

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

CONTROL DE NIVEL DE LA PLATAFORMA RIEGEL UTILIZANDO ARDUINO R4

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

Alumno:

Bernardo Rojas Valdebenito

Profesor Guía:

Mag. Guelis Montenegro Zamora

RESUMEN

KEYWORDS: CONTROLADOR PID, PLATAFORMA RIEGEL, CONTROL DE NIVEL, ARDUINO, IOT, PID DISCRETO.

El presente trabajo describe el diseño, desarrollo e implementación de un controlador PID programado en Arduino UNO para el subsistema de nivel de la plataforma Riegel de la Universidad Técnica Federico Santa María. Los elementos más importantes que componen el sistema son una bomba centrífuga encargada de impulsar el agua de un estanque hacia otro, un transmisor de nivel con salida de 4-20 [mA], una placa Arduino UNO R4 WiFi y una interfaz en Arduino Cloud para monitorear y controlar el nivel de manera remota.

Para el desarrollo del proyecto, se debe establecer la estrategia de control a utilizar diferenciando entre un sistema en lazo abierto y lazo cerrado. Es necesaria la adaptación de los circuitos de nivel y de actuación de la bomba en base a las características técnicas de la placa Arduino UNO. Se realiza la obtención de datos iniciales para encontrar la función de transferencia del sistema y así poder establecer las variables que se usarán en el controlador PID en base a un criterio de sintonización determinado. Para el correcto funcionamiento se debe discretizar el controlador PID, entendiendo las limitaciones físicas de la placa de desarrollo y posteriormente realizar un código que represente esta discretización en forma correcta. Finalmente se realizan las pruebas del controlador con diferentes parámetros para el posterior análisis y conclusiones.

ÍNDICE

INTRODUCCIÓN	1
CAPÍTULO 1: ANTECEDENTES GENERALES	2
1.1. CONTEXTO Y PLANTEAMIENTO DEL PROBLEMA	3
1.2. PLATAFORMA RIEGEL	3
1.2.1. La marca	4
1.2.2. La estación de trabajo	4
1.2.3. La instrumentación de la plataforma	5
1.2.4. Diagrama P&ID de la estación Riegel	8
1.3. HARDWARE Y SOFTWARE DEL CONTROL	9
1.3.1. Arduino	9
1.3.2. Placas de desarrollo Arduino	10
1.3.3. MATLAB	12
1.3.4. Arduino IDE	13
1.3.5. Arduino Cloud	13
1.4. ESTRATEGIA DE CONTROL PID	14
1.5. OBJETIVOS	15
1.5.1. Objetivo general	15
1.5.2. Objetivos específicos	15
CAPÍTULO 2: DISEÑO Y DESARROLLO TÉCNICO	16
2.1. SISTEMAS DE CONTROL	17
2.1.1. Sistema de control en lazo abierto	17
2.1.2. Sistema de control en lazo cerrado	18
2.2. DIAGRAMA DE BLOQUES DEL SISTEMA DE CONTROL DE NIVEL	18
2.3. CIRCUITO ADAPTADOR	19
2.3.1. Circuito adaptador sensor de nivel	19

2.3.2.	Circuito adaptador bomba centrífuga	21
2.4.	OBTENCIÓN DE DATOS Y FUNCIÓN DE TRANSFERENCIA	22
2.4.1.	Escalamiento del sensor de nivel.....	23
2.4.2.	Obtención de datos en lazo abierto.....	25
2.4.3.	Punto de operación y función de transferencia	28
2.5.	DISEÑO DEL CONTROLADOR PID.....	31
2.5.1.	Criterios de sintonización.....	31
2.5.2.	Discretización del controlador PID.....	32
2.5.3.	Código controlador PID.....	33
2.5.4.	Funcionamiento con Arduino Cloud	39
CAPÍTULO 3: RESULTADOS Y ANÁLISIS		43
3.1.	PRUEBAS DEL CONTROLADOR PI.....	44
3.1.1.	Respuesta del controlador PI abriendo válvula FV 03	45
3.1.2.	Respuesta del controlador PI cerrando válvula FV 02	46
3.1.3.	Respuesta del controlador PI con Kp alto	48
3.1.4.	Respuesta del controlador PI con Ki bajo	50
3.1.5.	Respuesta del controlador PI sin acción integral.....	52
3.1.6.	Respuesta del controlador PI sin anti-windup.....	54
3.2.	FUNCIONAMIENTO CON ARDUINO CLOUD.....	55
3.3.	OBSERVACIONES DE LOS RESULTADOS OBTENIDOS.....	57
CONCLUSIONES Y RECOMENDACIONES		58
BIBLIOGRAFÍA		59
ANEXOS		61
ANEXO A: CÓDIGO DE ARDUINO PARA RECOLECCIÓN DE DATOS.....		62
ANEXO B: CÓDIGO DE ARDUINO PARA CONTROLADOR PID.....		63
ANEXO C: CÓDIGO DE ARDUINO CLOUD PARA CONTROLADOR PID		65

