual.
182
183         Parametros
184         -----
185         x_meas : estado estimado [X1,X2,S1,S2] shape (4,)
186         u_prev : control previo D [d^-1]
187         d_est : perturbacion estimada del EKF (5o estado)
188         ysp : setpoints en el horizonte predictivo shape (nho,)
189
190         Retorna
191         -----
192         D_new : nuevo control D a enviar a la planta
193         du_opt : secuencia Du optima shape (nhi,)
194         info : dict con costo, status del solver, secuencia u
195         """
196         nho, nhi = self._nho, self._nhi
197
198         # Vector de parametros
199         p_val = np.concatenate([
200             x_meas.flatten(),
201             [u_prev],
202             [d_est],
203             np.asarray(ysp).flatten()[:nho],
204         ])
205
206         # Warm-start: desplazar Du del paso anterior
207         du0 = np.concatenate([self._du_warm[1:], [0.0]])
208
209         # Box constraints en Du (tasa de cambio del control)
210         lbdu = np.full(nhi, DELTA_D_MIN)
211         ubdu = np.full(nhi, DELTA_D_MAX)

```

```

212
213         try:
214             sol = self._solver(
215                 x0 = du0,
216                 lbx = lbdu,
217                 ubx = ubdu,
218                 lbg = self._lbg,
219                 ubg = self._ubg,
220                 p = p_val,
221             )
222
223             du_opt = np.array(sol["x"]).flatten()
224             cost = float(sol["f"])
225             stats = self._solver.stats()
226             status = stats["return_status"]
227             success = status in ("Solve_Succeeded", "
Solved_To_Acceptable_Level")
228
229         except Exception as exc:
230             print(f"[NMPC] ERROR solver: {exc}")
231             du_opt = np.zeros(nhi)
232             cost = np.nan
233             status = "solver_exception"
234             success = False
235
236         # Guardar para warm-start siguiente paso
237         self._du_warm = du_opt.copy()
238
239         # Reconstruir secuencia de control absoluta
240         u_seq = np.zeros(nhi)
241         u_k = u_prev
242         for k in range(nhi):
243             u_k = np.clip(u_k + du_opt[k], D_MIN, D_MAX)
244             u_seq[k] = u_k
245
246         D_new = float(u_seq[0])
247
248         info = {
249             "cost" : cost,
250             "status" : status,
251             "success": success,
252             "du_opt" : du_opt,
253             "u_seq" : u_seq,
254         }
255
256         return D_new, du_opt, info

```

Listing A.2: Implementación del controlador NMPC con formulación delta (nmpc.py)

A.3. Extended Kalman Filter.

```

1 import numpy as np
2 from am2_model import fa, ga, At, Ct
3 from config import AM2_PARAMS, DT, EKF_X0, EKF_P0_DIAG, EKF_Q_DIAG, EKF_R
4
5
6 class EKF:
7     """
8     Filtro de Kalman Extendido sobre el modelo AM2 extendido (5 estados).
9
10    Uso:
11        ekf = EKF()
12        for each step k:
13            x_post, P_post = ekf.step(y_meas_mmol, u_current)
14            x_nmpc = x_post[:4] # estados para NMPC
15            d_est = x_post[4] # perturbacion estimada
16    """
17
18    def __init__(
19        self,
20        x0: list | np.ndarray | None = None,
21        P0: np.ndarray | None = None,
22        Q: np.ndarray | None = None,
23        R: float | None = None,
24        p: dict = AM2_PARAMS,
25        dt: float = DT,
26    ):
27        # Prior inicial
28        self._x_prior = np.array(x0 if x0 is not None else EKF_X0, dtype=
float)
29        self._P_prior = (
30            P0.copy() if P0 is not None
31            else np.diag(EKF_P0_DIAG).astype(float)
32        )
33
34        # Matrices de ruido
35        self._Q = (
36            Q.copy() if Q is not None
37            else np.diag(EKF_Q_DIAG).astype(float)
38        )
39        self._R = np.array([[R if R is not None else EKF_R]], dtype=float)
40
41        self._p = p
42        self._dt = dt
43
44        # Primer paso: no hay medicion anterior -> saltar correccion
45        self._first_step = True

```

