

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
SEDE VIÑA DEL MAR – JOSÉ MIGUEL CARRERA

DISEÑO DE UN CONTROLADOR LQR APLICANDO UN SISTEMA DE CONTROL DE NIVEL

Trabajo de Titulación para optar al Título
Profesional de Ingeniero de Ejecución en
CONTROL E INSTRUMENTACIÓN
INDUSTRIAL

Alumno:

Simón Jesús Quiroz Pizarro

Profesor Guía:

Mag. Guelis Montenegro Zamora



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: Diseño de un controlador LQR aplicando un sistema de control de nivel.

Nombre del candidato(a): Simón Jesús Quiroz Pizarro

Carrera / Grado: Ingeniero de Ejecución en Control e Instrumentación Industrial

Campus: Viña del Mar **Departamento:** Electrotecnia e Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Guelis Darío Montenegro Zamora, 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.

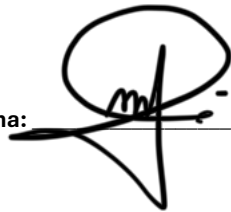
☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

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

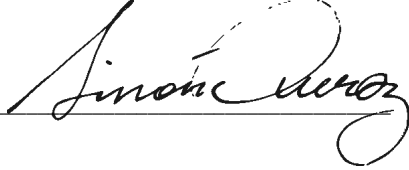
Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

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

Fecha: 26/10/2026 Firma: 

Estudiante o Candidato(a):

Fecha: 26/10/2026 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.

RESUMEN

KEYWORDS: LQR, GAIN SCHEDULING, CONTROL DISCRETO, PYTHON.

El presente trabajo consiste en el diseño e implementación de un sistema de control para un proceso de llenado y vaciado de un estanque, utilizando un controlador LQR junto con una estrategia de Gain Scheduling para adaptarse a distintos puntos de operación. Para llevar a cabo el estudio se empleó el software Factory I/O como plataforma de simulación, el cual permitió recrear una planta virtual y comunicarse en tiempo real con Python mediante el protocolo Modbus TCP/IP.

Durante el desarrollo del proyecto se implementó un sistema de adquisición de datos capaz de registrar el nivel, el flujo y las señales aplicadas a las válvulas del estanque, generando así el conjunto de datos necesario para identificar modelos lineales alrededor de diferentes condiciones de operación. A partir de estos modelos, se calcularon las matrices de ganancia del controlador LQR, lo que permitió construir una estructura de Gain Scheduling para seleccionar automáticamente la ganancia más adecuada según el nivel del estanque.

Posteriormente, se integró toda la lógica de control en lazo cerrado, incluyendo el cálculo de la acción LQR, la estimación de estados, el cambio automático de modelo y la limitación del esfuerzo de control. Finalmente, se realizaron pruebas utilizando una señal PRBS con el fin de evaluar cómo respondía el sistema frente a cambios rápidos y aleatorios en el proceso.

Los resultados obtenidos muestran que el controlador logró mantener la estabilidad del nivel y reaccionar de manera adecuada ante las perturbaciones generadas por la señal PRBS. Además, el Gain Scheduling permitió seleccionar correctamente las ganancias para cada tramo operativo, mejorando la precisión del control y asegurando un funcionamiento más robusto en todo el rango del estanque.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN	1
1. CAPÍTULO 1: PROBLEMAS Y OBJETIVOS	2
1.1 ANTECEDENTES GENERALES	3
1.1.1 Herramientas y entorno de trabajo	3
1.2 PLANTEAMIENTO DEL PROBLEMA	4
1.2.1 Definición del problema	4
1.2.2 Importancia de resolverlo	5
1.2.3 Involucrados	6
1.3 SOLUCIÓN DEL PROBLEMA	6
1.3.1 Requerimientos	7
1.4 OBJETIVO GENERAL	7
1.4.1 Objetivos específicos	7
1.5 CONTROLADOR LQR	8
1.5.1 Representación en variables de estado	8
1.5.2 Función de costo del controlador LQR	8
1.5.3 Ley de control	9
1.5.4 Ecuación algebraica de Riccati	9
1.6 ANTECEDENTES GENERALES	10
1.7 SISTEMA DE SIMULACIÓN Y ADQUISICIÓN DE DATOS	10
2. CAPÍTULO 2	12
2.1 ENTORNO DE PRUEBAS	13
2.1.1 Comunicación entre Python y Factory I/O	13
2.2 RUTINA DE ADQUISICIÓN DE DATOS	16
2.3 IDENTIFICACIÓN DE UN MODELO LPV	20
2.4 ECUACIÓN ALGEBRAICA DE RICCATI	22
2.5 ITERACIÓN DEL EXPERIMENTO	26
3 CAPÍTULO 3	30
3.1 LÓGICA DE CONTROL DISCRETA	31
3.1.1 Base de datos para selección de modelo	31
3.1.2 Controlador LQR	36
3.1.3 Gain Scheduler	38
3.1.4 Control de lazo cerrado	39
3.1.5 Bucle Principal	42

3.2	DEMOSTRACIÓN	43
3.2.1	Experimento 1	44
3.2.2	Experimento 2	46
3.2.3	Experimento 3	49
3.2.4	Experimento 4	51
CONCLUSIONES		54
BIBLIOGRAFÍA		55

ÍNDICE DE FIGURAS

Figura 1-1 Diagrama de Causa-Efecto para identificar el problema.	5
Figura 1-2 Estanque de agua modelado en la plataforma Factory I/O.	11
Figura 1-3 Interfaz de la plataforma mostrando las variables de operación.	11
Figura 2-1 Selección del controlador Modbus TCP/IP Server en Factory I/O.	14
Figura 2-2 Software Factory I/O.	14
Figura 2-3 Asignación de variables del proceso a registros Modbus en Factory I/O.	15
Figura 2-4 Experimento para identificación de sistemas.	21
Figura 2-5 Impacto de Q en la ganancia K .	25
Figura 2-6 Impacto de Q en los polos de lazo cerrado.	25
Figura 2-7 Impacto de Q en la solución de la ecuación de Riccati.	25
Figura 2-8 Impacto de Q en la ganancia K para el Sistema 1.	27
Figura 2-9 Impacto de Q en la ganancia K para el Sistema 2.	27
Figura 2-10 Impacto de Q en la ganancia K para el Sistema 3.	27
Figura 2-11 Impacto de Q en la ganancia K para el Sistema 4.	27
Figura 2-12 Impacto de Q en la ganancia K para el Sistema 5.	28
Figura 2-13 Impacto de Q en la ganancia K para el Sistema 6.	28
Figura 2-14 Impacto de Q en la ganancia K para el Sistema 7.	28
Figura 2-15 Impacto de Q en la ganancia K para el Sistema 8.	28
Figura 2-16 Impacto de Q en la ganancia K para el Sistema 9.	29
Figura 3-1 Experimento 1 de verificación.	45
Figura 3-2 Experimento 1 con PRBS de verificación.	46
Figura 3-3 Experimento 2 de verificación.	47
Figura 3-4 Experimento 2 con PRBS de verificación.	48
Figura 3-5 Experimento 3 de verificación.	49
Figura 3-6 Experimento 3 con PRBS de verificación.	50

Figura 3-7 Experimento 4 de verificación.	52
Figura 3-8 Experimento 4 con PRBS de verificación.	53

ÍNDICE DE TABLAS

Tabla 2-1 Resultados de la primera identificación de sistemas.	22
Tabla 2-2 Modelo de estructura de ganancia para el primer modelo.	24
Tabla 2-3 Modelo de estados para nueve sistemas lineales.	26
Tabla 3-1 Parámetros para Experimento 1.	44
Tabla 3-2 Resultados de Experimento 1.	45
Tabla 3-3 Parámetros para Experimento 1 con PRBS.	45
Tabla 3-4 Resultados de Experimento 1 con PRBS.	46
Tabla 3-5 Parámetros para Experimento 2.	47
Tabla 3-6 Resultados de Experimento 2.	47
Tabla 3-7 Parámetros para Experimento 2 con PRBS.	48
Tabla 3-8 Resultados de Experimento 2 con PRBS.	48
Tabla 3-9 Parámetros para Experimento 3.	49
Tabla 3-10 Resultados de Experimento 3.	50
Tabla 3-11 Parámetros para Experimento 3 con PRBS.	50
Tabla 3-12 Resultados de Experimento 3 con PRBS.	51
Tabla 3-13 Parámetros para Experimento 4.	51
Tabla 3-14 Resultados de Experimento 4.	52
Tabla 3-15 Parámetros para Experimento 4 con PRBS.	52
Tabla 3-16 Resultados de Experimento 4 con PRBS.	53

SIGLAS Y SIMBOLOGÍA

A. SIGLA

PLC	: Programmable Logic Controller.
LQR	: Linear Quadratic Regulator.
TCP/IP	: Transmission Control Protocol / Internet Protocol.
I/O	: Entrada / Salida.
CSV	: Comma Separated Values.
PID	: Proporcional Integral Derivativo
LPV	: Lineal de parámetros variables.
PRBS	: Pseudo-Random Binary Sequence

B. SIMBOLOGÍA

$x(t)$	: Vector de estados del sistema.
$u(t)$	: Vector de entradas o señales de control.
$y(t)$	: Vector de salidas.
$J(u)$	: Índice de desempeño o función costo.
$X(t)$	: Vector de estados del sistema en el tiempo.
A	: Matriz dinámica del sistema.
A^T	: Matriz dinámica del sistema transpuesta.
B	: Matriz de entrada que relaciona la entrada con la actuación.
B^T	: Matriz de entrada que relaciona la entrada con la actuación transpuesta.
C	: Matriz dinámica de salida.
D	: Matriz de transmisión directa.
K	: Matriz óptima de ganancia.
P	: Matriz simétrica que se busca resolver mediante la ecuación de Riccati.
Q	: Matriz de ponderación de estados.
R	: Matriz de ponderación de control.
L	: Solución real de polos de lazo cerrado.

INTRODUCCIÓN

En los entornos industriales modernos, el control automático es un recurso fundamental para garantizar eficiencia, calidad del producto y seguridad. Las plantas de producción y los sistemas de control requieren herramientas que permitan no solo operar de forma autónoma, sino también adquirir y procesar datos en tiempo real.

En la actualidad es posible realizar simulaciones mediante software que permite replicar y analizar sistemas industriales. Plataformas como Factory I/O hacen posible recrear entornos virtuales donde se simulan distintos componentes de un proceso, lo que facilita el estudio y validación de estrategias de control.

Mediante protocolos de comunicación populares, como Modbus TCP/IP, estas plataformas pueden comunicarse mediante programas externos, permitiendo una adquisición y análisis en tiempo real. Esta intervención posibilita implementar controladores de mayor robustez e implementar técnicas de aprendizaje automático para modelar y optimizar comportamientos dinámicos.

El presente trabajo se enmarca en el desarrollo de un sistema de adquisición de datos y control inteligente aplicado a un estanque de nivel, orientado principalmente a la evaluación dinámica del comportamiento del sistema ante señales aleatorias y perturbaciones controladas. El sistema propuesto emplea un controlador LQR (Linear Quadratic Regulator) en junto a un modelo de adquisición de datos para ser procesado por algoritmos de Matlab con el propósito de determinar una matriz de ganancia óptima.

CAPÍTULO 1: PROBLEMAS Y OBJETIVOS

1 **PROBLEMAS Y OBJETIVOS**

1.1 **ANTECEDENTES GENERALES**

En el contexto industrial, el llenado y vaciado de estanques es uno de los procesos más comunes dentro de la automatización de plantas industriales (alimenticia, química, farmacéutica, entre otras), en donde un control adecuado asegura la continuidad del proceso y la seguridad para los trabajadores del área.

Las industrias modernas buscan desarrollar sistemas que no solo automaticen operaciones, sino que además sean capaces de adquirir y registrar datos de proceso para su posterior análisis.

Para contextos de pruebas e investigación el uso de plataformas de simulaciones industriales, como Factory I/O, permite crear entornos virtuales en donde es posible recrear una planta con sistemas no lineales para el desarrollo de estrategias de control. Combinado con una comunicación bajo el protocolo ModBus TCP/IP es posible incorporar sistemas externos de control programado y adquisición de datos.

La idea central de este proyecto consiste en desarrollar un sistema que, mediante una rutina de adquisición y control de datos, sea capaz de determinar una función discreta de control basada en un conjunto de modelos lineales identificados en distintos puntos de operación. A partir de estos modelos, se implementa un esquema de Gain Scheduling que permite seleccionar la matriz de ganancia óptima del controlador LQR según el nivel deseado del estanque, asegurando un desempeño adecuado bajo diferentes condiciones del proceso.

1.1.1 **Herramientas y entorno de trabajo**

- Python: Lenguaje de programación utilizado para desarrollar las rutinas de control, la comunicación mediante el protocolo Modbus TCP/IP y la adquisición de datos. Se selecciona por su capacidad de poder trabajar con matrices y grandes bases de datos.
- Factory I/O: Entorno de simulación industrial que emula un sistema de llenado y vaciado de estanques. Permite generar condiciones realistas de proceso,

someter a la planta a perturbaciones y comunicar variables en tiempo real con Python mediante Modbus TCP/IP.

1.2 PLANTEAMIENTO DEL PROBLEMA

Los controladores clásicos del tipo PID son comúnmente utilizados en la industria por su simplicidad y efectividad en condiciones de operación favorables. Sin embargo, presentan dificultades para mantener un desempeño adecuado bajo diferentes puntos de operación.

Frente a estas limitaciones surge la necesidad de diseñar un sistema de control capaz de ajustar automáticamente sus parámetros para asegurar su correcto funcionamiento bajo distintos puntos de trabajo.

En este contexto, el presente trabajo propone el desarrollo de un sistema basado en una rutina de control capaz de determinar la matriz de ganancia óptima de un controlador LQR para distintos puntos de operación, ofreciendo así una alternativa más robusta frente a las variaciones dinámicas del sistema.