ÍNDICE DE FIGURAS

Figura 1-1. Logotipo Riegel.....	4
Figura 1-2. Logotipo DOS20.....	4
Figura 1-3. Estación de trabajo Riegel.....	5
Figura 1-4. Bomba centrífuga SPX CM30.	7
Figura 1-5. Esquema eléctrico bomba SPX CM30.	7
Figura 1-6. Transmisor piezoresistivo de nivel Keller 26Y.....	8
Figura 1-7. Diagrama P&ID de la estación de trabajo Riegel.	9
Figura 1-8. Logotipo Arduino.....	10
Figura 1-9. Placa Arduino UNO R4 WiFi.	11
Figura 1-10. Logotipo MATLAB.....	12
Figura 1-11. Pantalla inicial de Arduino IDE.	13
Figura 1-12. Logotipo Arduino Cloud.	14
Figura 2-1. Sistema de control en lazo abierto.	17
Figura 2-2. Sistema de control en lazo cerrado.	18
Figura 2-3. Diagrama de bloques sistema control de nivel.....	19
Figura 2-4. Circuito adaptador sensor de nivel.	20
Figura 2-5. Circuito adaptador bomba centrífuga.....	22
Figura 2-6. Gráfico nivel digital/nivel de agua.	24
Figura 2-7. Señal filtrada con Alpha 0.6.	25
Figura 2-8. Señal filtrada con Alpha 0.2.	26
Figura 2-9. Señal filtrada con Alpha 0.05.	26
Figura 2-10. Código de Arduino IDE para obtención de datos.....	27
Figura 2-11. Señales con y sin filtro EMA en sistema de nivel.	27
Figura 2-12. Gráficos de actuación y respuesta del sistema de nivel.	28
Figura 2-13. Identificación de función de transferencia.	29
Figura 2-14. Constantes y variables del programa.....	34
Figura 2-15. Configuración inicial del programa.	35

Figura 2-16. Bucle del programa.	35
Figura 2-17. Lectura nivel de agua del bucle.....	36
Figura 2-18. Primera condición del bucle.....	36
Figura 2-19. Segunda condición del bucle y anti-windup.	38
Figura 2-20. Salida y tiempo de muestreo del bucle.....	38
Figura 2-21. Instalación completada de placa.	39
Figura 2-22. Pantalla "Things" en Arduino Cloud.....	40
Figura 2-23. Dashboard con widgets asociados a variables.....	41
Figura 2-24. Uso de función para ingresar setpoint.....	42
Figura 2-25. Bucle del Sketch en Arduino Cloud.	42
Figura 3-1. Prueba con valores teóricos y abriendo válvula FV 03 al 50%.....	45
Figura 3-2. Prueba con valores teóricos y cerrando válvula FV 02 al 25%.....	47
Figura 3-3. Prueba con $K_p=10$	49
Figura 3-4. Prueba con $K_i=0.1$	51
Figura 3-5. Prueba sin acción integral.	53
Figura 3-6. Prueba sin anti-windup.	54
Figura 3-7. Dashboard de Arduino Cloud en funcionamiento.	55
Figura 3-8. Visualización de Dashboard desde smartphone.	56

ÍNDICE DE TABLAS

Tabla 2-1. Resumen datasheet transistor TIP122.	21
Tabla 2-2. Resumen datasheet AO LM358P.....	22
Tabla 2-3. Datos experimentales de sensor de nivel.	23
Tabla 2-4. Datos experimentales de sensor de nivel con bit.	24
Tabla 2-5. Resumen de datos de actuación y respuesta.....	29
Tabla 2-6. Constantes para criterio de Ziegler and Nichols.	32

SIGLA Y SIMBOLOGÍA

A. SIGLA:

BIBO	:	Bounded Input, Bounded Output.
DAC	:	Digital to Analog Converter.
DIN	:	Deutsches Institut für Normung.
EMA	:	Exponential Moving Average.
GPIO	:	General Purpose Input/Output.
IDE	:	Integrated Development Environment.
IoT	:	Internet of Things.
IP	:	Índice de Protección.
P&ID	:	Piping and Instrumentation Diagram/Drawing.
PI	:	Control Proporcional Integral.
PID	:	Control Proporcional Integral Derivativo.
PWM	:	Pulse Width Modulation.
USB	:	Universal Serial Bus.
WiFi	:	Wireless Fidelity.

B. SIMBOLOGÍA:

A	:	Amperio.
bar	:	Bar.
C	:	Capacitancia.
cm	:	Centímetro.
Hz	:	Hertz.
kB	:	Kilobyte.
$k\Omega$	:	KiloOhm.
l/min	:	Litros por minuto.

m	:	Metro(s).
mA	:	Miliamperio.
mbar	:	Milibar.
MHz	:	Mega Hertz.
μF	:	Micro Faradio.
R	:	Resistencia.
V	:	Volts.
Vcc	:	Volts en corriente continua.
W	:	Watts.
Δ	:	Delta.
°	:	Grado(s) angular(es).
°C	:	Grado(s) Celsius.
Ω	:	Ohm.
%	:	Por ciento.

INTRODUCCIÓN

En el ámbito de la industria productiva, el control y la automatización es fundamental para desarrollar sistemas que permitan regular un sinnúmero de variables dentro de un proceso. Variables como presión, flujo, temperatura y nivel son las más comunes y dentro de ellas, el nivel de líquidos en depósitos es de las más críticas por las aplicaciones relacionadas. Tratamiento de aguas, procesos químicos, sistemas de enfriamiento o almacenamiento de fluidos son de los usos más populares y obligados dentro de la industria. Mantener el nivel de cierto líquido dentro de un rango determinado, permite asegurar la eficiencia y calidad de un proceso e incluso puede prevenir riesgos como el desbordamiento del fluido almacenado, entre otros.