```

46
47     # Historial (opcional, para diagnostico)
48     self.history: list[dict] = []
49
50     # -----
51     # API publica
52     # -----
53
54     def step(self, y_meas: float, u: float) -> tuple[np.ndarray, np.ndarray]:
55         """
56         Un ciclo completo del EKF.
57
58         simulate_ADM1.NMPC-AM2.m lineas 480-500:
59             1. Measurement update -> x_post, P_post
60             2. Time update         -> x_prior, P_prior (usa x_post sin clamp)
61             3. No-negatividad      -> clamp sobre x_post DESPUES del time update
62
63         Parametros
64         -----
65         y_meas : medicion del flujo CH4 [mmol/L/d]
66         u       : dilution rate D aplicado en este paso [d^-1]
67
68         Retorna
69         -----
70         x_post : estado posterior      shape (5,)
71         P_post : covarianza posterior          shape (5,5)
72         """
73         if self._first_step:
74             # En el primer paso solo prediccion (no hay medicion pasada)
75             x_post = self._x_prior.copy()
76             P_post = self._P_prior.copy()
77             self._first_step = False
78         else:
79             x_post, P_post = self._measurement_update(y_meas, u)
80
81         # Time update ANTES del clamp - lineas 492-493
82         # fa y At se evaluan con x_post sin clampar
83         self._time_update(x_post, P_post, u)
84
85         # No-negatividad DESPUES del time update - lineas 496-500
86         # Solo estados fisicos [X1, X2, S1, S2]; d se deja libre
87         for i in range(4):
88             x_post[i] = max(0.0, x_post[i])
89
90         # Registrar
91         self.history.append({
92             "x_post" : x_post.copy(),
93             "P_post" : P_post.copy(),

```

```

94         "y_meas" : y_meas,
95         "u"      : u,
96     })
97
98     return x_post, P_post
99
100 @property
101 def x_prior(self) -> np.ndarray:
102     """Acceso al estado prior actual (antes de la correccion)."""
103     return self._x_prior.copy()
104
105 @property
106 def P_prior(self) -> np.ndarray:
107     return self._P_prior.copy()
108
109 # -----
110 # funciones Internass
111 # -----
112
113 def _measurement_update(
114     self, y_meas: float, u: float
115 ) -> tuple[np.ndarray, np.ndarray]:
116     """
117     Correccion (measurement update).
118     simulate_ADM1_NMPC_AM2.m lineas 480-483.
119     """
120     x_pr = self._x_prior # estimacion actualizada del estado
121     P_pr = self._P_prior #estimacion actualizada covarianza
122
123     C = Ct(x_pr, u, self._p)
124
125     S_innov = self._R + C @ P_pr @ C.T
126     K       = P_pr @ C.T @ np.linalg.inv(S_innov) # ganancia kalman
127
128     y_hat   = ga(x_pr, u, self._p)
129     innov   = y_meas - y_hat
130     x_post  = x_pr + K.flatten() * innov
131
132     P_post  = (np.eye(5) - K @ C) @ P_pr
133     #P_post = 0.5 * (P_post + P_post.T) # simetrizar(estabilidad
134     #numERICA)
135     return x_post, P_post
136
137 def _time_update(
138     self, x_post: np.ndarray, P_post: np.ndarray, u: float
139 ) -> None:
140     """ Prediccion """
141     A = At(x_post, u, self._p, self._dt)

```

```
141     self._x_prior = fa(x_post, u, self._p, self._dt)
142     self._P_prior = self._Q + A @ P_post @ A.T
143     self._P_prior = 0.5 * (self._P_prior + self._P_prior.T)
```

Listing A.3: Implementación EKF, archivo ekf.py

CÓDIGOS PARA PRUEBA SIMPLE.