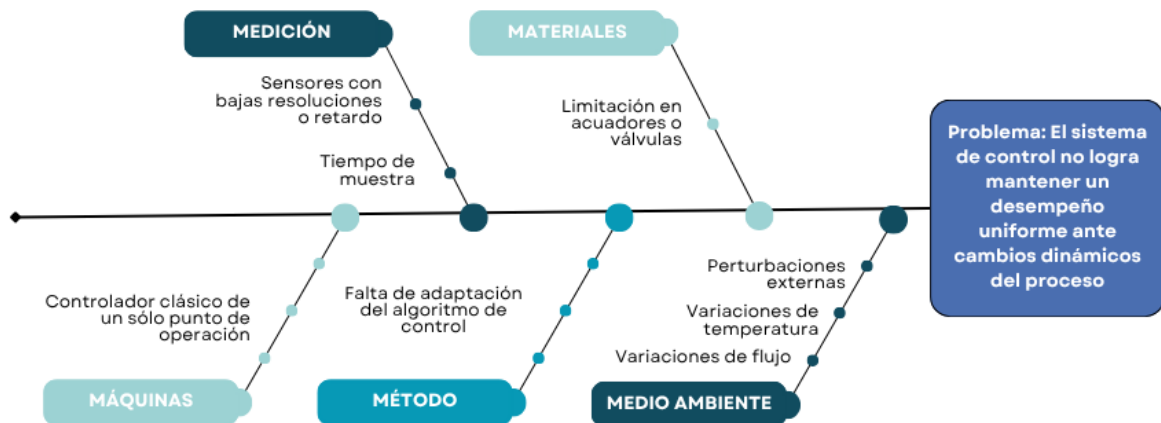
1.2.1 Definición del problema

En los sistemas industriales de control de nivel, el proceso tiene como variables principales el caudal, el nivel actual del estanque y la geometría del recipiente. Todas estas dependencias generan un comportamiento no lineal y variable en el tiempo, por lo que la respuesta del sistema cambia según el punto de operación.

En la figura 1-1, Diagrama de Causa-Efecto para identificar el problema, se presenta el Diagrama de Ishikawa, en el cual se agrupan los factores que influyen en el problema central: “El sistema de control no logra mantener un desempeño uniforme entre cambios dinámicos del proceso”.

Esta incapacidad de mantener un comportamiento consistente en distintos puntos de operación constituye el problema principal que motiva este trabajo. En un sistema de control de nivel donde las condiciones varían constantemente, resulta necesario adaptar la estrategia de control; por ello, se plantea el desarrollo de un sistema capaz de ajustar la ganancia de un controlador LQR según el punto de operación.

Para lograrlo, se implementa una estructura de Gain Scheduling basada en modelos lineales identificados en diferentes condiciones del proceso, permitiendo seleccionar la ganancia óptima y asegurar un rendimiento adecuado en todo el rango operativo.



Fuente: Elaboración propia.

Figura 1-1. Diagrama de Causa-Efecto para identificar el problema.

1.2.2 Importancia de resolverlo

La resolución de este problema permitirá realizar una investigación adecuada de controladores poco habituales, demostrando los siguientes beneficios frente a controladores clásicos:

- Mejora del desempeño dinámico: el controlador LQR puede ajustar su acción en función de un criterio de optimización, reduciendo el error y el costo de control.
- Eficacia a distintos puntos de operación: el uso de Gain Scheduling permite recalcular las ganancias adecuadas según el nivel, caudal o carga actual.
- Optimización del proceso: se logra una regulación más eficiente del nivel, reduciendo tiempos de llenado, sobreimpulsos y energía gastada.

En conclusión, el proyecto busca demostrar la viabilidad de emplear una estructura de Gain Scheduling para la sintonización y selección de la ganancia del controlador LQR dentro de un entorno de simulación industrial.

1.2.3 Involucrados

Los problemas abordados en este trabajo resultan de interés a estudiantes y profesionales de áreas afines a la automatización y control industrial, que deseen investigar sobre la comunicación entre sistemas físicos y entornos de programación, particularmente mediante el protocolo ModBus TCP/IP.

Así mismo, el desarrollo propuesto promueve la utilización de controladores modernos y herramientas, como Python, para la creación de soluciones industriales.

1.3 SOLUCIÓN DEL PROBLEMA

Para abordar el problema mencionado, se propone el desarrollo de un sistema de control adaptativo y adquisición de datos, implementado en Python y comunicado con el entorno Factory I/O mediante el protocolo Modbus TCP/IP.

El sistema actuará como un analizador de datos, capaz de ejecutar rutinas automáticas de llenado y vaciado del estanque bajo distintas condiciones de operación. Se registrarán las variables relacionadas al nivel, flujo, apertura de válvulas y comportamiento dinámico del proceso.

El propósito de esta etapa es analizar la respuesta del sistema frente a diferentes estímulos, generando un conjunto de datos que permita identificar modelos lineales en distintos puntos de operación. Con estos modelos, se determina la matriz de ganancia del controlador LQR asociada a cada condición del proceso, estableciendo posteriormente una estructura de Gain Scheduling que selecciona la ganancia adecuada según el nivel actual del estanque.

De esta forma, el sistema no sólo cumple con la función de controlar el nivel del estanque, sino también de evaluar y registrar su desempeño en distintas condiciones de operación, proporcionando la base para implementar una estrategia de control más robusta frente a variaciones en la dinámica del proceso.

1.3.1. Requerimientos

El sistema por desarrollar deberá cumplir con los siguientes requerimientos:

- Adquisición de datos: el sistema debe ejecutar rutinas de registro periódico de las variables del proceso con una frecuencia de muestreo constante.
- Control de proceso: el sistema ejecutará rutinas de llenado y vaciado del estanque incorporando su detección de umbrales de nivel.
- Plataforma de simulación: se requerirá del software Factory I/O, complementando su módulo de comunicación ModBus TCP/IP, que permitirá su comunicación en tiempo real.
- Generación de dataset: los datos recopilados durante la prueba tienen que ser almacenados en formato CSV.

1.4 OBJETIVO GENERAL

Desarrollar un sistema de control adaptativo para un proceso de llenado y vaciado de un estanque, utilizando un controlador LQR cuya matriz de ganancia se determine a partir de modelos lineales identificados en distintos puntos de operación, integrando un entorno de simulación y adquisición de datos en tiempo real.

1.4.1 Objetivos específicos

Para cumplir con el objetivo general, se deberían cumplir los siguientes objetivos específicos:

- Implementar una plataforma de simulación y adquisición de datos.
- Diseñar e implementar rutinas de vaciado y llenado bajo diferentes condiciones de operación.
- Registrar variables del proceso, tales como nivel, flujo, apertura de válvulas y su comportamiento dinámico.
- Generar un conjunto de datos representativo del comportamiento del sistema en distintos puntos de operación.

- Identificar modelos lineales del proceso para cada tramo operativo del estanque.
- Calcular las matrices de ganancia del controlador LQR para cada modelo identificado.
- Evaluar el desempeño del sistema propuesto, analizando estabilidad, precisión y respuesta ante perturbaciones.

1.5 CONTROLADOR LQR

El controlador LQR es una técnica de control óptimo que utiliza la representación del sistema en espacio de estados. Su finalidad es encontrar una ganancia que permita regular el sistema minimizando un costo que penaliza tanto el error como el esfuerzo de control. Esta característica lo convierte en una herramienta adecuada para procesos cuya dinámica cambia según el punto de operación, como el estanque estudiado.

1.5.1 Representación en variables de estado

Un sistema variable en el tiempo puede modelarse mediante ecuaciones diferenciales que representan la evolución temporal de sus variables internas. En el enfoque de control, estas ecuaciones se representan como variables de estado.

Se considera el sistema dinámico lineal e invariante en el tiempo:

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

$$y(t) = Cx(t) + Du(t)$$

1.5.2 Función de costo del Controlador LQR

El objetivo del control óptimo es diseñar una ley de control que haga que el comportamiento del sistema sea estable y eficiente, minimizando el costo de ello.

$$J(u) = \int_0^{\infty} (X(t)^T Q X + u(t)^T R u(t)) dt$$

Interpretación:

- El primer término $X(t)^T Q X$ mide qué tan lejos están los estados del sistema de los valores deseados.
- El segundo término $u(t)^T R u(t)$ mide el costo asociado al esfuerzo de control.
- El objetivo de un controlador LQR es encontrar una señal de control $u(t)$ que minimice el valor de $J(u)$ logrando un equilibrio entre mantener los datos regulador y no gastar demasiado esfuerzo en el control

1.5.3 Ley de control

La acción de control óptima posee la siguiente forma:

$$u(t) = -Kx(t)$$

Donde K corresponde a la matriz de ganancias del controlador LQR, calculada a partir de la solución de la ecuación de Riccati, y $x(t)$ es el vector de estados del sistema.

1.5.4 Ecuación algebraica de Riccati

La ecuación algebraica de Riccati es una ecuación matricial no lineal que está en el contexto del control óptimo. Su solución permite obtener la matriz P , la cual representa cómo el controlador corrige los estados del sistema. Para obtener K se resuelve mediante la ecuación algebraica de Riccati.

$$A^T P + P A - P B R^{-1} B^T P + Q = 0$$

1.6 **ANTECEDENTES GENERALES**

El presente trabajo enmarca una metodología experimental cuantitativa, cuyo objetivo es obtener un conjunto de datos representativo del comportamiento dinámico de un estanque de nivel sometido a control mediante un controlador LQR.

La metodología se estructura en cuatro etapas:

- Diseño del experimento de simulación.
- Implementación de un sistema de adquisición de datos y control.
- Generación de dataset.
- Procesamiento y organización del dataset.

En este capítulo se describen en detalle estas etapas, junto con los criterios utilizados para la selección de variables, diseño de pruebas y correcciones de las mediciones.

1.7 **SISTEMA DE SIMULACIÓN Y ADQUISICIÓN DE DATOS**

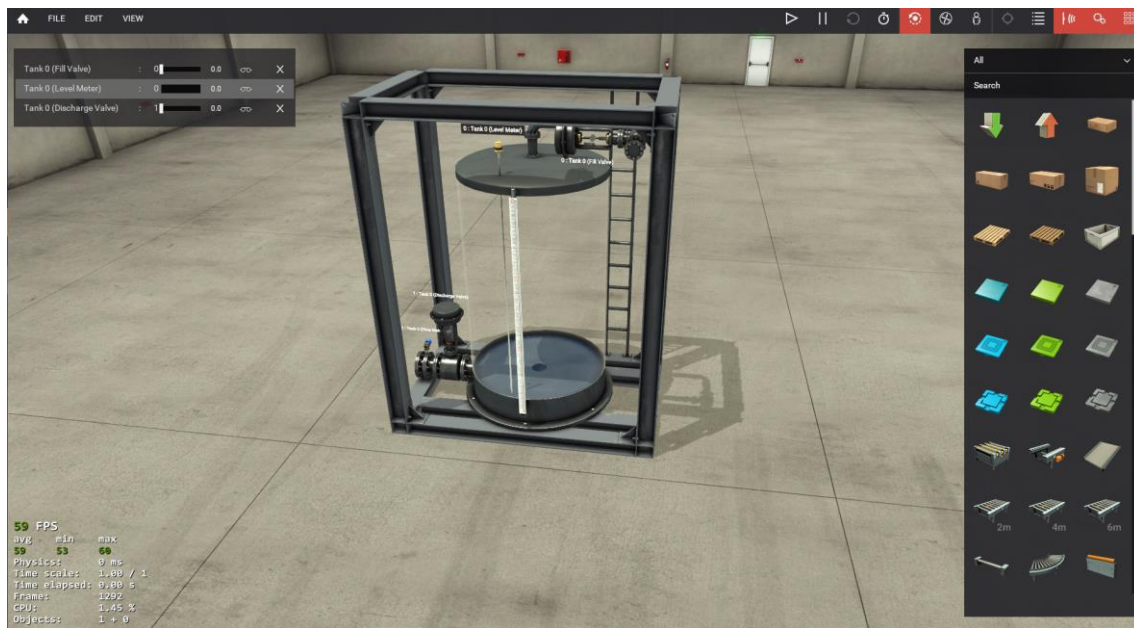
El proceso bajo estudio es un estanque de nivel modelado en Factory I/O, el cual tiene los siguientes sensores disponibles.

- Caudal de entrada.
- Caudal de salida.
- Medición de nivel.
- Medición de flujo.

Cada uno de dichos valores son variables que comprenden un valor entre 0 a 1000 para su funcionamiento.

El comportamiento del sistema es no lineal, ya que depende del nivel actual y de las características propias del software otorgadas al estanque. El objetivo del control es regular el nivel alrededor de una referencia deseada, actuando sobre la válvula de entrada.

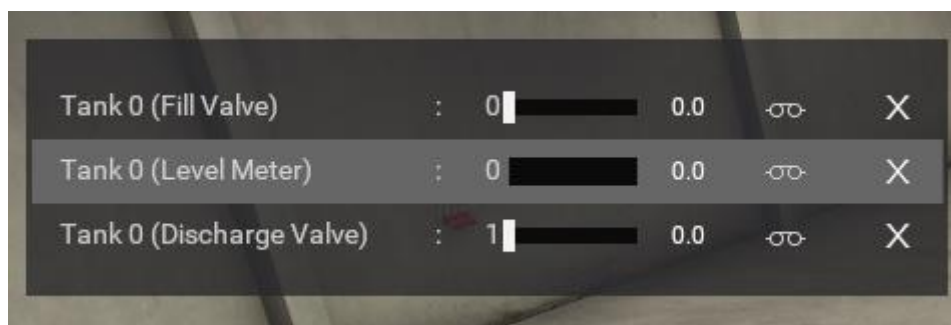
En la figura 1-2 se presenta el estanque de agua modelado en la plataforma *Factory I/O*, el cual constituye el proceso de simulación utilizado en este estudio.



Fuente: Software de Factory I/O.

Figura 1-2. Estanque de agua modelado en la plataforma Factory I/O.

En la figura 1-3 se muestra la interfaz de la plataforma, donde se observan las variables de operación disponibles: apertura de válvulas de entrada y salida, sensor de nivel y sensor de flujo.



Fuente: Software de Factory I/O.

Figura 1-3. Interfaz de la plataforma mostrando las variables de operación.

CAPÍTULO 2: ENSAYO EN LAZO ABIERTO

2 **ENSAYO EN LAZO ABIERTO**

En este capítulo se presenta el procedimiento utilizado para realizar los ensayos en lazo abierto y obtener la información necesaria para identificar el comportamiento dinámico del sistema. Para ello, se configura el entorno de simulación y la comunicación entre Factory I/O y Python mediante el protocolo Modbus TCP/IP. Posteriormente, se desarrolla la rutina de adquisición de datos para registrar las variables del proceso y generar los conjuntos de datos necesarios para la identificación de los modelos lineales en distintos puntos de operación.