El presente trabajo se centra en el diseño, adaptación e implementación de un sistema automático de control de nivel en la plataforma Riegel de la Universidad Técnica Federico Santa María, empleando un controlador PI programado en la placa Arduino UNO R4 WiFi. El sistema está compuesto por un transmisor de nivel con salida de 4-20 [mA], una bomba centrífuga controlada desde la placa de Arduino y una interfaz gráfica de monitoreo desarrollada en la plataforma IoT Arduino Cloud, para la visualización y operación en tiempo real.

El trabajo está organizado en tres capítulos: el primero, establece los antecedentes generales del proyecto; el segundo, tiene por finalidad dar a conocer el diseño y desarrollo técnico, mostrando paso a paso la obtención del controlador; el tercero, presenta los resultados obtenidos por medio de diferentes pruebas realizadas, mostrando las diferencias en el comportamiento del sistema.

CAPÍTULO 1: ANTECEDENTES GENERALES

1. ANTECEDENTES GENERALES

Este capítulo tiene por finalidad informar el desafío a resolver por este proyecto y la estrategia que se utilizará para su resolución. Además, se dará a conocer el hardware y software implicado en la solución, la instrumentación de la plataforma donde se trabajará y los objetivos generales y específicos.

1.1. CONTEXTO Y PLANTEAMIENTO DEL PROBLEMA

La plataforma multivariable Riegel en el laboratorio de control de procesos de la universidad permite realizar pruebas prácticas en variables físicas como nivel, presión y flujo. Si bien su operación puede ser manual, se requiere de un sistema autónomo que permita regular el nivel de agua.

En ese contexto, se propuso como desafío la implementación de un sistema de control PID basado en Arduino UNO R4 WiFi, complementado con su visualización remota a través de alguna plataforma IoT. Este desafío incluye:

- Diseño, desarrollo e implementación física de un controlador PID.
- Integración a una plataforma IoT para su visualización en línea.
- Validación del sistema frente a condiciones reales de operación.

1.2. PLATAFORMA RIEGEL

Para conocer la plataforma Riegel, se dará un contexto general acerca de las marcas implicadas en su creación, así como también de la estación de trabajo misma instalada dentro de las dependencias de la universidad.

1.2.1. La marca

Riegel (figura 1-1) es una empresa multidisciplinaria con más de 15 años de experiencia en la gestión de recursos hídricos y de fluidos, dedicada al desarrollo electrónico, diseño y fabricación de equipos de telemetría y automatización. Junto con la empresa DOS20 (figura 1-2), trabajan en conjunto para desarrollar e implementar soluciones tecnológicas y mejorar la gestión de recursos hídricos y fluidos en general.



Fuente: <https://dos20.cl/wp-content/uploads/2021/02/Logo-Riegel.png>

Figura 1-1. Logotipo Riegel.



Fuente: <https://dos20.cl/wp-content/uploads/2021/02/Logo-Dos20.png>

Figura 1-2. Logotipo DOS20.

1.2.2. La estación de trabajo

Dentro de la Universidad Técnica Federico Santa María, específicamente en el edificio M, sala 203, se encuentra el laboratorio de control de procesos, ahí está la plataforma multivariable Riegel.

Esta estación de trabajo es una plataforma de experimentación donde se pueden controlar cuatro variables: nivel, flujo, presión y temperatura. En la figura 1-3 se muestra la estación de trabajo utilizada en el proyecto.



Fuente: Elaboración propia. Sala de control de procesos, edificio M.

Figura 1-3. Estación de trabajo Riegel.

1.2.3. La instrumentación de la plataforma

Para medir cada una de las señales de las variables mencionadas anteriormente, la plataforma posee los siguientes sensores:

- Un flujómetro MAGX2 de la marca Arkon.
- Un manómetro de la marca Winters.
- Dos transmisores piezoresistivos de nivel low-cost marca Keller, serie 26Y.
- Un transmisor de presión de alta precisión marca Keller, serie 33X.
- Termocupla tipo K.

Por otra parte, cuenta con los siguientes actuadores:

- Bomba de circulación de 24 [Vcc] marca SPX, modelo CM30.
- Actuador eléctrico VALPES, modelo ER Plus.
- Calefactor.
- 5 válvulas manuales.

Como el proyecto consiste en el control de nivel de la plataforma, se hará la descripción solamente de los instrumentos que participan en dicho control.

1.2.3.1. Bomba de circulación SPX, modelo CM30

Las bombas CM son centrífugas de aspiración normal y se deben montar de forma que reciban agua o que se ceben antes de la puesta en marcha. No deben funcionar en seco, aunque durante algún tiempo pueden soportar este tipo de funcionamiento (no superior a 30 minutos).

Los motores son de funcionamiento continuo y permite una variación de $\pm 20\%$. El sobrevoltaje reduce la vida de la bomba.

La figura 1-4 muestra la bomba utilizada en la plataforma Riegel, mientras que en la figura 1-5 se aprecia la conexión eléctrica para su funcionamiento.

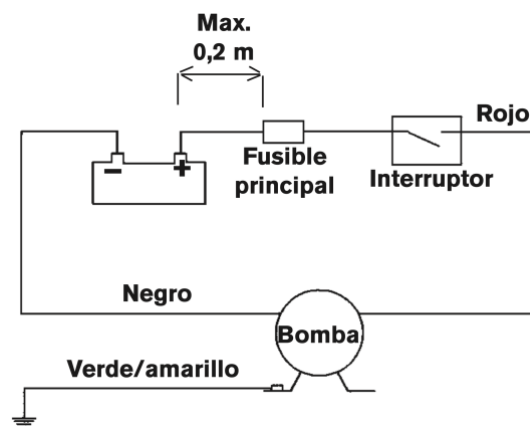
Datos técnicos del motor:

- Motor de imán permanente, 24 [Vcc].
- Potencia: 26 [W].
- Corriente: 0.8 [A].
- Temperatura líquido: $-40^{\circ} - +100^{\circ}\text{C}$.
- Flujo: 22.5 [l/min].
- Presión máxima: 2.5 bar.
- Protección IP67 (DIN40050).