```
1 %% Configuracion
2 IP    = "192.168.1.87"; % ip server
3 PORT  = 502;
4 SCALE = 100; %factor de escalamiento
5 m = modbus('tcpip', IP, PORT);
6 % Direcciones MATLAB = registro Modbus + 1
7 REG_Y    = 1; % Reg_0 (y)
8 REG_U    = 2; % Reg_1 (u)
9 REG_SP   = 3; % Reg_2 (setpoint)
10 REG_READY = 4; % Reg_3 (pi_ready)
11
12 %% G(s) = 6 / ( (s+1) (s+2) (s+3) )
13 dt = 0.1; %paso virtual de simulacion
14 Ac = [ 0  1  0;
15        0  0  1;
16       -6 -11 -6];
17 Bc = [0; 0; 1];
18 Cc = [6 0 0];
19 Dc = 0;
20 sysC = ss(Ac, Bc, Cc, Dc);
21 sysD = c2d(sysC, dt, 'zoh'); % discretizar sistema
22 [Ad, Bd, Cd, ~] = ssdata(sysD);
23 nx = size(Ad,1);
24 x = zeros(nx,1);
```

Listing B.1: Codigo Simple Matlab, configuracion y sistema.

```

1 import casadi as ca
2 import numpy as np
3 from scipy.signal import cont2discrete, place_poles
4 from pyModbusTCP.client import ModbusClient
5 import time
6
7 # =====
8 # Protocolo:
9 #   Reg_0 = y           (escribe MATLAB)
10 #   Reg_1 = u           (escribe Python)
11 #   Reg_2 = setpoint    (escribe MATLAB)
12 #   Reg_3 = pi_ready    (escribe Python, lee MATLAB)
13 #
14 #   pi_ready = 1 : MATLAB puede avanzar un paso
15 #   pi_ready = 0 : MATLAB ya avanzo, esperando el proximo u
16 # Como cada paso de la planta ocurre exactamente cuando el u
17 # correspondiente ya fue escrito.
18 # =====
19 NODEJS_IP = "192.168.1.87" # ip servidor 192.168.1.85
20 NODEJS_PORT = 502 ; SCALE = 100
21 REG_Y = 0 ; REG_U = 1 ; REG_SP = 2 ; REG_READY = 3
22
23 c = ModbusClient(host=NODEJS_IP, port=NODEJS_PORT, auto_open=True)
24
25 # Modelo:  $G(s) = 6/((s+1)(s+2)(s+3))$ 
26 dt = 0.1
27 Ac = np.array([[ 0, 1, 0],
28               [ 0, 0, 1],
29               [-6, -11, -6]], dtype=float)
30
31 Bc = np.array([[0.], [0.], [1.]])
32 Cc = np.array([[6., 0., 0.]])
33
34 sys_d = cont2discrete((Ac, Bc, Cc, [[0.]]), dt, method='zoh')
35 Ad = sys_d[0]; Bd = sys_d[1] ; Cd = sys_d[2]
36 nx = Ad.shape[0]
37
38 print(f"Planta:  $G(s) = 6/((s+1)(s+2)(s+3))$  | dt (virtual): {dt}s")
39 # ===== Observador =====
40 n_aug = nx + 1
41
42 A_aug = np.zeros((n_aug, n_aug))
43 A_aug[:nx, :nx] = Ad
44 A_aug[nx, nx] = 1.0
45
46 B_aug = np.zeros((n_aug, 1))
47 B_aug[:nx, 0] = Bd.flatten()
48

```