A partir de los modelos obtenidos, se analizan los parámetros asociados al diseño del controlador LQR y la solución de la ecuación de Riccati. Finalmente, el procedimiento se repite en diferentes condiciones de operación con el propósito de obtener los modelos y ganancias que posteriormente conformarán la estructura de Gain Scheduling.

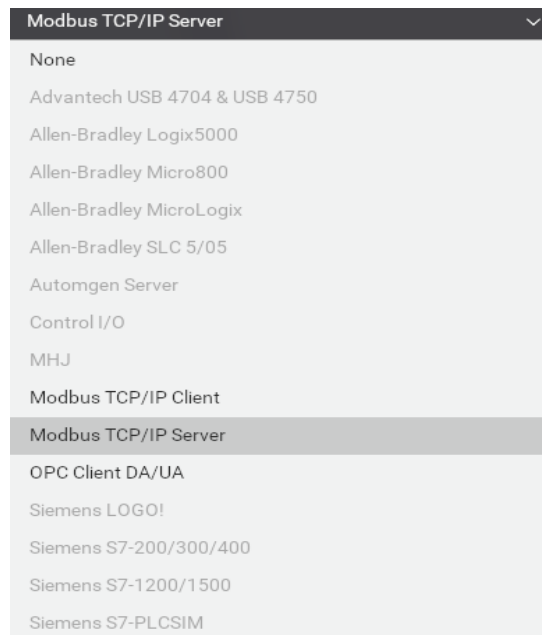
2.1 **ENTORNO DE PRUEBAS**

Para llevar a cabo el experimento fue necesario configurar correctamente el entorno de pruebas, asegurando que tanto el software de simulación como el código de control pudieran comunicarse sin problemas. En esta sección se describen los ajustes iniciales que permiten establecer dicha comunicación y preparar el sistema para la adquisición de datos.

2.1.1 **Comunicación entre Python y Factory I/O**

Para establecer comunicación entre Python y Factory I/O fue necesario realizar una configuración inicial dentro del software de simulación.

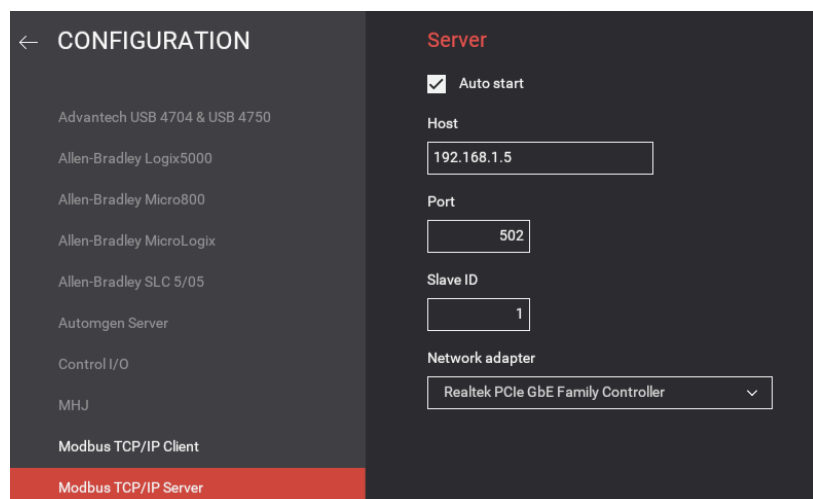
El primer paso consiste en seleccionar en Factory I/O el controlador Modbus TCP/IP Server, el cual permite establecer la comunicación entre la planta simulada y el programa en Python. La selección de este controlador dentro de las opciones disponibles en Factory I/O se presenta en la Figura 2-1.



Fuente: Software Factory I/O.

Figura 2-1. Selección del controlador Modbus TCP/IP Server en Factory I/O.

Una vez seleccionado el servidor Modbus TCP/IP, se deben configurar los parámetros necesarios para establecer la conexión. Como se muestra en la Figura 2-2, la interfaz permite visualizar la dirección IP del equipo y definir el puerto utilizado para la comunicación. Estos parámetros deben coincidir con los utilizados posteriormente por el programa desarrollado en Python.



Fuente: Software Factory I/O.

Figura 2-2. Configuración del servidor Modbus TCP/IP en Factory I/O.

La interacción entre el código de control y Factory I/O se realiza mediante el protocolo Modbus TCP/IP. En este protocolo, los valores obtenidos desde la planta se leen como Input Registers y las señales de control, como la apertura de las válvulas, se escriben en Holding Registers.

La asignación de las variables del estanque a los registros Modbus utilizados durante los experimentos se presenta en la Figura 2-3.



Fuente: Software Factory I/O.

Figura 2-3. Asignación de variables del proceso a registros Modbus en Factory I/O.

Con esta configuración, se propone utilizar el siguiente fragmento de código para la adquisición de datos, implementado a través de un objeto que se encarga de realizar las mediciones y enviar la información correspondiente.

2.2 RUTINA DE ADQUISICIÓN DE DATOS

Para implementar la comunicación con Factory I/O, y obtener las variables del proceso en tiempo real, se desarrolló un script en Python que permite leer los sensores, controlar las válvulas y registrar las mediciones en un archivo CSV. Con el fin de mantener el código organizado y facilitar el uso posterior en distintos experimentos, se definió una clase llamada Estanque, la cual encapsula todas las funciones necesarias para conectarse al servidor Modbus, leer datos y ejecutar rutinas de prueba.

El script comienza importando las librerías necesarias para la adquisición de datos, comunicación Modbus y visualización de gráficos. Estas bibliotecas permiten manejar archivos CSV, trabajar con tiempos de muestreo y establecer la conexión con Factory I/O mediante el protocolo Modbus TCP/IP.

La simulación se ejecuta a una velocidad equivalente a cuatro veces la dinámica real del proceso. Bajo esta condición, un intervalo de muestreo de 0.25 segundos permite capturar adecuadamente las variaciones rápidas de las variables sin perder información relevante.

```
import csv
import time
from datetime import datetime
from pyModbusTCP.client import ModbusClient
import matplotlib.pyplot as plt
FactoryIO_IP = "192.168.1.5" # Dirección de IP
SampleTime = 0.25
Tiempo_Experimento_Segundos = 240
Cantidad_Experimentos = 1
apertura_entrada = 0
```

La clase Estanque agrupa todas las funciones que permiten interactuar con la planta simulada. Dentro de esta clase se implementan métodos para conectarse al servidor, leer el nivel del estanque y modificar la apertura de las válvulas. También, incluye un método para ejecutar un experimento completo y almacenar los resultados obtenidos.

```

class Estanque:
    def __init__(self, host=FactoryIO_IP, port=502,
sample_time=SampleTime):
        self.client = ModbusClient(host=host, port=port)
        self.sample_time = sample_time
        #Estados iniciales en 0
        self.nivel = 0
        self.flujo = 0
        self.control_output = 0
        self.data_log = []

    def conectar(self):
        self.client.open()
        time.sleep(1)

    def desconectar(self):
        self.client.close()

    def leer_sensores(self):
        nivel = self.client.read_input_registers(1, 1)
        if nivel:
            self.nivel = nivel[0]

    def escribir_valvula_salida(self, value):
        self.client.write_single_register(1, value)

    def escribir_valvula_entrada(self, value):
        self.client.write_single_register(0, value)

    def vaciar_estanque(self):
        self.escribir_valvula_entrada(0)
        self.escribir_valvula_salida(0)
        self.client.write_single_coil(0, True)
        time.sleep(0.5)
        self.client.write_single_coil(0, False)
        time.sleep(1)

    def ejecutar_experimento(self, salida, entrada, delta):
        self.data_log = []
        t0 = time.time()

```

```

total_steps = int(Tiempo_Experimento_Segundos / self.sample_time)

self.escribir_valvula_salida(salida)
mid_step = total_steps // 2
for i2 in range(total_steps):
    if i2 == mid_step:
        entrada = int(entrada * delta)
        self.escribir_valvula_entrada(entrada)
        # leer sensores
        self.leer_sensores()
        # guardar log de datos
        self.data_log.append({
            "time": round(time.time() - t0, 2),
            "nivel": self.nivel,
            "entrada": entrada,
            "salida": salida
        })

    time.sleep(self.sample_time)

# al final, cerrar válvulas
self.escribir_valvula_entrada(0)
self.escribir_valvula_salida(0)

```

Finalmente, el método guardar permite exportar los datos recopilados a un archivo CSV. Además, genera automáticamente un gráfico en formato PNG que muestra la evolución del nivel y la apertura de la válvula durante el experimento.

```

def guardar(self, contador):
    filename = f"dataset_estanque 900.csv"
    keys = list(self.data_log[0].keys())
    with open(filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=keys, delimiter=';')
        writer.writeheader()
        writer.writerows(self.data_log)

    print(f"Datos guardados: {filename}")

```

```

# Generar gráfico si matplotlib está disponible
if plt is None:
    print("matplotlib no disponible, se omite la generación de
gráfico.")
    return

try:
    times = [row['time'] for row in self.data_log]
    niveles = [row['nivel'] for row in self.data_log]
    entradas = [row['entrada'] for row in self.data_log]

    fig, ax1 = plt.subplots()
    ax1.plot(times, niveles, 'b-', label='Nivel')
    ax1.set_xlabel('Tiempo (s)')
    ax1.set_ylabel('Nivel', color='b')
    ax1.tick_params(axis='y', labelcolor='b')

    ax2 = ax1.twinx()
    ax2.plot(times, entradas, 'r--', label='Apertura entrada')
    ax2.set_ylabel('Apertura entrada', color='r')
    ax2.tick_params(axis='y', labelcolor='r')

    plt.title(f"Experimento {contador} - {filename}")
    fig.tight_layout()

    plot_filename = filename.replace('.csv', '.png')
    fig.savefig(plot_filename)
    plt.close(fig)
    print(f"Gráfico guardado: {plot_filename}")
except Exception as e:
    print(f"Error al generar gráfico: {e}")

```

La siguiente sección presenta el código principal del proyecto. Este bloque se encarga de ejecutar el experimento, enviando a Factory I/O los valores correspondientes a la válvula de entrada, la válvula de salida y el momento en que se aplicará la variación sobre la apertura de la válvula de entrada.

```

if __name__ == "__main__":
    estanque = Estanque()

    try:
        estanque.conectar()
        entrada = 500
        salida = 500
        delta = 1.05
        estanque.vaciar_estanque()
        estanque.ejecutar_experimento(salida, entrada, delta)
        estanque.guardar(contador)
        estanque.desconectar()

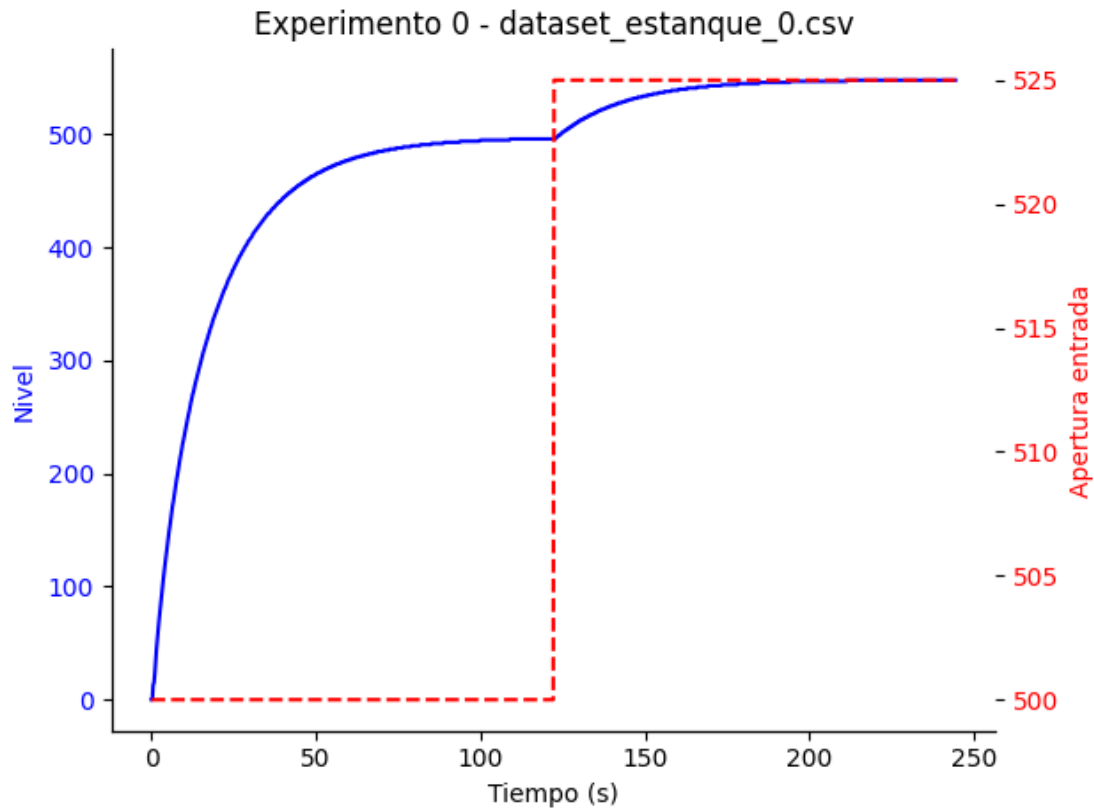
```

Tras realizar el primer experimento, se procede a identificar el sistema.

2.3 IDENTIFICACIÓN DE UN MODELO LPV

Para este primer ensayo se tomó una muestra de datos con el objetivo de identificar las ecuaciones de estado utilizando la herramienta *System Identification* de MATLAB. En este experimento inicial se asignó un valor de 500 a la válvula de entrada y 500 a la válvula de salida, lo que corresponde a un 50% de apertura en cada caso, dado que el programa trabaja en un rango de 0 a 1000. Posteriormente, se aplicó un incremento del 5% sobre dicho valor, equivalente a un ajuste hasta 525 unidades.

Con esta configuración se obtuvo la respuesta experimental presentada en la Figura 2-4, donde se observa inicialmente el proceso de estabilización del nivel del estanque y, posteriormente, la respuesta del sistema ante el incremento aplicado a la válvula de entrada. Esta variación permite analizar la dinámica del proceso alrededor del punto de operación $(u_q, y_q) = (500, 496)$, definido a partir de la condición de equilibrio establecida en torno al 50 % de apertura de las válvulas.