Fuente: Manual SPX para bomba modelo CM10/30.

Figura 1-4. Bomba centrífuga SPX CM30.



Fuente: Manual SPX para bomba modelo CM10/30.

Figura 1-5. Esquema eléctrico bomba SPX CM30.

1.2.3.2. Transmisor de nivel Keller, serie 26Y

Los transmisores de nivel de la serie 26Y se utilizan para medidas de nivel de agua en las que se requiera un buen nivel de precisión. Estos transmisores tienen un error de temperatura muy bajo, lo que se consigue gracias a la compensación digital de una señal completamente analógica.

La figura 1-6 muestra el transmisor de nivel utilizado en la plataforma Riegel.

Características de funcionamiento:

- Muy alta precisión, extraordinaria estabilidad a largo plazo, sin histéresis de presión.
- Protección integrada contra sobretensión y polaridad inversa.
- Protección IP68.
- Rango de presión de 100 mbar hasta 10 bar.
- Compacto, carcasa robusta hecha de acero inoxidable.
- Corriente de salida: 4-20 [mA].

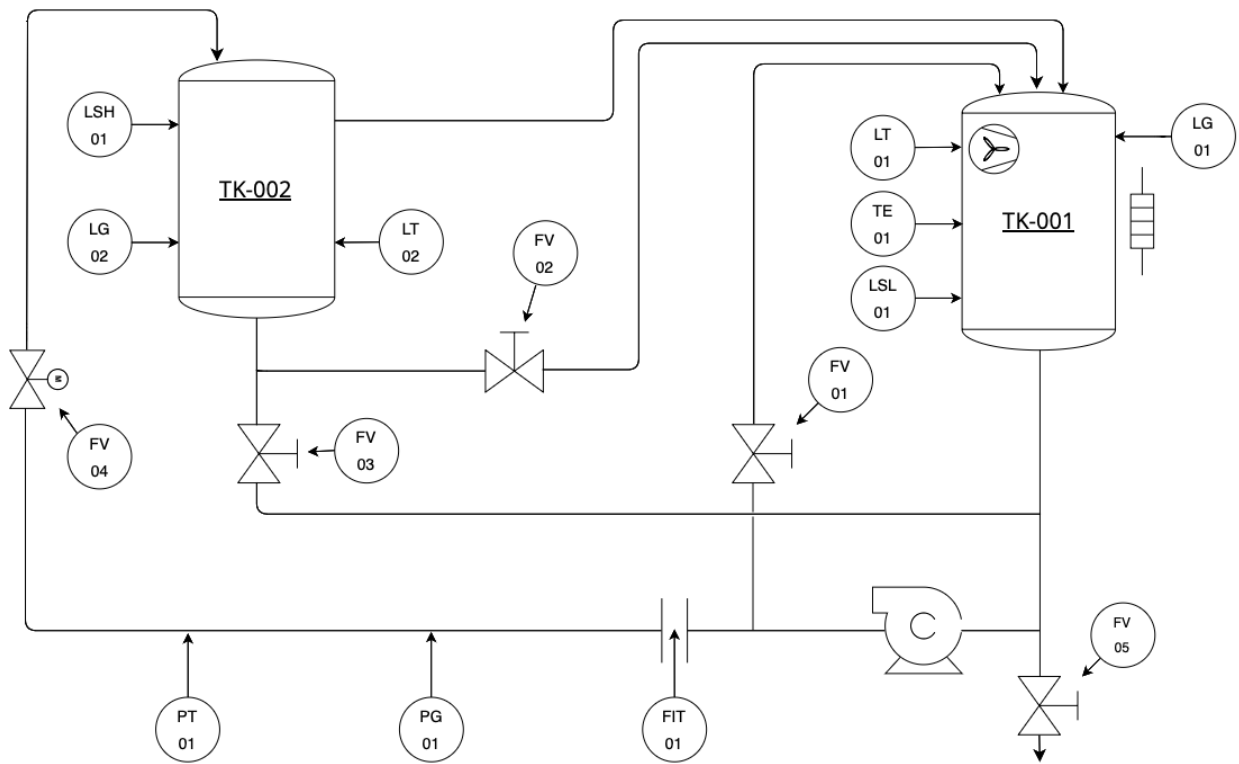


Fuente: Manual Keller para transmisor piezoresistivo de nivel 36Y.

Figura 1-6. Transmisor piezoresistivo de nivel Keller 26Y.

1.2.4. Diagrama P&ID de la estación Riegel

En la figura 1-7, se muestra el diagrama P&ID de la estación Riegel, con todos los elementos actuadores (válvulas, bomba, etc.) y de control utilizados (sensores, transmisores, etc.).



Fuente: Elaboración propia en web de visual-paradigm. Basado en estación Riegel, sala control de procesos.

Figura 1-7. Diagrama P&ID de la estación de trabajo Riegel.

1.3. HARDWARE Y SOFTWARE DEL CONTROL

En este apartado se dará a conocer el hardware y software utilizado para el diseño, desarrollo e implementación del controlador PID.