```

49 C_aug = np.zeros((1, n_aug))
50 C_aug[0, :nx] = Cd.flatten()
51 C_aug[0, nx] = 1.0
52
53 polos_c_aug = np.array([-3.0+1.0j, -3.0-1.0j, -4.5, -2.0])
54 polos_d_aug = np.exp(polos_c_aug * dt)
55
56 result_aug = place_poles(A_aug.T, C_aug.T, polos_d_aug)
57 L_aug = result_aug.gain.matrix.T
58
59 print(f"Polos observador aumentado: {np.round(polos_d_aug, 3)}")
60 print(f"L_aug = {np.round(L_aug.flatten(), 4)}\n")
61 #===== MPC =====
62 nz = nx
63 Az = Ad ; Bz = Bd ; Cz = Cd
64 N = 15 ; Q = 80.0 ; R = 5.0 ; Q_T = Q * 15
65 U_MAX = 6.0 ; U_MIN = -6.0
66
67 z_sym = ca.MX.sym('z', nz)
68 sp_sym = ca.MX.sym('sp')
69 d_sym = ca.MX.sym('d')
70 U_sym = ca.MX.sym('U', N)
71 Az_ca = ca.DM(Az) ; Bz_ca = ca.DM(Bz) ; Cz_ca = ca.DM(Cz)
72
73 costo = 0
74 z_k = z_sym
75 g_constraints = [] ; lb_g = [] ; ub_g = []
76
77 for k in range(N):
78     y_k = Cz_ca @ z_k + d_sym
79     e_k = sp_sym - y_k
80
81     costo += Q * e_k**2
82     costo += R * U_sym[k]**2
83
84     g_constraints.append(U_sym[k])
85     lb_g.append(U_MIN)
86     ub_g.append(U_MAX)
87
88     z_k = Az_ca @ z_k + Bz_ca @ U_sym[k]
89
90 y_N = Cz_ca @ z_k + d_sym
91 costo += Q_T * (sp_sym - y_N)**2
92
93 params = ca.vertcat(z_sym, sp_sym, d_sym)
94 nlp = {... # omitido por brevedad
95 opts = {... # omitido por brevedad
96

```

```

97 solver = ca.nlpsol('mpc', 'ipopt', nlp, opts)
98
99 # Modbus
100 def to_uint16(val: float) -> int:
101     v = int(val * SCALE)
102     v = max(-32768, min(32767, v))
103     return v + 65536 if v < 0 else v
104
105 def from_uint16(raw: int) -> float:
106     if raw > 32767:
107         raw -= 65536
108     return raw / SCALE
109
110 def wait_flag(target: int, timeout: float = 5.0, poll_dt: float = 0.005) ->
    bool:
111     """Espera hasta que Reg_READY == target. Devuelve False si hubo timeout."""
112     t0 = time.time()
113     while True:
114         regs = c.read_holding_registers(REG_READY, 1)
115         if regs is not None and regs[0] == target:
116             return True
117         if time.time() - t0 > timeout:
118             return False
119         time.sleep(poll_dt)
120
121 # Estado inicial
122 xi_est = np.zeros((n_aug, 1))
123 u_prev = 0.0 # ultimo u enviado
124 U_warm = np.zeros(N)
125 y_hist = [] ; paso = 0
126
127 print(f"Conectando a {NODEJS_IP}:{NODEJS_PORT}.")
128 try:
129     print("Esperando conexion con servidor.")
130     while True:
131         regs = c.read_holding_registers(0, 1)
132         if regs is not None:
133             print("Conexion confirmada, iniciando MPC.\n")
134             break
135         time.sleep(0.1)
136
137     while True:
138         t_start = time.time()
139         # 1. Esperar a Matlab
140         if not wait_flag(0, timeout=5.0):
141             print("Timeout esperando pi_ready==0.")
142             continue

```

```

143
144 # 2. Leer medicion y setpoint
145 regs = c.read_holding_registers(REG.Y, 3) # Reg_0, Reg_1, Reg_2
146 if regs is None:
147     print("Sin respuesta al leer medicion.")
148     continue
149
150 y = from_uint16(regs[0])
151
152 if USAR_SETPOINT_FIJO:
153     setpoint = SETPOINT_FIJO
154 else:
155     setpoint = from_uint16(regs[2])
156
157 y_hist.append(y)
158 if len(y_hist) > 10:
159     y_hist.pop(0)
160     if max(y_hist) - min(y_hist) < 0.005:
161         print("Senal muerta.")
162
163 # 3. Observador
164 xi_pred = A_aug @ xi_est + B_aug * u_prev
165 y_pred = float((C_aug @ xi_pred).flat[0])
166 innov = y - y_pred
167 xi_est = xi_pred + L_aug * innov
168 x_est = xi_est[:nx]
169 d_hat = float(xi_est[nx])
170
171 # 4. Resolver mpc
172 p_val = np.concatenate([x_est.flatten(), [setpoint, d_hat]])
173 sol = solver(
174     x0 = U_warm,
175     lbx = [U_MIN] * N,
176     ubx = [U_MAX] * N,
177     lbg = lbg,
178     ubg = ubg,
179     p = p_val
180 )
181 U_opt = np.array(sol['x']).flatten()
182 u = float(np.clip(U_opt[0], U_MIN, U_MAX))
183 U_warm = np.append(U_opt[1:], U_opt[-1])
184 # 5. Enviar control y avisar a matlab
185 c.write_single_register(REG.U, to_uint16(u))
186 c.write_single_register(REG.READY, 1) # avisa a MATLAB para avanzar
187
188 u_prev = u
189 paso += 1
190 t_ms = (time.time() - t_start) * 1000

```