Fuente: Elaboración propia.

Figura 2-4. Experimento para identificación de sistemas.

Tras obtener la muestra de datos, fue necesario adaptar la información para que pudiera ser analizada correctamente por la herramienta *System Identification*. Para ello se realizó un preprocesamiento en dos etapas:

1. Antes de aplicar el delta, todas las variables asociadas al nivel debían quedar centradas en cero. Esto significa que el nivel medido antes del cambio debía considerarse como la línea base del sistema.
2. Después de aplicar el delta, a cada uno de los niveles registrados se le restó el valor estacionario correspondiente al punto de operación. De esta forma, la respuesta dinámica queda expresada en relación al equilibrio inicial, lo cual es necesario para identificar un modelo lineal alrededor de ese punto.

Con este ajuste se obtiene una señal adecuada para ser procesada por *Ident*. Utilizando un estimador basado en un modelo de espacio de estados, fue posible determinar los parámetros asociados a la ecuación del sistema. El análisis se realizó en

el plano discreto para asegurar compatibilidad con el controlador que se implementará posteriormente.

El modelo identificado tiene la siguiente estructura:

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) + Ke(t) \\ y(t) &= Cx(t) + Dy(t)\end{aligned}$$

Los parámetros obtenidos se presentan en la Tabla 2-1.

Tabla 2-1. Resultados de la primera identificación de sistemas.

Variable de estado	Modelo de espacio de estados
A	0.9882
B	0.00008905
C	0.2768
D	0
K	0.0017
Aceptación	98.25%

Fuente: Elaboración propia.

2.4 ECUACIÓN LINEAL DE RICCATI

Una vez identificado el modelo del sistema, el siguiente paso consiste en determinar la matriz de ganancia óptima del controlador LQR. Para ello es necesario resolver la ecuación algebraica de Riccati, la cual aparece en el contexto del control óptimo y permite obtener la matriz P , que describe cómo el controlador penaliza las desviaciones de los estados del sistema.

El método parte de definir dos matrices de ponderación:

- Q , que determina la importancia relativa de los estados,
- R , que penaliza la magnitud de la señal de control.

En este trabajo, la matriz Q puede obtenerse a partir de la identidad:

$$Q = C^T C$$

De este modo, la única variable libre es la matriz R , la cual se ajusta manualmente para controlar el esfuerzo de la acción de control. Al variar R es posible

generar diferentes configuraciones del controlador, obteniendo distintas ganancias y, por consecuencia, diferentes comportamientos dinámicos del sistema.

En MATLAB, este procedimiento puede resolverse utilizando las funciones *icare* o *dlqr*, dependiendo si se trabaja en el dominio continuo o discreto:

$$[X, K, L] = \text{icare}(A, B, Q, R) \rightarrow \text{Para modelos continuos}$$

$$[K, X, L] = \text{dlqr}(A, B, Q, R) \rightarrow \text{Para modelos discretos}$$

Si se parte desde un modelo continuo, las matrices pueden discretizarse utilizando:

$$x_{k+1} = A_d x_k + B_d u_k + K_d e_k y_k = C x_k + e_k$$

Dónde:

$$\begin{aligned} A_d &= e^{AT_s} \\ B_b &= \left(\int_0^{T_s} e^{AT} dt \right) B = B \frac{e^{AT_s} - 1}{A} \\ K_b &= \left(\int_0^{T_s} e^{AT} dt \right) K = K \frac{e^{AT_s} - 1}{A} \end{aligned}$$

Con ello, al iterar el valor de R en una escala logarítmica es posible obtener un conjunto de ganancias distintas, cada una asociada a un polo diferente del sistema. Estos polos determinan el comportamiento dinámico bajo diferentes niveles de penalización del esfuerzo de control. En este contexto, aumentar el valor de Q hace que el controlador sea más agresivo, ya que se penalizan con mayor intensidad las desviaciones del estado, mientras que aumentar el valor de R lo hace más conservador, al dar mayor peso a la minimización del esfuerzo de control. De esta manera se genera una familia de controladores que permite regular cuánto esfuerzo puede ejercer la entrada $u(t)$ sin comprometer la estabilidad.

Los resultados obtenidos para los distintos valores de Q se presentan en la Tabla 2-2, donde se muestran las ganancias K , los polos de lazo cerrado L y los valores de P correspondientes al primer modelo identificado.

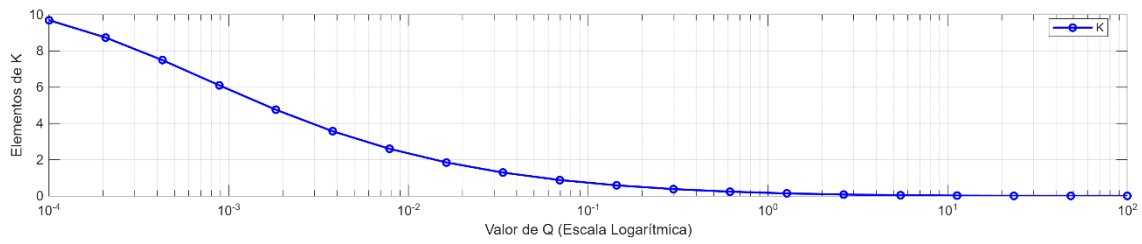
Este enfoque se conoce como Gain Scheduling, y corresponde a disponer de múltiples ganancias dentro del mismo sistema, seleccionándolas según el punto de operación.

Tabla 2-2. Modelo de estructura de ganancia para el primer modelo.

Q	K	L	P
0.0001	9.6979	0.1246	0.0874
0.0002	8.7391	0.2100	0.0967
0.0004	7.4927	0.3210	0.1122
0.0009	6.1059	0.4445	0.1366
0.0018	4.7563	0.5647	0.1734
0.0038	3.5711	0.6702	0.2269
0.0078	2.6040	0.7563	0.3034
0.0162	1.8541	0.8231	0.4107
0.0336	1.2928	0.8731	0.5586
0.0695	0.8830	0.9096	0.7579
0.1438	0.5896	0.9357	1.0178
0.2976	0.3831	0.9541	1.3419
0.6158	0.2404	0.9668	1.7199
1.2743	0.1446	0.9754	2.1215
2.6367	0.0828	0.9809	2.4980
5.4556	0.0450	0.9842	2.8013
11.2884	0.0234	0.9861	3.0104
23.3572	0.0118	0.9872	3.1364
48.3293	0.0058	0.9877	3.2053
100.0000	0.0029	0.9880	3.2409

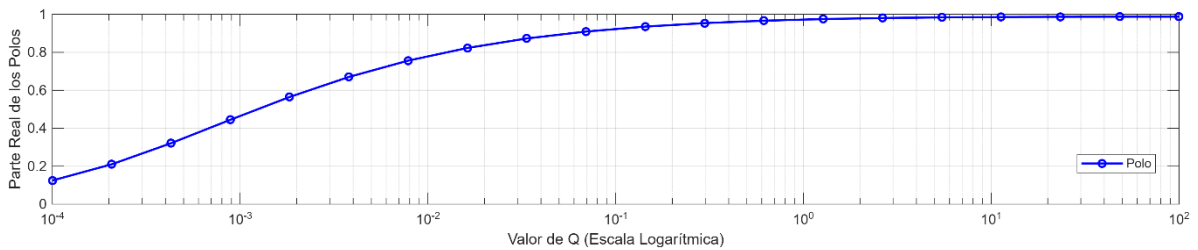
Fuente: Elaboración propia.

El efecto de la variación de Q sobre los parámetros obtenidos para el controlador se presenta gráficamente en las Figuras 2-5, 2-6 y 2-7. La Figura 2-5 muestra el comportamiento de la ganancia K , la Figura 2-6 presenta la evolución de los polos de lazo cerrado y la Figura 2-7 muestra el comportamiento de la solución P de la ecuación de Riccati.



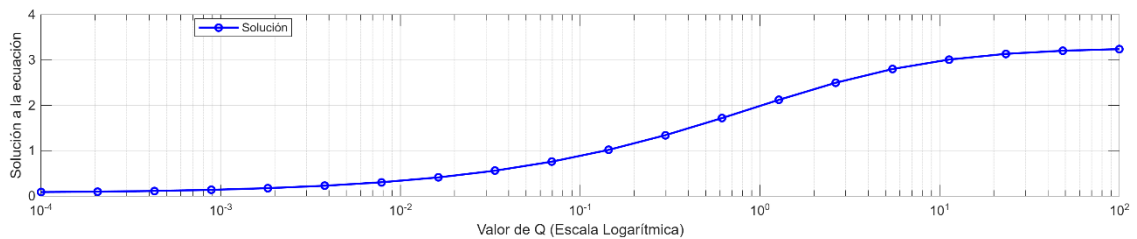
Fuente: Elaboración propia.

Figura 2-5. Impacto de Q en la ganancia K .



Fuente: Elaboración propia.

Figura 2-6. Impacto de Q en los polos de lazo cerrado.



Fuente: Elaboración propia.

Figura 2-7. Impacto de Q en la Solución de la ecuación de Riccati.

Como se aprecia en las Figuras 2-5, 2-6 y 2-7, cuando los valores de Q son pequeños, el controlador prioriza minimizar el esfuerzo de control antes que regular el estado. Esto provoca que la ganancia K aumente, ya que el sistema necesita aplicar acciones más fuertes para compensar el bajo peso asignado al estado en la función de costo.

En cambio, para valores más altos de Q , la penalización sobre las desviaciones del estado se vuelve más significativa. En este caso, la solución de Riccati converge a valores que generan acciones de control más suaves y estables. Los polos del sistema se

desplazan hacia una región estable y dejan de variar significativamente, lo que indica que el sistema se vuelve menos sensible a incrementos adicionales en Q .

Tanto las matrices X , K como los polos asociados evidencian que el sistema alcanza un comportamiento estable bajo este rango de parámetros.

Con estos resultados, se procede a realizar un ensayo en Factory I/O para validar la respuesta obtenida mediante simulación y comprobar que las ganancias generadas para este primer punto de operación efectivamente representan el comportamiento real del sistema.

2.5 ITERACIÓN DEL EXPERIMENTO

Tras obtener una matriz de ganancia para el primer punto de operación, fue necesario verificar si el controlador mantenía un desempeño adecuado en otros niveles del estanque. Para ello, se repitió el experimento en distintos puntos de operación, obteniendo un total de nueve modelos lineales de espacio de estados y sus respectivas matrices de ganancia.

Los parámetros obtenidos para cada uno de los nueve modelos identificados se presentan en la Tabla 2-3, donde se muestran los valores correspondientes a cada punto de operación y los parámetros asociados al modelo de espacio de estados.

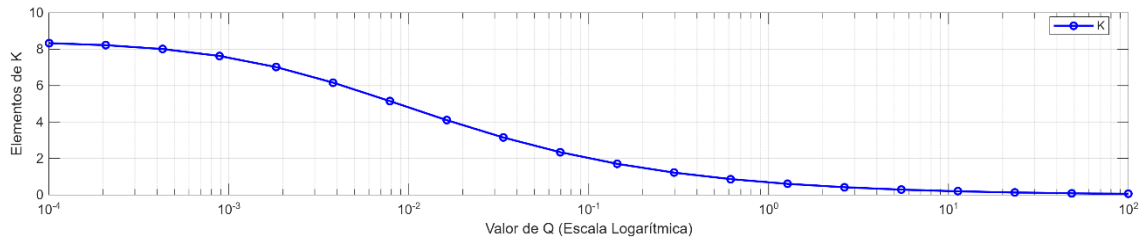
Tabla 2-3. Modelo de estados para nueve sistemas lineales.

Entrada	Salida	A	B	C	D	E
100	100	0.9974	0.00012	0.7336	0	0.04701
200	200	0.9951	0.00009	0.4658	0	0.00120
300	300	0.9928	0.00009	0.3467	0	0.00170
400	400	0.9905	0.00009	0.2994	0	0.00180
500	500	0.9882	0.00009	0.2768	0	0.00170
600	600	0.9861	0.00009	0.2602	0	0.00190
700	700	0.9839	0.00010	0.2481	0	0.00190
800	800	0.9815	0.00010	0.2366	0	0.00190
900	900	0.9794	0.00010	0.2276	0	0.00200

Fuente: Elaboración propia.

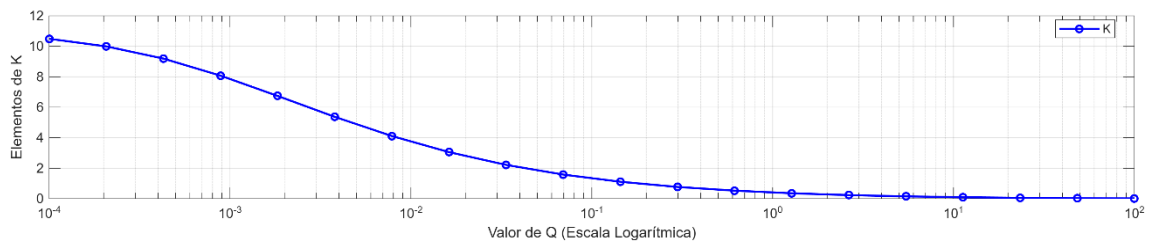
A partir de cada uno de los modelos identificados se repitió el procedimiento de cálculo de la ganancia K , variando el valor de Q en una escala logarítmica. Los

resultados obtenidos para los nueve puntos de operación se presentan en las Figuras 2-8 a 2-16, donde se observa la evolución de la ganancia K en función de Q para cada sistema.



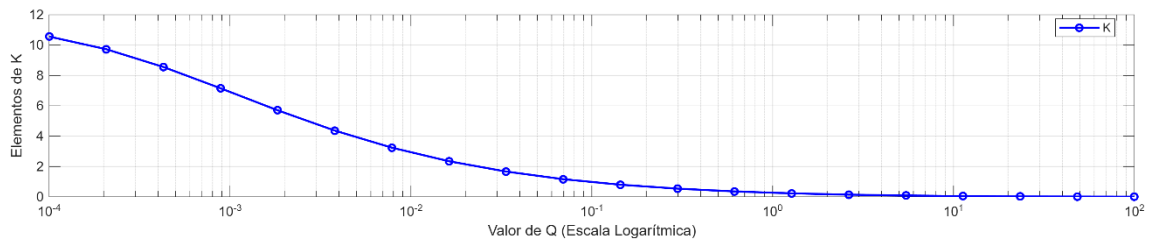
Fuente: Elaboración propia.