1.3.1. Arduino

Arduino (logotipo en la figura 1-8) es una plataforma de electrónica de código abierto, basada en hardware y software fáciles de usar. Sus placas de desarrollo son capaces de leer entradas y convertirlas en una salida. Para hacer todo esto, utiliza el lenguaje de programación Arduino (basado en Wiring) y el software Arduino (IDE), basado en Processing.

Arduino nació en el Ivrea Interaction Design Institute como una herramienta fácil para la creación rápida de prototipos, para estudiantes sin experiencia en electrónica y programación.

Las ventajas más notables de Arduino son:

- Económico.
- Multiplataforma.
- Entorno de programación simple y claro.
- Software de código abierto y extensible.
- Hardware de código abierto y extensible.



Fuente: <https://www.arduino.cc>.

Figura 1-8. Logotipo Arduino.

1.3.2. Placas de desarrollo Arduino

Arduino a través de los años ha lanzado numerosos productos de hardware: placas, shields, kits y otros. Sus productos más reconocidos son:

- Familia NANO: Son placas de desarrollo de pequeño tamaño, que van desde características básicas hasta módulos con Bluetooth y WiFi.
- Familia MKR: Son una serie de placas, shields y soportes que se pueden combinar para la creación de proyectos, sin ningún circuito adicional.
- Familia MEGA: Placas para proyectos que exigen mucha potencia informática y pines GPIO.

- Familia Classic: Son placas consideradas clásicas y la columna vertebral de la marca. Placas como Arduino UNO, Arduino Leonardo o Arduino Micro componen esta familia.

1.3.2.1. Arduino UNO R4 WiFi

La placa Arduino UNO R4 WiFi (mostrada en la figura 1-9) es la primera placa UNO que cuenta con un microcontrolador de 32 bits y un módulo ESP32-S3 WiFi (ESP32-S3-MINI-1-N8). Cuenta con un microcontrolador de la serie RA4M1 de Renesas (R7FA4M1AB3CFM#AA0), basado en un microprocesador Arm Cortex-M4 de 48 MHz. La memoria del UNO R4 WiFi es más grande que la de sus predecesores, con 256 kB de flash, 32 kB de SRAM y 8 kB de EEPROM.

Especificaciones técnicas:

- Voltaje de funcionamiento del circuito: 5 [V] (ESP32-S3 es 3.3 [V]).
- Voltaje de entrada (VIN): 6-24 [V].
- Corriente de CC por pin E/S: 8 [mA].
- Puerto de programación USB-C.
- 14 pines de E/S digitales.
- 6 pines de entrada analógicos.
- 1 pin DAC (salida analógica) de 12 bits.
- 6 pines PWM.



Fuente: <https://store.arduino.cc/>.

Figura 1-9. Placa Arduino UNO R4 WiFi.

1.3.3. MATLAB

MATLAB (logotipo en la figura 1-10) es una plataforma de programación y cálculo numérico usada para el análisis de datos, desarrollo de algoritmos y creación de modelos.

Este software combina el análisis y los procesos de diseño con un lenguaje de programación que expresa las matemáticas directamente.

Las prestaciones de MATLAB son comúnmente el análisis de datos, realización de gráficas, programación de scripts y funciones, creación de apps y cálculo en la nube. Entre sus usos se encuentra:

- Sistemas de control.
- Deep learning.
- Procesamiento de imágenes y visión artificial.
- Machine learning.
- Mantenimiento predictivo.
- Robótica.
- Procesamiento de señales.
- Prueba y medición.
- Comunicaciones inalámbricas.



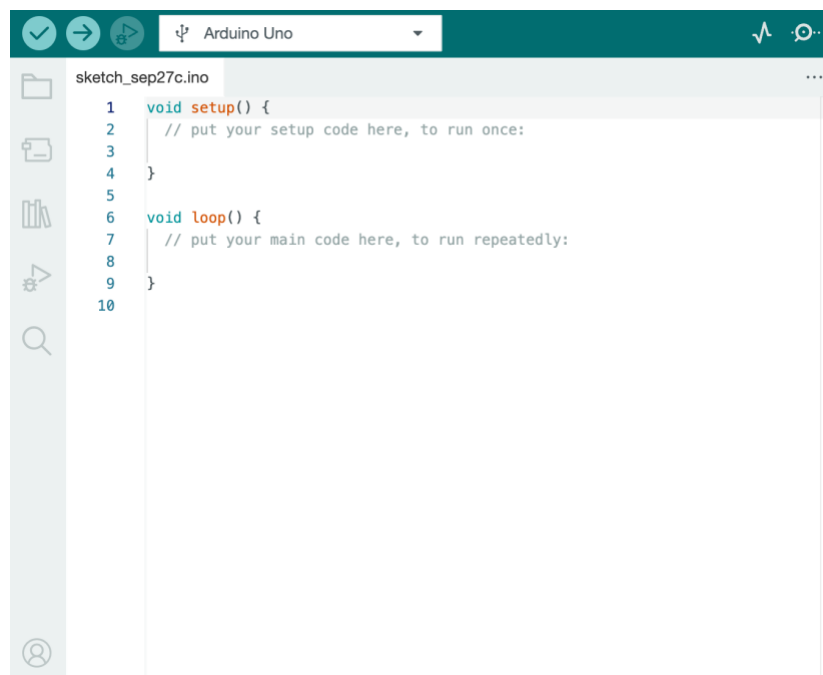
Fuente: <https://la.mathworks.com/products/matlab.html>.

Figura 1-10. Logotipo MATLAB.