```
191         print(f"[{paso:4d}] sp={setpoint:.2f} | y={y:.4f} | e={setpoint-y:+.4f} | ")
192         f"d={d_hat:+.4f} | u={u:.4f} | t_ciclo={t_ms:.1f}ms")
193
194
195 except KeyboardInterrupt:
196     print("\nMPC detenido.")
197     c.write_single_register(REG.U, 0)
198     c.write_single_register(REG.READY, 0)
```

Listing B.2: Codigo Simple MPC.

Índice de figuras

3.1. Seguimiento de referencia de la producción de metano obtenido por la estrategia EKF-NMPC en los resultados previos.	24
3.2. Seguimiento de referencia de la producción de metano obtenido por la estrategia EKF-NMPC, señal real en los resultados previos.	25
3.3. Seguimiento de referencia de la producción de metano obtenido por la estrategia EKF-NMPC, señal con ruido de medición en los resultados previos.	25
3.4. Tasa de dilución aplicada y perturbación estimada en los resultados previos.	26
3.5. Error de seguimiento respecto de la referencia en los resultados previos.	26
3.6. Estados en los resultados previos.	27
4.1. Esquema inicial de la arquitectura de comunicación propuesta.	29
4.2. Arquitectura final de comunicación, incorporando el servidor Modbus TCP.	29
5.1. Respuesta de la planta simple ante una referencia escalonada.	44
5.2. Seguimiento de referencia de la producción de metano para el problema de interés.	45
5.3. Error de seguimiento de la producción de metano.	47
5.4. Tasa de dilución calculada por el controlador NMPC.	48
5.5. Estimaciones de biomasa X_1 y X_2 obtenidas por el EKF.	49
5.6. Estimación del estado S_2 obtenida por el EKF.	49
5.7. Estimación del disturbio de entrada utilizada para compensación libre de offset.	51

Índice de tablas

3.1. Parámetros del modelo AM2, adaptado de [21].	22
3.2. Métricas de desempeño reportadas para las estrategias finales en [21].	27
5.1. Error de seguimiento de Y por segmento de setpoint.	46
5.2. Tasa de dilución D al final de cada segmento.	47
5.3. RMSE de la estimación EKF respecto del estado real de la planta, por implementación.	50
5.4. RMSE de la estimación EKF entre la implementación Original y la HIL.	50
5.5. Perturbación estimada $\hat{d}_k$ al final de cada segmento.	51

BIBLIOGRAFÍA

- [1] D. J. Batstone and B. Virdis, “The role of anaerobic digestion in the emerging energy economy,” *Current Opinion in Biotechnology*, vol. 27, pp. 142–149, 2014.
- [2] J. Jimenez, E. Latrille, J. Harmand, A. Robles, J. Ferrer, D. Gaida, C. Wolf, F. Mairet, O. Bernard, V. Alcaraz-Gonzalez, H. Mendez-Acosta, D. Zitomer, D. Totzke, H. Spanjers, F. Jacobi, A. Guwy, R. Dinsdale, G. Premier, S. Mazhegrane, G. Ruiz-Filippi, A. Seco, T. Ribeiro, A. Pauss, and J.-P. Steyer, “Instrumentation and control of anaerobic digestion processes: A review and some research challenges,” *Reviews in Environmental Science and Bio/Technology*, vol. 14, no. 4, pp. 615–648, 2015.
- [3] D. Nguyen, V. Gadhamshetty, S. Nitayavardhana, and S. K. Khanal, “Automatic process control in anaerobic digestion technology: A critical review,” *Bioresource Technology*, vol. 193, pp. 513–522, 2015.
- [4] D. J. Batstone, J. Keller, I. Angelidaki, S. V. Kalyuzhnyi, S. G. Pavlostathis, A. Rozzi, W. T. M. Sanders, H. Siegrist, and V. A. Vavilin, “The IWA anaerobic digestion model no. 1 (ADM1),” *Water Science and Technology*, vol. 45, no. 10, pp. 65–73, 2002.
- [5] O. Bernard, Z. Hadj-Sadok, D. Dochain, A. Genovesi, and J.-P. Steyer, “Dynamical model development and parameter identification for an anaerobic wastewater treatment process,” *Biotechnology and Bioengineering*, vol. 75, no. 4, pp. 424–438, 2001.