Figura 2-8. Impacto de Q en la ganancia K para el Sistema 1.



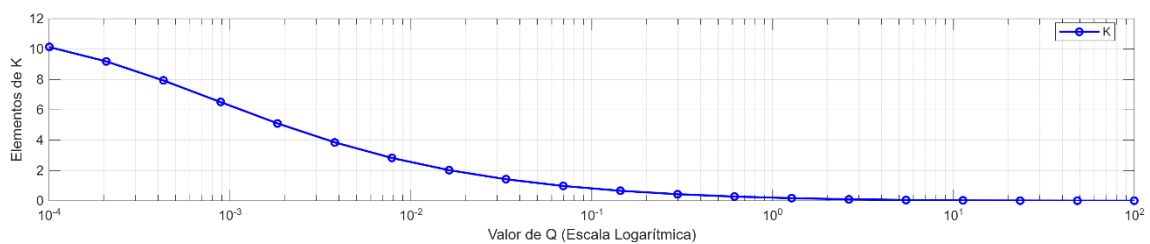
Fuente: Elaboración propia.

Figura 2-9. Impacto de Q en la ganancia K para el Sistema 2.



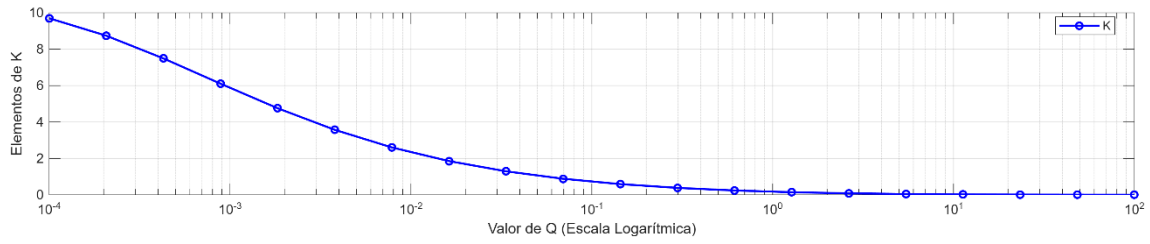
Fuente: Elaboración propia.

Figura 2-10. Impacto de Q en la ganancia K para el Sistema 3.



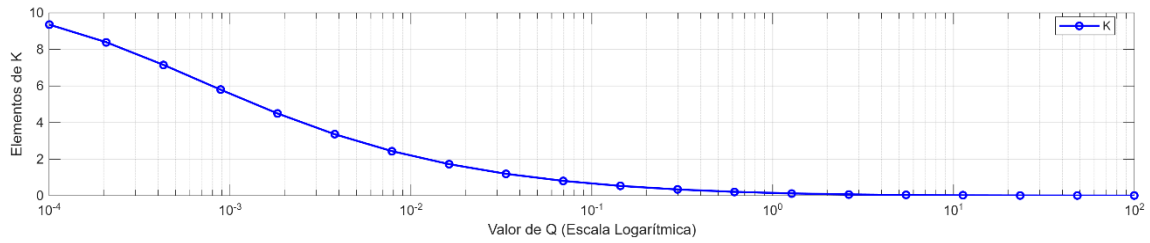
Fuente: Elaboración propia.

Figura 2-11. Impacto de Q en la ganancia K para el Sistema 4.



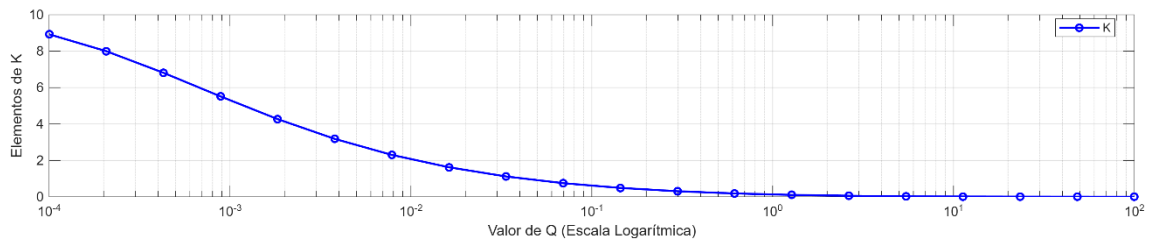
Fuente: Elaboración propia.

Figura 2-12. Impacto de Q en la ganancia K para el Sistema 5.



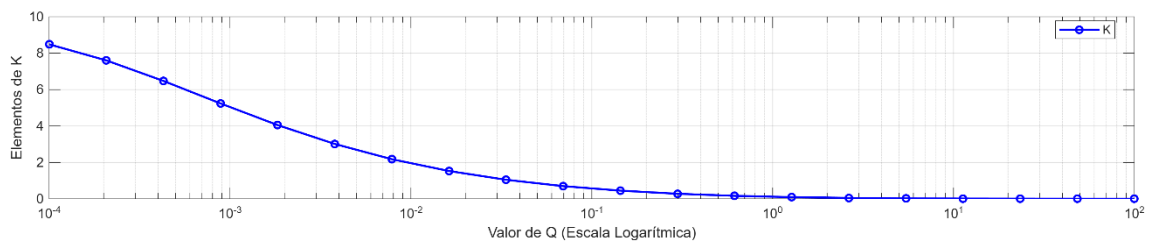
Fuente: Elaboración propia.

Figura 2-13. Impacto de Q en la ganancia K para el Sistema 6.



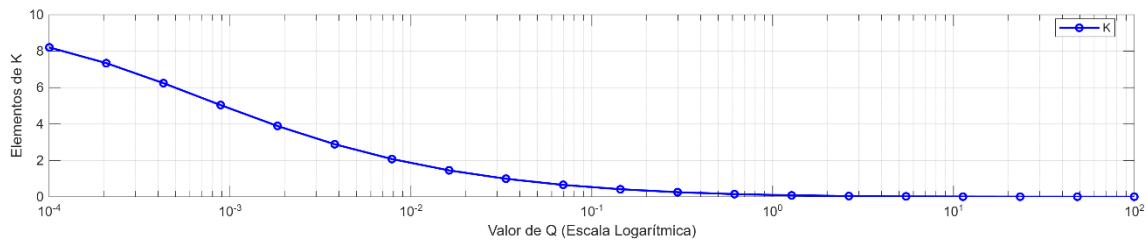
Fuente: Elaboración propia.

Figura 2-14. Impacto de Q en la ganancia K para el Sistema 7.



Fuente: Elaboración propia.

Figura 2-15. Impacto de Q en la ganancia K para el Sistema 8.



Fuente: Elaboración propia.

Figura 2-16. Impacto de Q en la ganancia K para el Sistema 9.

Como se observa en las Figuras 2-8 a 2-16, la variación de Q produce un comportamiento similar en el aumento de la ganancia K para los nueve modelos, sin embargo, los valores obtenidos dependen del punto de operación considerado. La obtención de estos nueve modelos lineales permitió representar de manera más completa el comportamiento del estanque en distintos puntos de operación. Cada identificación entregó valores específicos para las matrices del sistema y para la ganancia óptima del controlador, lo que confirmó que el proceso presenta variaciones dinámicas significativas según el nivel en el que se encuentre. Este análisis evidencia que un único controlador no sería suficiente para cubrir todo el rango operativo, ya que las condiciones de estabilidad y rapidez de respuesta cambian a lo largo del proceso.

Gracias a esta caracterización, fue posible construir una estructura de Gain Scheduling basada en estos nueve modelos, donde cada uno actúa como referencia para un tramo definido del nivel del estanque. Este enfoque permite seleccionar la ganancia más adecuada según el punto de operación, asegurando un comportamiento más estable y una respuesta más consistente frente a perturbaciones o cambios en la demanda.

Con los modelos identificados y las ganancias obtenidas, el siguiente paso consiste en integrar esta información dentro de la lógica de control en lazo cerrado. Esto permitirá evaluar, ahora bajo condiciones de simulación dinámica, si la estrategia de selección de ganancias logra mantener el nivel del estanque regulado de forma eficiente. De esta manera, los resultados obtenidos en este capítulo establecen la base necesaria para desarrollar y probar el controlador LQR con Gain Scheduling en el entorno de Factory I/O, lo que será abordado en el capítulo siguiente.

CAPÍTULO 3: ENSAYO DE LAZO CERRADO

3 ENSAYO DE LAZO CERRADO

En este capítulo se desarrolla la implementación y evaluación del sistema de control a lazo cerrado a partir de los modelos obtenidos. Para ello se integra el controlador LQR con una estrategia de Gain Scheduling, permitiendo seleccionar la ganancia de control en función al punto de operación del estanque.

3.1 LÓGICA DE CONTROL DISCRETA

En este capítulo se presenta la implementación del controlador LQR en lazo cerrado utilizando los modelos identificados en el capítulo anterior y la estructura de Gain Scheduling desarrollada para seleccionar la ganancia adecuada según el punto de operación.

Para llevar a cabo este ensayo, se creó un programa en Python que integra la adquisición de datos, el cálculo de la acción de control y la comunicación en tiempo real con Factory I/O mediante el protocolo Modbus TCP/IP.

El código se encuentra estructurado en diferentes clases y funciones que organizan el proceso de control, permitiendo seleccionar el modelo correspondiente, aplicar la ley LQR, limitar la acción de control y registrar todos los datos necesarios para evaluar el desempeño del sistema.

A continuación, se describe detalladamente cada componente del código, explicando su función dentro de la lógica de control y cómo interactúa con la planta simulada.

3.1.1 Base de datos para selección de modelo

Con el objetivo de organizar los modelos identificados y permitir la aplicación del Gain Scheduling, se creó una estructura que almacena, para cada tramo de operación del estanque, las matrices discretas del sistema y sus respectivas ganancias LQR.

Para ello, se definió primero una lista de valores de Q distribuidos logarítmicamente, que sirven como referencia para generar diferentes configuraciones

del controlador. Luego, mediante la dataclass `Modelo_Discreto`, se registró la información relevante de cada modelo: nombre del tramo, rango de setpoint, modelos de espacio, valores de referencia y la tabla de pares obtenida para ese punto de operación.

Finalmente, se generaron nueve modelos discretos que cubren todo el rango operativo del estanque, y todos fueron agrupados en `MODEL_TABLE`. Esta base de datos permite que el Gain Scheduler seleccione automáticamente el modelo adecuado según el valor de la salida y , garantizando que el controlador se ajuste al comportamiento real del sistema en cada punto de operación.

```
Log_table = [0.0001, 0.000206914, 0.000428133, 0.000885867, 0.001832981,
0.00379269, 0.0078476, 0.016237767, 0.06951928, 0.143844989, 0.297635144,
0.615848211, 1.274274986, 2.636650899, 5.455594781, 11.28837892, 23.35721469,
48.32930239, 100]
```

```
@dataclass
```

```
class Modelo_Discreto:
```

```
    name: str
```

```
    sp_min: float
```

```
    sp_max: float
```

```
    Ad: float
```

```
    Bd: float
```

```
    Cd: float
```

```
    Dd: float
```

```
    Kd: float
```

```
    u0: float = U0_DEFAULT
```

```
    y0: float = Y0_DEFAULT
```

```
    gain_table: List[Tuple[float, float]] = field(default_factory=list)
```

```
Modelo1 = Modelo_Discreto(
```

```
    name= "Setpoint 51 - 150",
```

```
    sp_min= 51.0,
```

```
    sp_max= 150.0,
```

```
    Ad= 0.9974,
```

```
    Bd= 1.1816e-04,
```

```
    Cd= 0.7336,
```

```
    Dd= 0,
```

```

Kd= 4.7013e-04,
u0= 100.0,
y0= 379.0,
gain_table = list(zip(Log_table, [8.331118997, 8.220087839, 8.007588642,
7.627487428, 7.016293121, 6.164029568, 5.149030461, 4.104153876, 3.147190183,
2.342686586, 1.705649162, 1.221496421, 0.863699722, 0.604308002, 0.418740075,
0.287233556, 0.194702714, 0.130001192, 0.08507241, 0.054184057]))
)

```

```

Modelo2 = Modelo_Discreto(
    name= "Setpoint 151 - 250",
    sp_min= 151.0,
    sp_max= 250.0,
    Ad= 0.9951,
    Bd= 8.9974e-05,
    Cd= 0.4658,
    Dd= 0,
    Kd= 0.0012,
    u0= 200.0,
    y0= 463.0,
    gain_table = list(zip(Log_table, [10.49228979, 9.994190813, 9.192845113,
8.074725189, 6.742262905, 5.369845668, 4.112404286, 3.055022024, 2.217634862,
1.581215587, 1.110978085, 0.770239812, 0.52674398, 0.354580043, 0.234006886,
0.150485363, 0.093543729, 0.055711603, 0.031578399, 0.017029706]))
)

```

```

Modelo3 = Modelo_Discreto(
    name= "Setpoint 251 - 350",
    sp_min= 251.0,
    sp_max= 350.0,
    Ad= 0.9928,
    Bd= 8.5125e-05,
    Cd= 0.3467,
    Dd= 0,
    Kd= 0.0017,
    u0= 300.0,
    y0= 478.0,
    gain_table = list(zip(Log_table, [10.56214888, 9.728494849, 8.558366403,
7.155530797, 5.703207634, 4.367383568, 3.241025964, 2.347399494, 1.667493786,