1.3.4. Arduino IDE

El IDE es un conjunto de herramientas de software que permiten desarrollar y grabar los códigos de programación. Para empezar a programar Arduino, es necesario descargar un IDE. Este permite escribir, depurar, editar y grabar el programa a realizar (llamados “sketches” en el entorno de Arduino) de una manera sencilla.

En la figura 1-11 se muestra la pantalla inicial de trabajo del software.



Fuente: Elaboración propia. Captura de pantalla desde programa Arduino IDE.

Figura 1-11. Pantalla inicial de Arduino IDE.

1.3.5. Arduino Cloud

Arduino Cloud (logotipo en la figura 1-12) es una plataforma IoT (“Internet de las cosas”, del inglés “Internet of Things”) que sirve para crear, monitorear y desplegar proyectos, controlando dispositivos a distancia, visualizando y compartiendo datos desde placas de desarrollo compatibles. En particular, se utilizará la versión gratuita, perfecta para el propósito

de este proyecto, dejando agregar dos “Things” (número de objetos que cada usuario puede conectar con la placa a la vez), cinco variables y retención de datos en la nube de un día.

Por otro lado, existen diferentes versiones de pago (Maker, Team, Custom), diferenciándose en la cantidad de Things permitidas (25, 100 e ilimitadas respectivamente), variables ilimitadas para las tres versiones y retención de datos de tres meses, un año y dos años respectivamente.



Fuente: <https://app.arduino.cc>.

Figura 1-12. Logotipo Arduino Cloud.

1.4. ESTRATEGIA DE CONTROL PID

Para el control del nivel de líquido en la plataforma, se usará una estrategia de control PID. Esta estrategia basta para solucionar problemas de control de variables industriales como nivel, presión, temperatura, flujo, etc.

Al combinar la acción de control proporcional, la acción de control integral y la acción de control derivativa, se obtiene la acción de control proporcional-integral-derivativa (PID) y tiene las ventajas de cada una de las tres acciones de control individuales. La estructura de un PID paralelo está dada por la siguiente ecuación:

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt}$$

Donde K_p es la ganancia proporcional; K_i es la ganancia integral igual a $\frac{K_p}{T_i}$, siendo T_i el tiempo integral; K_d es la ganancia derivativa igual a $K_p T_d$, siendo T_d el tiempo derivativo.

La acción proporcional provee una acción que es proporcional al error de control para pequeños errores, intentando aproximar el error a cero; la acción integral provee una acción proporcional al error acumulado en el tiempo. Fuerza el error en estado estacionario a cero en presencia de referencia escalón y perturbaciones; y la acción derivativa provee una acción proporcional a la razón de cambio en el error, su función es reducir las oscilaciones causadas por perturbaciones en los procesos.

1.5. OBJETIVOS

A continuación, se hará descripción de los objetivos, tanto general como específicos, que se deben lograr durante la ejecución del proyecto.

1.5.1. Objetivo general

Implementar un controlador PID para regular la variable de nivel de la plataforma Riegel, utilizando Arduino UNO R4 WiFi.

1.5.2. Objetivos específicos

- Estudiar la plataforma Riegel y la instrumentación que posee.
- Diseñar un circuito de adaptación para trabajar con la bomba.
- Programar el controlador PID con Arduino UNO R4 WiFi.
- Establecer la comunicación entre un dashboard y Arduino UNO R4 WiFi.
- Configurar el dashboard para graficar actuación y nivel.
- Obtener resultados prácticos de la implementación del proyecto.

CAPÍTULO 2: DISEÑO Y DESARROLLO TÉCNICO

2. DISEÑO Y DESARROLLO TÉCNICO

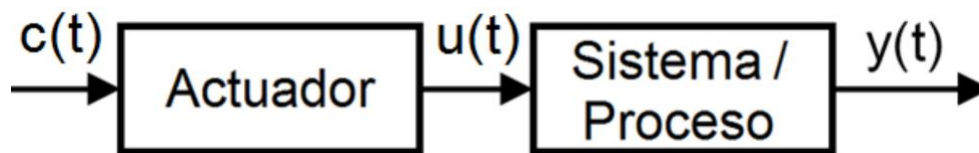
Este capítulo tiene por finalidad dar a conocer el diseño y desarrollo del control PID a implementar en la placa Arduino UNO R4 WiFi, para el control automático de nivel de agua en la plataforma Riegel. Se mostrarán los aspectos teóricos y prácticos para la realización de un sistema eficiente de control, desde la selección de los componentes hasta la programación del controlador en Arduino IDE y la visualización en Arduino Cloud.

2.1. SISTEMAS DE CONTROL

Existen diversas formas para ejecutar el control de nivel de un estanque de agua, en particular de la plataforma Riegel. Se puede mencionar el control en lazo abierto y el control en lazo cerrado, que se detallan a continuación.

2.1.1. Sistema de control en lazo abierto

Los sistemas en los que la acción de control no se ve afectada por la salida se denominan “sistemas de control en lazo abierto”. En este tipo de sistemas, aunque la salida puede ser medida y/o monitoreada, dicha información no se utiliza para retroalimentar el sistema ni para compararla con la entrada, por lo tanto, a cada entrada de referencia le corresponde una condición operativa fija. Es por esto que la precisión del sistema depende de la calibración y ante la presencia de perturbaciones, este sistema no realiza un control correcto. En la figura 2-1 se muestra el diagrama de bloques de un sistema de control en lazo abierto.