- [6] S. Hassam, E. Ficara, A. Leva, and J. Harmand, “A generic and systematic procedure to derive a simplified model from the anaerobic digestion model no. 1 (ADM1),” *Biochemical Engineering Journal*, vol. 99, pp. 193–203, 2015.
- [7] D. Li, I. Lee, and H. Kim, “Application of the linearized ADM1 (LADM) to lab-scale anaerobic digestion system,” *Journal of Environmental Chemical Engineering*, vol. 9, no. 3, p. 105193, 2021.
- [8] L. Dewasme, M. Sbarciog, E. Rocha-Cózatl, F. Haugen, and A. Vande Wouwer, “State and

- unknown input estimation of an anaerobic digestion reactor with experimental validation,” *Control Engineering Practice*, vol. 85, pp. 280–289, 2019.
- [9] G. Lara-Cisneros, R. Aguilar-López, D. Dochain, and R. Femat, “On-line estimation of VFA concentration in anaerobic digestion via methane outflow rate measurements,” *Computers & Chemical Engineering*, vol. 94, pp. 250–256, 2016.
- [10] M. A. Ghanavati, E. Vafa, and M. Shahrokhi, “Control of an anaerobic bioreactor using a fuzzy supervisory controller,” *Journal of Process Control*, vol. 103, pp. 87–99, 2021.
- [11] A. Chaabna and S. Semcheddine, “Robust methane production from anaerobic digestion of maize silage: Feeding control using sliding mode control strategy with anaerobic digestion model no. 1,” *Bioresource Technology*, vol. 416, p. 131749, 2025.
- [12] D. Q. Mayne, “Model predictive control: Recent developments and future promise,” *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.
- [13] M. M. F. Borrelli, A. Bemporad, *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.
- [14] E. Mauky, S. Weinrich, H.-J. Nägele, H. F. Jacobi, J. Liebetrau, and M. Nelles, “Model predictive control for demand-driven biogas production in full scale,” *Chemical Engineering & Technology*, vol. 39, no. 4, pp. 652–664, 2016.
- [15] H. Kil, D. Li, Y. Xi, and J. Li, “Model predictive control with on-line model identification for anaerobic digestion processes,” *Biochemical Engineering Journal*, vol. 128, pp. 63–75, 2017.
- [16] S. Fawzy, M. Saeed, A. Eladl, and M. El-Saadawi, “Adaptive control system for biogas power plant using model predictive control,” *Journal of Modern Power Systems and Clean Energy*, vol. 9, no. 5, pp. 1193–1202, 2021.
- [17] W. Ahmed and J. Rodríguez, “A model predictive optimal control system for the practical automatic start-up of anaerobic digesters,” *Water Research*, vol. 174, p. 115599, 2020.
- [18] S. García-Gen, L. O. Santos, and A. Vande Wouwer, “Application of a nonlinear model predictive controller to the anaerobic digestion of readily biodegradable substrates,” *IFAC-PapersOnLine*, vol. 55, no. 7, pp. 909–914, 2022.
- [19] E. R. Piceno-Díaz, L. A. Ricardez-Sandoval, M. A. Gutierrez-Limon, H. O. Méndez-Acosta, and H. Puebla, “Robust nonlinear model predictive control for two-stage anaerobic digesters,” *Industrial & Engineering Chemistry Research*, vol. 59, no. 52, pp. 22 559–22 572, 2020.

- [20] R. Kamesh and K. Y. Rani, "Iterative EKF as a controller in novel MPC formulation: First principles model based IEKF-MPC for SISO systems," *Computers & Chemical Engineering*, vol. 163, p. 107833, 2022.
- [21] M. A. Azúa-Poblete, "State estimation and model predictive control of the anaerobic digestion process for the optimization of biogas production," Master's thesis, Universidad Técnica Federico Santa María, 2025.