```

```
1.16489031, 0.800788601, 0.540937024, 0.357793573, 0.230389562, 0.143295976,
0.085344807, 0.048359017, 0.026066956, 0.013473175, 0.006761412]))
)
```

```
Modelo4 = Modelo_Discreto(
    name= "Setpoint 351 - 450",
    sp_min= 351.0,
    sp_max= 450.0,
    Ad= 0.9905,
    Bd= 8.6279e-05,
    Cd= 0.2994,
    Dd= 0,
    Kd= 0.0018,
    u0= 400.0,
    y0= 494.0,
    gain_table = list(zip(Log_table, [10.13563581, 9.188615225, 7.932859236,
6.509132331, 5.10245398, 3.853215132, 2.82572801, 2.024489797, 1.422202999,
0.980919995, 0.6635956, 0.438864894, 0.282123507, 0.174922892, 0.103723214,
0.058475665, 0.031370216, 0.016155047, 0.008088212, 0.003981743]))
)
```

```
Modelo5 = Modelo_Discreto(
    name= "Setpoint 451 - 550",
    sp_min= 451.0,
    sp_max= 550.0,
    Ad= 0.9882,
    Bd= 8.9051e-05,
    Cd= 0.2768,
    Dd= 0,
    Kd= 0.0017,
    u0= 500.0,
    y0= 496.0,
    gain_table = list(zip(Log_table, [9.697932423, 8.739105541, 7.492747952,
6.10587596, 4.756254633, 3.571059627, 2.603968672, 1.8540758, 1.292775331,
0.883041279, 0.589619385, 0.383058496, 0.240441594, 0.144604474, 0.082752463,
0.045004633, 0.023419346, 0.011804331, 0.005833491, 0.002851346]))
)
```

```
Modelo6 = Modelo_Discreto(
    name= "Setpoint 551 - 650",
```

```

    sp_min= 551.0,
    sp_max= 650.0,
    Ad= 0.9861,
    Bd= 9.1304e-05,
    Cd= 0.2602,
    Dd= 0,
    Kd= 0.0019,
    u0= 600.0,
    y0= 497.0,
    gain_table = list(zip(Log_table, [9.358124457, 8.391785455, 7.155150383,
5.798810787, 4.494071683, 3.357990789, 2.436566898, 1.725209025, 1.194596241,
0.808551858, 0.533251001, 0.340714853, 0.209269168, 0.122588691, 0.06820279,
0.036156816, 0.018457871, 0.009190739, 0.004510724, 0.00219684]))
)

```

```

Modelo7 = Modelo_Discreto(
    name= "Setpoint 651 - 750",
    sp_min= 651.0,
    sp_max= 750.0,
    Ad= 0.9839,
    Bd= 9.5414e-05,
    Cd= 0.2481,
    Dd= 0,
    Kd= 0.0019,
    u0= 700.0,
    y0= 497.0,
    gain_table = list(zip(Log_table, [8.925744602, 7.998873697, 6.814134225,
5.516122947, 4.268609084, 3.183134429, 2.303310673, 1.62454171, 1.118744299,
0.751390195, 0.490262787, 0.308745104, 0.186168585, 0.106763389, 0.058158053,
0.03029889, 0.015282833, 0.007555537, 0.003693841, 0.001795427]))
)

```

```

Modelo8 = Modelo_Discreto(
    name= "Setpoint 751 - 850",
    sp_min= 751.0,
    sp_max= 850.0,
    Ad= 0.9815,
    Bd= 1.0014e-04,
    Cd= 0.2366,
    Dd= 0,

```

```

Kd= 0.0019,
u0= 800.0,
y0= 497.0,
gain_table = list(zip(Log_table, [8.485184099, 7.603657991, 6.475920277,
5.239422089, 4.05035622, 3.01539153, 2.176441946, 1.529370235, 1.047569506,
0.698250845, 0.450813381, 0.279956218, 0.165922263, 0.093375085, 0.049985037,
0.025695013, 0.012849095, 0.006321112, 0.003082315, 0.001496222]))
)

```

```

Modelo9 = Modelo_Discreto(
    name= "Setpoint 851 - 950",
    sp_min= 851.0,
    sp_max= 950.0,
    Ad= 0.9794,
    Bd= 1.0303e-04,
    Cd= 0.2276,
    Dd= 0,
    Kd= 0.0020,
    u0= 900.0,
    y0= 497.0,
    gain_table = list(zip(Log_table, [8.208651564, 7.344699627, 6.243976592,
5.041637378, 3.888919068, 2.887886213, 2.077873462, 1.45408793, 0.990430501,
0.655109613, 0.418594301, 0.256489437, 0.149619829, 0.082828602, 0.043712648,
0.022241838, 0.011052502, 0.005418411, 0.002637384, 0.00127909]))
)

```

```

MODEL_TABLE: List[Modelo_Discreto] = [
    Modelo1, Modelo2, Modelo3, Modelo4,
    Modelo5, Modelo6, Modelo7, Modelo8, Modelo9
]

```

3.1.2 Controlador LQR

El objeto ControladorLQR es el encargado de calcular la acción de control a partir del estado estimado del sistema y del modelo seleccionado por el *Gain Scheduler*.

Este controlador utiliza la tabla de ganancias definida en la base de datos del modelo, seleccionando la ganancia correspondiente según el índice indicado.

Dentro del método step, el controlador realiza tres operaciones principales:

- Cálculo de la referencia del estado: A partir de la referencia solicitada y del valor de equilibrio del modelo (y_0), se obtiene el estado de referencia x_{ref} .
- Aplicación de la ley de control LQR: Utilizando la ecuación $u = -K(x - x_{ref})$, se obtiene una variación de control que luego se suma al sesgo u_0 asociado al punto de operación.
- Limitación de la señal de control: Finalmente, la acción calculada se restringe a un rango permitido para evitar saturaciones o comportamientos no deseados.

El método retorna la acción de control resultante junto con información útil para análisis posterior, como el error respecto a la referencia y los valores de K y Q utilizados durante ese instante. Esto permite monitorear el comportamiento del controlador y verificar si la ganancia aplicada es coherente con el punto de operación seleccionado.

```
class ControladorLQR:
    def __init__(self, gain_index: int = 0, u_min: float = UMIN, u_max: float = UMAX):
        self.gain_index = int(gain_index)
        self.u_min = float(u_min)
        self.u_max = float(u_max)

    def set_gain_index(self, gain_index: int):
        self.gain_index = int(gain_index)

    def step(self, xhat: float, sp: float, model: Modelo_Discreto, u_bias: Optional[float] = None):
        if not model.gain_table:
            raise ValueError(f"El modelo {model.name} no tiene tabla de ganancia.")

        if self.gain_index < 0 or self.gain_index >= len(model.gain_table):
            raise IndexError(
                f"Índice {self.gain_index} fuera de rango para {model.name}."
            )

        f"Máximo válido: {len(model.gain_table) - 1}"
    )
```

```

if u_bias is None:
    u_bias = model.u0

Q_lqr, K_lqr = model.gain_table[self.gain_index]

# Referencia de salida -> referencia de estado
dy_ref = float(sp) - float(model.y0)
x_ref = dy_ref / float(model.Cd)

# Ley LQR
du = -K_lqr * (float(xhat) - x_ref)
u_cmd = float(u_bias) + du
u_cmd = clamp(u_cmd, self.u_min, self.u_max)

# Para logging
y_hat = float(model.Cd) * float(xhat) + float(model.y0)
err_y = float(sp) - y_hat

return int(round(u_cmd)), err_y, K_lqr, Q_lqr

```

3.1.3 Gain Scheduler

El Gain Scheduler se encarga de seleccionar automáticamente el modelo discreto más adecuado según el valor de la salida y . Para ello, recorre la base de datos y elige la instancia de `Modelo_Discreto` cuyo rango $[sp_{max}]$ contiene el valor de referencia. Esta selección fija las matrices (D_d), los sesgos (y_0) y la tabla de pares (K) que usará el controlador LQR en ese punto de operación.

```

class GainScheduler:
    def __init__(self, models: List[Modelo_Discreto]):
        if not models:
            raise ValueError("No hay tabla de modelos.")
        self.models = models

    def pick(self, sp: float) -> Modelo_Discreto:
        sp = float(sp)

```

```

for i, m in enumerate(self.models):
    if i < len(self.models) - 1:
        if m.sp_min <= sp < m.sp_max:
            return m
    else:
        if m.sp_min <= sp <= m.sp_max:
            return m

raise ValueError(f"Setpoint {sp} fuera de rango de modelos.")

```

3.1.4 Control de lazo cerrado

La función `Control_Lazo_Cerrado` integra todos los componentes del sistema para ejecutar el control LQR en tiempo real sobre la planta simulada en Factory I/O. Esta función recibe el objeto `Estanque` (encargado de la comunicación Modbus), el `GainScheduler` (que selecciona el modelo según el setpoint), el valor de referencia `sp`, el índice de ganancia `gain_index`, la apertura fija de la válvula de salida y la duración del experimento. A partir de estos parámetros, coordina la lectura de sensores, el cálculo de la acción de control y el registro de datos.

En cada ciclo, se lee el nivel desde Factory I/O y se convierte a estado con el modelo activo:

$$x_{meas} = \frac{y_{meas} - y_0}{C_d}$$

A partir del setpoint, se calcula:

$$x_{ref} = \frac{sp - y_0}{C_d}$$

Esto permite aplicar la ley LQR alrededor del equilibrio del tramo.

El controlador aplica $u = u_{bias} - K(x_{meas} - x_{ref})$, donde u_{bias} es, por defecto, u_0 del modelo. Así se genera la acción de control centrada en el punto de operación.

La salida de control se satura entre $[UMAX]$ y además se limita la pendiente Δu con un máximo DU_MAX por ciclo. Esto evita acciones bruscas que puedan producir inestabilidades.

La válvula de entrada recibe el nuevo comando u_cmd . En paralelo se registra un log de cada instante: tiempo, nivel, setpoint, error, valores Q y K usados, el modelo activo, u_{bias} , y las variables intermedias (x_{ref}). Este registro permite analizar el desempeño, reproducir ensayos y comparar configuraciones de ganancia.

```
def Control_Lazo_Cerrado(
    estanque: Estanque,
    scheduler: GainScheduler,
    sp: float,
    gain_index: int = Esfuerzo,
    salida_fija: int = VALVULA_SALIDA_FIJA,
    segundos: int = Tiempo_Experimento_Segundos,
    u_bias_override: Optional[float] = None,
):
    total_steps = int(segundos / estanque.sample_time)

    # Modelo fijo según setpoint
    model = scheduler.pick(sp)

    lqr = ControladorLQR(gain_index=gain_index, u_min=UMIN, u_max=UMAX)

    estanque.escribir_valvula_salida(int(salida_fija))
    estanque.escribir_valvula_entrada(int(model.u0))

    # Esperar algunas muestras
    for _ in range(5):
        estanque.leer_sensores()
        time.sleep(estanque.sample_time)

    Q_sel, K_sel = model.gain_table[gain_index]

    print("Inicio del programa controlador LQR")
    print(
        f"Modelo={model.name} | "
        f"y0={model.y0:.2f} | "
```

```

    f"u0={model.u0:.1f} | "
    f"SP={sp} | "
    f"gain_index={gain_index} | "
    f"Q={Q_sel} | "
    f"K={K_sel}"
)

t0 = time.time()
estanque.data_log = []
u_prev = int(model.u0)
for _ in range(total_steps):
    t = time.time() - t0
    estanque.leer_sensores()
    y_meas = float(estanque.nivel)
    x_meas = (y_meas - float(model.y0)) / float(model.Cd)
    u_bias = float(u_bias_override) if u_bias_override is not None else
float(model.u0)
    u_lqr, err_y, K_used, Q_used = lqr.step(
        xhat=x_meas,
        sp=sp,
        model=model,
        u_bias=u_bias
    )
    # Limitador de pendiente
    DU_MAX = 25
    du = u_lqr - u_prev
    du = clamp(du, -DU_MAX, DU_MAX)
    u_cmd = u_prev + du
    u_cmd = int(clamp(u_cmd, UMIN, UMAX))
    u_prev = u_cmd

    estanque.escribir_valvula_entrada(u_cmd)

    dy_ref = float(sp) - float(model.y0)
    x_ref = dy_ref / float(model.Cd)
    du_lqr = -float(K_used) * (float(x_meas) - float(x_ref))

    # Log
    estanque.data_log.append({
        "time": round(t, 3),

```

```

    "nivel": float(y_meas),
    "nivel_control": float(y_meas),
    "sp": float(sp),
    "error_y": float(err_y),
    "u_lqr_raw": int(u_lqr),
    "u_cmd": int(u_cmd),
    "u_bias": float(u_bias),
    "du_lqr": float(du_lqr),
    "salida": int(salida_fija),
    "x_meas": float(x_meas),
    "x_ref": float(x_ref),
    "modelo": model.name,
    "K_lqr": float(K_used),
    "Q_lqr": float(Q_used),
    "gain_index": int(lqr.gain_index),
    "y0_usado": float(model.y0),
    "u0_usado": float(model.u0),
})

```

```

time.sleep(estanque.sample_time)
estanque.escribir_valvula_entrada(0)
estanque.escribir_valvula_salida(0)
print("Experimento terminado.")

```

3.1.5 Bucle principal

El bucle principal coordina la ejecución completa del experimento. Primero crea los objetos Estanque y GainScheduler, establece la conexión con Factory I/O y vacía el estanque. Luego selecciona el modelo correspondiente a la referencia y llama a la función Control_Lazo_Cerrado, que realiza todo el proceso de control en tiempo real. Al finalizar el ensayo, el programa guarda el archivo CSV con los datos registrados, genera la gráfica del experimento y cierra la conexión.

```

if __name__ == "__main__":
    estanque = Estanque()
    scheduler = GainScheduler(MODEL_TABLE)

```

```

try:
    estanque.conectar()
    print("Conectado. Lazo cerrado con LQR.")
    estanque.vaciar_estanque()
    model = scheduler.pick(sp)
    Control_Lazo_Cerrado(
        estanque=estanque,
        scheduler=scheduler,
        sp=sp,
        gain_index= Esfuerzo,
        salida_fija=VALVULA_SALIDA_FIJA,
        segundos=Tiempo_Experimento_Segundos,
    )
    stamp = time.strftime("%Y%m%d_%H%M%S")
    csv_name = f"closed_loop_{stamp}.csv"
    png_name = f"closed_loop_{stamp}.png"
    estanque.guardar(csv_name)
    estanque.plot(png_name)

except KeyboardInterrupt:
    print("Interrumpido por usuario.")
finally:
    estanque.desconectar()

```

3.2 **DEMOSTRACIÓN**

Una vez que el sistema de control estuvo completamente implementado y funcionando con el controlador LQR y el Gain Scheduling, fue necesario comprobar cómo se comportaba el controlador bajo condiciones más exigentes. Hasta este punto, la mayoría de las pruebas se habían hecho con señales constantes, por lo que no mostraban totalmente cómo reaccionaba el sistema ante cambios rápidos o inesperados.

Por esta razón se decidió usar una señal PRBS. Este tipo de señal es utilizada en control porque cambia de manera brusca y aleatoria, lo que obliga al sistema a

reaccionar constantemente. Gracias a esto es posible evaluar si el controlador realmente logra mantener el nivel del estanque estable.