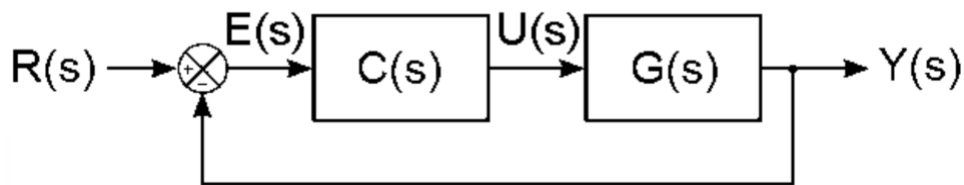
Fuente: PPT “INTRODUCCIÓN”, asignatura Control de Procesos, profesor Guelis Montenegro Z.

Figura 2-1. Sistema de control en lazo abierto.

Siendo $c(t)$ la señal de control, $u(t)$ la variable manipulada e $y(t)$ la respuesta o variable de proceso.

2.1.2. Sistema de control en lazo cerrado

Son sistemas en donde se alimenta al controlador con la señal de error de la actuación, que es la diferencia entre la señal de entrada y la de salida; es decir, existe una retroalimentación con el fin de reducir el error y llevar la salida del sistema a un valor deseado. El uso de un sistema de control en lazo cerrado trabaja bien en presencia de perturbaciones, por lo que es adecuado para realizar el control de una planta determinada. En la figura 2-2 se muestra el diagrama de bloques de un sistema de control en lazo cerrado.



Fuente: PPT "INTRODUCCIÓN", asignatura Control de Procesos, profesor Guelis Montenegro Z.

Figura 2-2. Sistema de control en lazo cerrado.

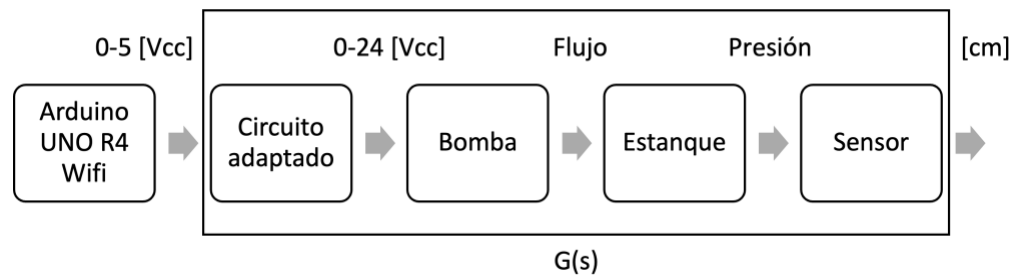
Siendo $R(s)$ la referencia o setpoint, $E(s)$ el error o diferencia entre la referencia y la respuesta del sistema, $U(s)$ la actuación o variable manipulada, $G(s)$ es la planta, proceso o sistema e $Y(s)$ la respuesta o variable de proceso.

2.2. DIAGRAMA DE BLOQUES DEL SISTEMA DE CONTROL DE NIVEL

Para poder realizar el análisis en lazo abierto del control de nivel, es necesario conocer las partes que lo componen. En primer lugar, la placa de desarrollo Arduino UNO R4 WiFi puede entregar un máximo de 5 [Vcc] con 8 [mA] por pin, por lo que se debe adaptar un circuito para el funcionamiento de la bomba de 24 [Vcc]. Luego del funcionamiento de la bomba, se produce un

flujo de agua hacia el estanque, que a su vez provoca presión sobre el sensor. Este sensor envía una señal de 4-20 [mA] que no puede entrar directamente a la placa por las limitaciones de hardware antes mencionadas, por lo que se debe usar un resistor para convertir la corriente en una tensión segura y legible por la entrada analógica de Arduino UNO.

De esta manera, el diagrama de bloques del sistema de control de nivel tiene la estructura mostrada en la figura 2-3.



Fuente: Elaboración propia. Basada en análisis propio de funcionamiento del sistema de control de nivel.

Figura 2-3. Diagrama de bloques sistema control de nivel.

2.3. CIRCUITO ADAPTADOR

Para el funcionamiento del sistema de control, es necesario adaptar un circuito para la lectura del nivel del estanque (figura 2-4) y para el funcionamiento de la bomba (figura 2-5). Estos circuitos no deben ser modificados, ya que pueden alterar la función de transferencia del sistema en lazo abierto.

2.3.1. Circuito adaptador sensor de nivel

El sensor de nivel del estanque entrega una señal de corriente de 4-20 [mA], mientras que el Arduino debe recibir un voltaje de 0-5 [Vcc] en sus entradas analógicas. Por medio de la ley de Ohm, se calcula la resistencia para obtener un voltaje similar al de la entrada analógica de Arduino UNO.

$$R = \frac{5 [V]}{0.02 [A]} = 250 [\Omega]$$

Además:

$$V = 0.004 [A] \cdot 250 [\Omega]$$

$$V = 1 [V]$$

Esto significa que para una resistencia de 250 $[\Omega]$, se tiene un voltaje de 1-5 [V] para una señal de 4-20 [mA].