- [22] M. Azúa-Poblete, A. L. Cedeño, J. C. Agüero, L. O. Santos, L. Dewasme, A. Vande Wouwer, and S. García-Gen, "Enhancing anaerobic digestion performance with offset-free model predictive control," *Journal of Water Process Engineering*, vol. 78, p. 108785, 2025.
- [23] L. Wang, F. Long, W. Liao, and H. Liu, "Prediction of anaerobic digestion performance and identification of critical operational parameters using machine learning algorithms," *Bioresource Technology*, vol. 298, p. 122495, 2020.
- [24] P. Ganeshan, A. Bose, J. Lee, S. Barathi, and K. Rajendran, "Machine learning for high solid anaerobic digestion: Performance prediction and optimization," *Bioresource Technology*, vol. 400, p. 130665, 2024.
- [25] A. Pandey, O. Momeni, and P. Pandey, "Desing and implementation of iot-enabled device for real-time monitoring of greenhouse gas emissions, and pressure in anaerobic reactors," *IEEE Access*, vol. 12, 2024.
- [26] S. Adhau, S. Patil, D. Ingole, and D. Sonawane, "Implementation and analysis of nonlinear model predictive controller on embedded systems for real-time applications," in *2019 18th European Control Conference (ECC)*, 2019, pp. 3359–3364.
- [27] V. Patne, D. Ingole, and D. Sonawane, "Towards fast nonlinear model predictive control for embedded applications," *IFAC-PapersOnLine*, vol. 55, no. 22, pp. 304–309, 2022, 22nd IFAC Symposium on Automatic Control in Aerospace ACA 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405896323003099>
- [28] P. Krupa, J. Cámara, I. Alvarado, D. Limón, and T. Alamo, "Real-time implementation of mpc for tracking in embedded systems: Application to a two-wheeled inverted pendulum," in *European Control Conference (ECC)*, 2021, pp. 669–674. [Online]. Available: <https://ieeexplore.ieee.org/document/9483102>
- [29] D. Ferri Beltrán, "Implementación de algoritmos de control avanzados mpc en pls industriales," Master's thesis, Universitat Politècnica de València, 2019.
- [30] P. Krupa, N. Saraf, D. Limon, and A. Bemporad, "Plc implementation of a real-time embedded mpc algorithm based on linear

- input/output models.” *IFAC-PapersOnLine*, vol. 53, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S240589632030714X>
- [31] T. A. Rajasingham, “Formulation of the optimization problem,” in *Nonlinear Model Predictive Control of Combustion Engines: From Fundamentals to Applications*. Cham: Springer International Publishing, 2021, pp. 139–172.
- [32] Modbus Organization, “MODBUS application protocol specification v1.1b3,” Modbus Organization, Tech. Rep., 2012. [Online]. Available: <https://www.modbus.org/file/secure/modbusprotocolspecification.pdf>
- [33] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, “CasADi – A software framework for nonlinear optimization and optimal control,” *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.
- [34] A. Wächter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.
- [35] B. C. B. Benyahia, T. Sari and J. Harmand, “Bifurcation and stability analysis of a two step model for monitoring anaerobic digestion processes,” *Journal of Water Process Engineering*, vol. 22, 2012. [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S0959152412000972>
- [36] M. Azúa-Poblete, A. L. Cedeño, J. C. Agüero, and L. O. Santos, “Enhancing anaerobic digestion performance with offset-free model predictive control,” *Elsevier Water Process Engineering*, vol. 78, no. 108785, 2025.