En esta sección se presentan los resultados obtenidos con el sistema de control implementado. Para cada uno de los cuatro puntos de operación evaluados se realizó un ensayo utilizando una referencia fija y posteriormente, un segundo ensayo empleando una señal PRBS como referencia. Esta comparación permite observar el comportamiento del controlador ante una condición estable como su respuesta frente a cambios rápidos y continuos de la referencia.

Las figuras muestran el nivel, el setpoint, la acción de control y otras variables internas del controlador, mientras que las tablas resumen los principales parámetros y métricas de desempeño de cada ensayo. De esta forma, es posible evaluar la capacidad del controlador LQR con Gain Scheduling para mantener la estabilidad y realizar un seguimiento adecuado de la referencia en distintos puntos de operación.

3.2.1 Experimento 1

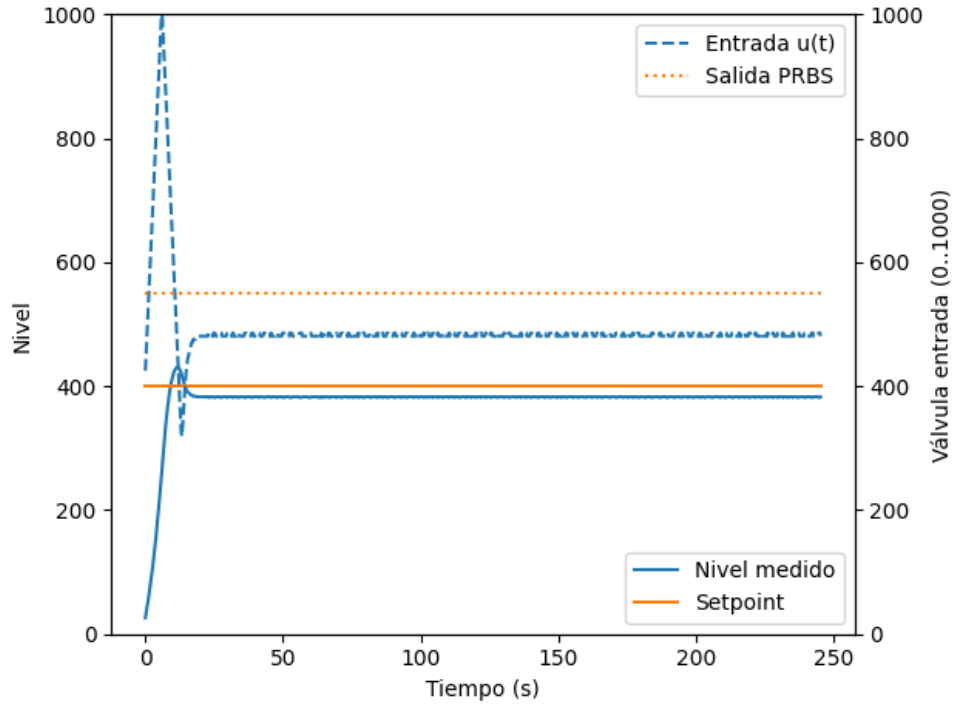
Para el primer experimento se seleccionó un nivel esperado de 400 unidades y el modelo correspondiente al intervalo de operación comprendido entre 350 y 450 unidades. Los parámetros utilizados durante este ensayo se presentan en la Tabla 3-1.

Tabla 3-1. Parámetros para Experimento 1.

Nivel Esperado	Modelo	Gain	Válvula de salida
400.0	Setpoint 350 - 450	1.422202999	550.0

Fuente: Elaboración propia.

Utilizando estos parámetros se realizó el ensayo inicial con una referencia constante. La respuesta obtenida se muestra en la Figura 3-1.



Fuente: Elaboración propia.

Figura 3-1. Experimento 1 de verificación

Los resultados obtenidos de esta respuesta se presentan en la Tabla 3-2.

Tabla 3-2. Resultados de Experimento 1

Overshoot	Tiempo de asentamiento	Error relativo
7.5%	16.35s	4.33%

Fuente: Elaboración propia.

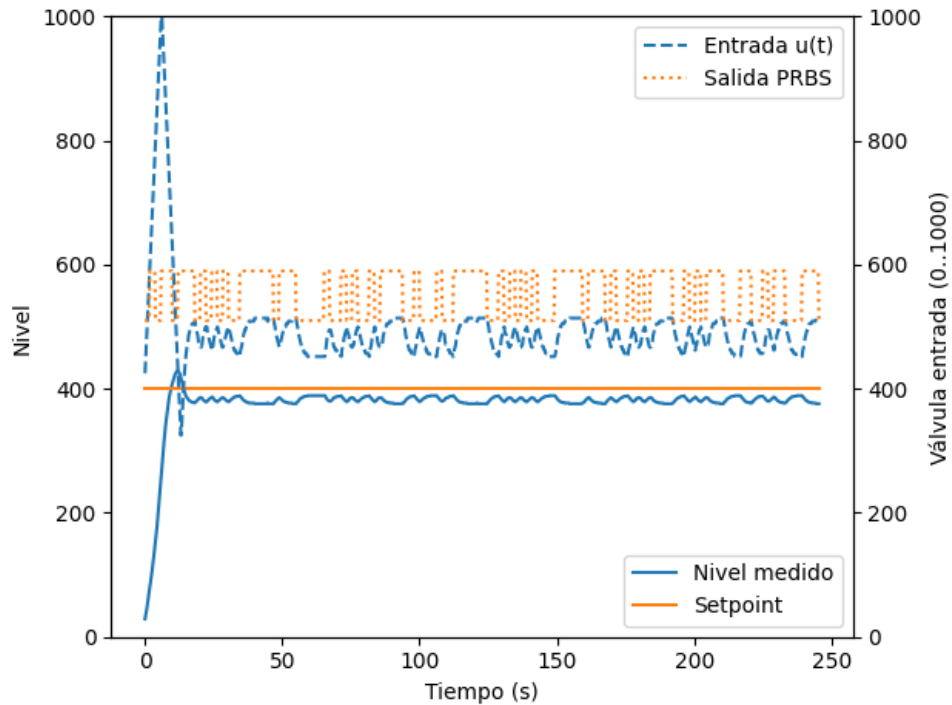
Posteriormente, se realizó un segundo ensayo para el mismo punto de operación utilizando una señal PRBS como referencia. Los parámetros utilizados para esta prueba se presentan en la Tabla 3-3.

Tabla 3-3. Parámetros para Experimento 1 con PRBS.

Nivel Esperado	Modelo	Gain	Válvula de salida
400.0	Setpoint 350 - 450	1.422202999	550.0 ± 40

Fuente: Elaboración propia.

La respuesta del sistema frente a la señal PRBS se presenta en la Figura 3-2, permitiendo observar el comportamiento del nivel ante las variaciones continuas de la referencia.



Fuente: Elaboración propia.

Figura 3-2. Experimento 1 con PRBS de verificación.

Finalmente, las métricas obtenidas durante el ensayo con PRBS se resumen en la Tabla 3-4.

Tabla 3-4. Resultados de Experimento 1 con PRBS

Overshoot	Tiempo de asentamiento	Error relativo
5.75%	15.60s	4.13%

Fuente: Elaboración propia.

Como se observa en la Figura 3-2 y en la Tabla 3-4, el sistema presentó un sobreimpulso de 5,75 % y un tiempo de asentamiento de 15,60 s. Además, el error relativo de 4,13 % indica que el controlador fue capaz de mantener el nivel próximo al valor deseado pese a las variaciones continuas de la referencia.

4.0.2 Experimento 2

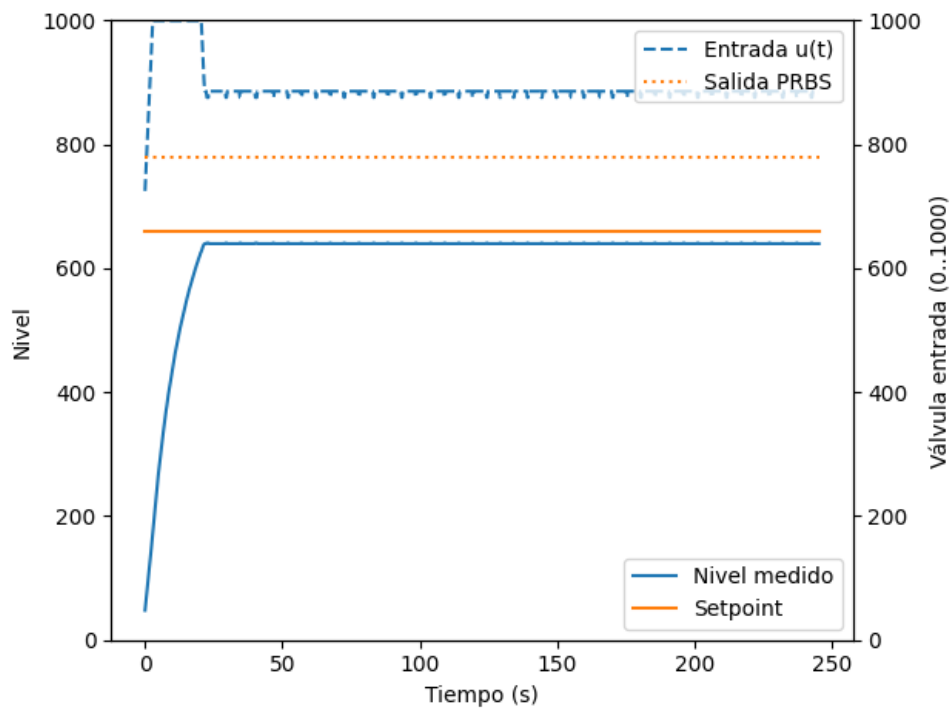
Para el segundo experimento se seleccionó un nivel esperado de 660 unidades, correspondiente al intervalo de operación comprendido entre 650 y 750 unidades. Los parámetros utilizados durante este ensayo se presentan en la Tabla 3-5.

Tabla 3-5. Parámetros para Experimento 2.

Nivel Esperado	Modelo	Gain	Válvula de salida
660.0	Setpoint 650 - 750	2.303310673	780.0

Fuente: Elaboración propia.

Utilizando estos parámetros se realizó el ensayo inicial con una referencia constante. La respuesta obtenida se muestra en la Figura 3-3.



Fuente: Elaboración propia.

Figura 3-3. Experimento 2 de verificación.

Los resultados obtenidos de esta respuesta se presentan en la Tabla 3-6.

Tabla 3-6. Resultados de Experimento 2

Overshoot	Tiempo de asentamiento	Error relativo
0%	20.98s	3.02%

Fuente: Elaboración propia.

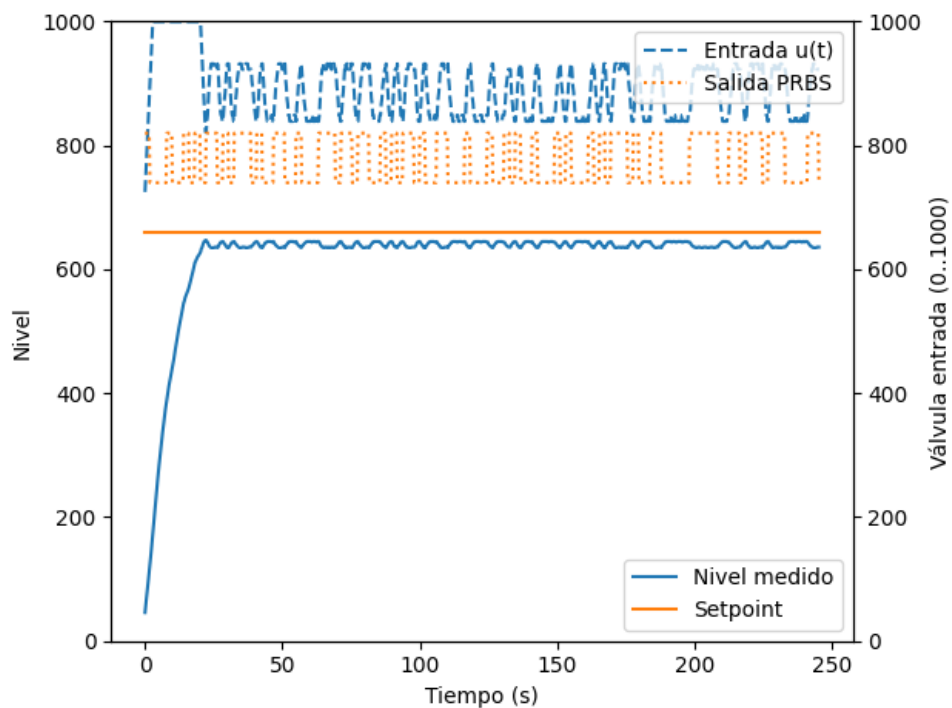
Posteriormente, el mismo punto de operación fue sometido a una referencia PRBS. Los parámetros correspondientes a esta prueba se presentan en la Tabla 3-7.

Tabla 3-7. Parámetros para Experimento 2 con PRBS.

Nivel Esperado	Modelo	Gain	Válvula de salida
660.0	Setpoint 650 - 750	2.303310673	780.0 ± 40

Fuente: Elaboración propia.

El comportamiento del sistema frente a estas variaciones se presenta en la Figura 3-4.



Fuente: Elaboración propia.

Figura 3-4. Experimento 2 con PRBS de verificación.

Los resultados cuantitativos de este ensayo se resumen en la Tabla 3-8.

Tabla 3-8. Resultados de Experimento 2 con PRBS

Overshoot	Tiempo de asentamiento	Error relativo
0%	22.76s	3.28%

Fuente: Elaboración propia.

Como se aprecia en la Figura 3-4 y en la Tabla 3-8, uno de los aspectos más relevantes fue la ausencia de sobreimpulso, lo que indica que el controlador alcanzó la referencia sin sobrepasar el valor deseado. El tiempo de asentamiento fue de 22,76 s y el error relativo alcanzó un 3,28 %, manteniendo una respuesta controlada durante el ensayo.

4.0.3 Experimento 3

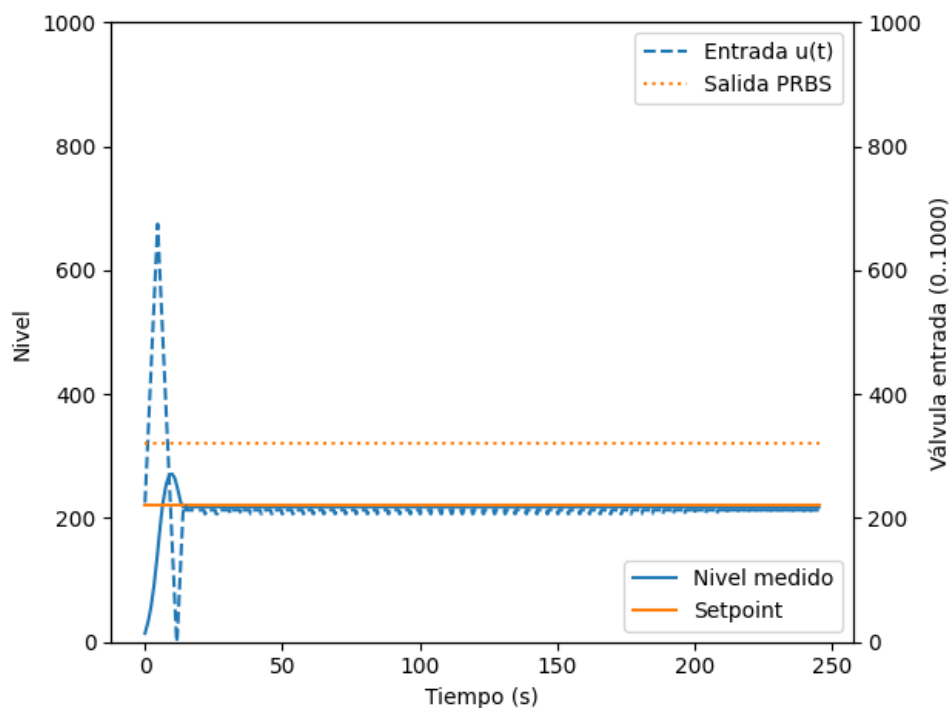
Para el tercer experimento se trabajó con un nivel esperado de 220 unidades, correspondiente al intervalo de operación comprendido entre 150 y 250 unidades. Los parámetros seleccionados para el ensayo se presentan en la Tabla 3-9.

Tabla 3-9. Parámetros para Experimento 3.

Nivel Esperado	Modelo	Gain	Válvula de salida
220.0	Setpoint 150 - 250	3.055022024	320.0

Fuente: Elaboración propia.

Utilizando estos parámetros se realizó el ensayo inicial con una referencia constante. La respuesta obtenida se muestra en la Figura 3-5.



Fuente: Elaboración propia.

Figura 3-5. Experimento 3 de verificación.

Las métricas calculadas para esta respuesta se muestran en la Tabla 3-10.

Tabla 3-10. Resultados de Experimento 3

Overshoot	Tiempo de asentamiento	Error relativo
23.18%	13.55s	0.86%

Fuente: Elaboración propia.

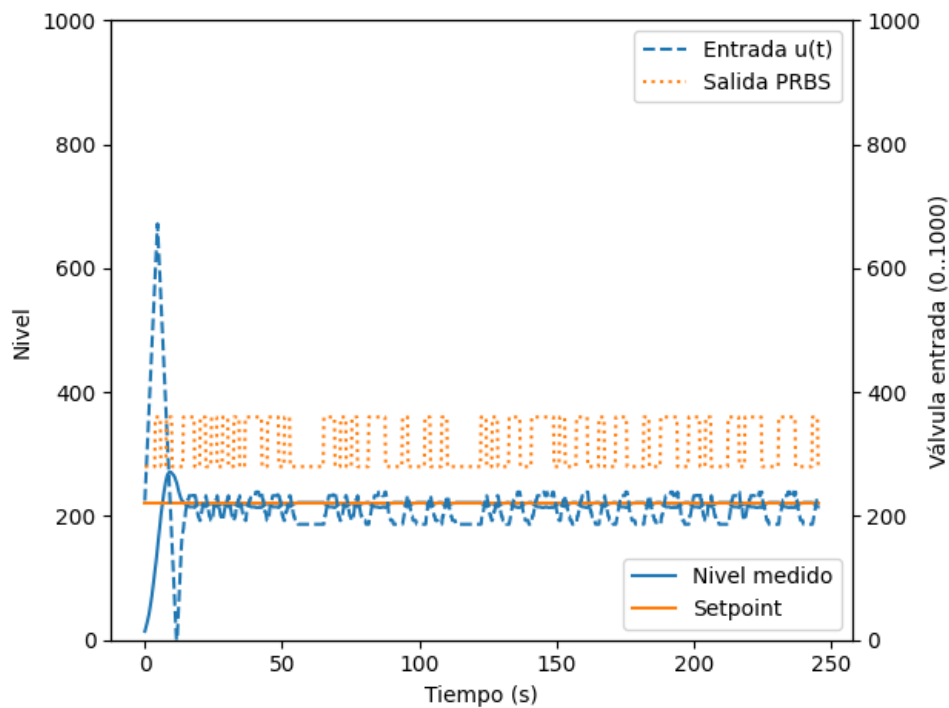
Posteriormente se aplicó la señal PRBS manteniendo el mismo punto de operación. Los parámetros utilizados durante esta prueba se presentan en la Tabla 3-11.

Tabla 3-11. Parámetros para Experimento 3 con PRBS.

Nivel Esperado	Modelo	Gain	Válvula de salida
220.0	Setpoint 150 - 250	3.055022024	320.0 ± 40

Fuente: Elaboración propia.

La respuesta registrada durante el ensayo se presenta en la Figura 3-6.



Fuente: Elaboración propia.

Figura 3-6. Experimento 3 con PRBS de verificación.

Los resultados obtenidos a partir de esta respuesta se resumen en la Tabla 3-12.

Tabla 3-12. Resultados de Experimento 3 con PRBS

Overshoot	Tiempo de asentamiento	Error relativo
23.18%	37.60s	0.11%

Fuente: Elaboración propia.

Debido a la dinámica del sistema en este rango de operación, se observó un overshoot de 23.18%, siendo el más alto de los experimentos realizados. A pesar de ello, el controlador consiguió estabilizar rápidamente el nivel y mantener un error relativo de 0.11%.

4.0.3 Experimento 4:

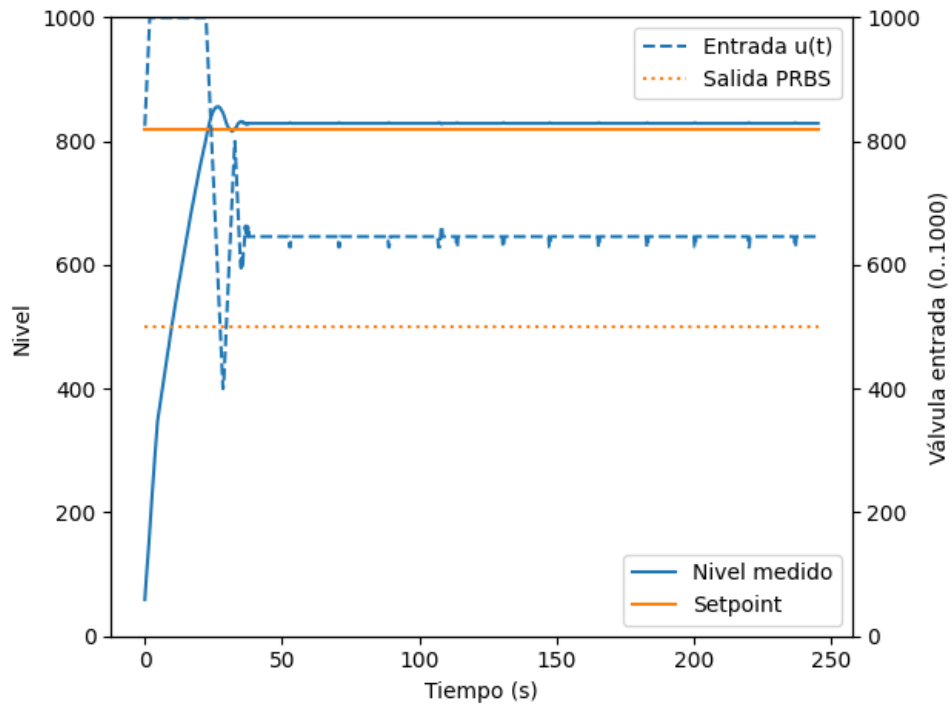
Para el cuarto experimento se seleccionó un nivel esperado de 820 unidades, correspondiente al intervalo comprendido entre 750 y 850 unidades. Los parámetros utilizados durante este ensayo se presentan en la Tabla 3-13.

Tabla 3-13. Parámetros para Experimento 4.

Nivel Esperado	Modelo	Gain	Válvula de salida
820.0	Setpoint 750 - 850	4.05035622	500.0

Fuente: Elaboración propia.

Utilizando estos parámetros se realizó el ensayo inicial con una referencia constante. La respuesta obtenida se muestra en la Figura 3-7.



Fuente: Elaboración propia.

Figura 3-7. Experimento 4 de verificación.

Las métricas calculadas para esta prueba se presentan en la Tabla 3-14.

Tabla 3-14. Resultados de Experimento 4

Overshoot	Tiempo de asentamiento	Error relativo
4.39%	30.77s	1.10%

Fuente: Elaboración propia.

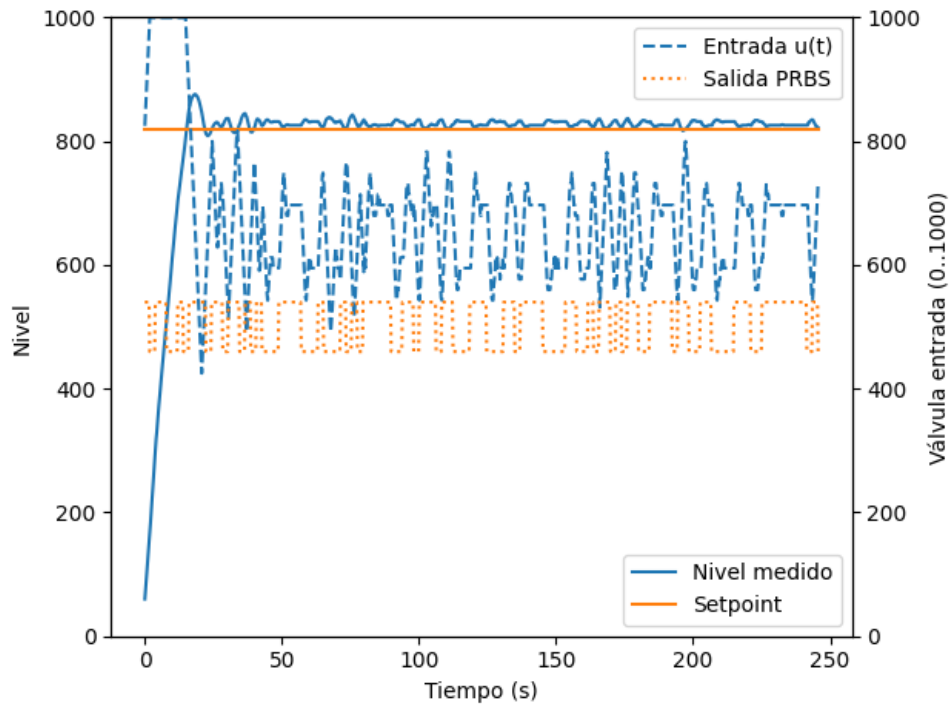
Posteriormente se realizó el ensayo utilizando una señal PRBS. Los parámetros utilizados se presentan en la Tabla 3-15.

Tabla 3-15. Parámetros para Experimento 4 con PRBS.

Nivel Esperado	Modelo	Gain	Válvula de salida
820.0	Setpoint 750 - 850	4.05035622	500.0 ± 40

Fuente: Elaboración propia.

El comportamiento del nivel durante esta prueba se presenta en la Figura 3-8.



Fuente: Elaboración propia.

Figura 3-8. Experimento 4 con PRBS de verificación.

Finalmente, las métricas obtenidas durante este experimento se resumen en la Tabla 3-16.

Tabla 3-16. Resultados de Experimento 4 con PRBS

Overshoot	Tiempo de asentamiento	Error relativo
6.83%	24.33s	1.05%

Fuente: Elaboración propia.

El controlador presentó una respuesta con una gran cantidad de oscilación por la ganancia del sistema, a pesar de ello logrando mantener el nivel cercano a la referencia durante todo el ensayo. Además, el sistema alcanzó estabilidad en 24.3 segundos, con un error relativo final de 1.05%, demostrando un comportamiento controlado y consistente en este rango de operación.

CONCLUSIONES

Los ensayos realizados con la señal PRBS permitieron evaluar al sistema en condiciones mucho más dinámicas que las pruebas anteriores, y los resultados fueron positivos. El controlador LQR logró mantener la estabilidad del nivel incluso cuando la señal PRBS generaba cambios muy rápidos. Se pudo observar que el sistema reaccionaba sin perder el control y sin presentar oscilaciones excesivas.

Además, el Gain Scheduling funcionó correctamente: en cada tramo operativo el sistema seleccionó el modelo adecuado y aplicó la ganancia correspondiente. Esto ayudó a que el error se mantuviera bajo, incluso cuando el estanque cambiaba de nivel rápidamente debido a la señal de prueba.

En resumen, las pruebas con PRBS demuestran que el sistema de control desarrollado es capaz de adaptarse a diferentes condiciones de operación y de responder de forma estable ante perturbaciones rápidas. Esto confirma que la metodología utilizada en el diseño y la identificación fue adecuada, y que el controlador se comporta de manera consistente con lo esperado.

BIBLIOGRAFÍA

COMPANION resources to "The Riccati Equation and its importance for LQR control" [en línea]. <https://resourcium.org/journey/companion-resources-riccati-equation-and-its-importance-lqr-control>. [Consulta: 31 mayo 2026].

CONTROL adaptativo y robusto [en línea]. <https://personal.us.es/rubio/ControlAdaptativoyRobusto.pdf>. [Consulta: 31 mayo 2026].

ORNELAS TÉLLEZ, Fernando. Control óptimo [en línea]. https://dep.fie.umich.mx/~fornelas/data/uploads/propiedades_lqr.pdf. [Consulta: 31 mayo 2026].

QUIROZ PIZARRO, Simón. Repositorio GitHub del proyecto de control LQR [en línea]. <https://github.com/SimonQuirozPizarro/Tesis-Controlador-LQR>. [Consulta: 31 mayo 2026].

WANG, Jian; LIU, Rui y BALDI, Pierre. Machine Learning and System Identification for Estimation in Physical Systems [en línea]. <https://arxiv.org/pdf/1906.02003>. [Consulta: 31 mayo 2026].