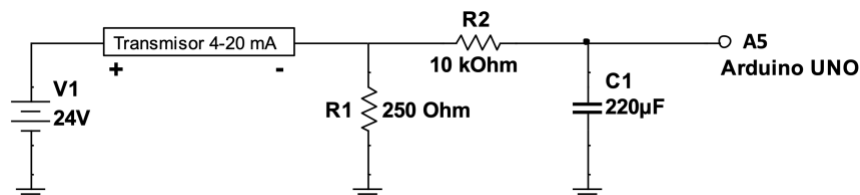
Junto con esto, se decide agregar un filtro pasabajos, para eliminar en parte el ruido producido por la caída del agua hacia el estanque. La frecuencia de corte de un filtro pasabajos está dada por:

$$f_c = \frac{1}{2\pi RC} [Hz]$$

Se establece una frecuencia de corte de 0,1 [Hz] y una resistencia de 10 [k Ω], con lo que es posible encontrar el valor del capacitor:

$$C = \frac{1}{2\pi \cdot 10 [k\Omega] \cdot 0.1 [Hz]} = 160 [\mu F]$$

Se utiliza un valor comercial de 220 $[\mu F]$, por lo que la frecuencia de corte real obtenida es de 0.07 [Hz].



Fuente: Elaboración propia. En base a circuito realizado con NI Multisim.

Figura 2-4. Circuito adaptador sensor de nivel.

2.3.2. Circuito adaptador bomba centrífuga

Para el funcionamiento de la bomba se debe considerar su voltaje, su intensidad de corriente consumida y compararla con las limitaciones de software de la placa Arduino UNO. Visto en el capítulo anterior, se sabe que la placa entrega un voltaje de 5 [Vcc] con un máximo de 8 [mA].

Para la amplificación de la corriente entregada por Arduino UNO, se determina utilizar un transistor de tipo Darlington TIP122, cuyas características están en la tabla 2-1.

Tabla 2-1. Resumen datasheet transistor TIP122.

VALORES MÁXIMOS ABSOLUTOS	
VBE	2.5 V
IC	5 A
IB	0.1 A
Hfe	1000

Fuente: Elaboración propia. Basada en hoja de datos transistor TIP122.

Para obtener la resistencia Rb necesaria para el circuito, se debe obtener la ganancia de corriente (hFE) del transistor y luego calcular el valor de la corriente de base. Se sabe que Ic es 0.8 [A] (consumo de la bomba centrífuga).

$$h_{FE} = \frac{I_C}{I_B}$$

$$I_B = \frac{0.8 [A]}{1000} = 0.0008 [A]$$

Sabiendo el valor de la corriente de base del transistor, se procede al cálculo de la resistencia de base del circuito:

$$R_B = \frac{V_B - V_{BE}}{I_B}$$

$$R_B = \frac{5 [V] - 2.5[V]}{0.0008 [A]} = 3125 [\Omega]$$

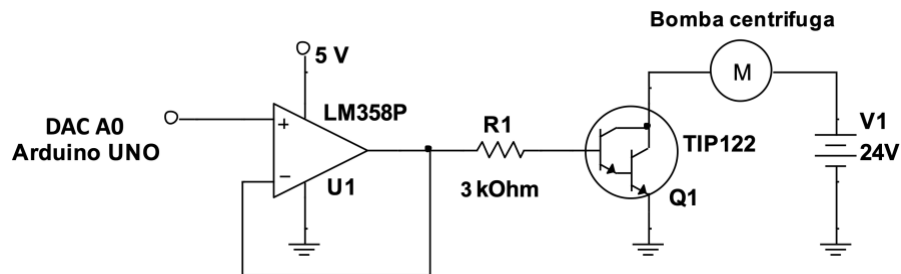
Se utilizan tres resistencias comerciales de 1 [kΩ] en serie.

Junto a lo anterior, se utiliza un circuito integrado LM358P como seguidor de voltaje a la salida del controlador, esto con el fin de proteger el sistema. Las características del circuito integrado LM358P están en la tabla 2-2.

Tabla 2-2. Resumen datasheet AO LM358P.

VALORES MÁXIMOS	
Voltaje de alimentación	3 V – 32 V
Corriente de salida	20 mA
Temperatura de operación	-40°C – 85°C

Fuente: Elaboración propia. Basada en hoja de datos AO LM358P.



Fuente: Elaboración propia. En base a circuito realizado con NI Multisim.

Figura 2-5. Circuito adaptador bomba centrífuga.

2.4. OBTENCIÓN DE DATOS Y FUNCIÓN DE TRANSFERENCIA

Posterior a realizar el conexionado físico del circuito para el funcionamiento del sistema, se debe proceder a la obtención de datos y determinación de la función de transferencia para el

diseño del control PID. Para esto se necesitan realizar algunos pasos previos, los cuales se detallan a continuación.

2.4.1. Escalamiento del sensor de nivel

Para la correcta lectura del nivel de agua en el estanque, se debe realizar la medición de voltaje para un determinado número de niveles de agua y posteriormente calcular el nivel digital correspondiente debido a la conversión análogo-digital que realiza el controlador Arduino UNO.

Para la conversión, se sabe que una lectura de 0 [V] en la entrada digital corresponde al nivel 0 y 5 [V] al nivel 1023. Con esta información se procede a realizar el escalamiento. Los datos experimentales obtenidos desde el sensor de nivel se encuentran en la tabla 2-3.

Tabla 2-3. Datos experimentales de sensor de nivel.

Medición	Nivel [cm]	Voltaje [V]
1	0	0.998
2	30	2.149

Fuente: Elaboración propia. Basada en datos experimentales en laboratorio.

Con una simple regla de tres, se puede obtener el nivel digital para los voltajes obtenidos experimentalmente.

$$\frac{0.998}{5} = \frac{x}{1023}$$

$$x = \frac{0.998 \cdot 1023}{5} = 204$$

$$\frac{2.149}{5} = \frac{x}{1023}$$

$$x = \frac{2.149 \cdot 1023}{5} = 440$$

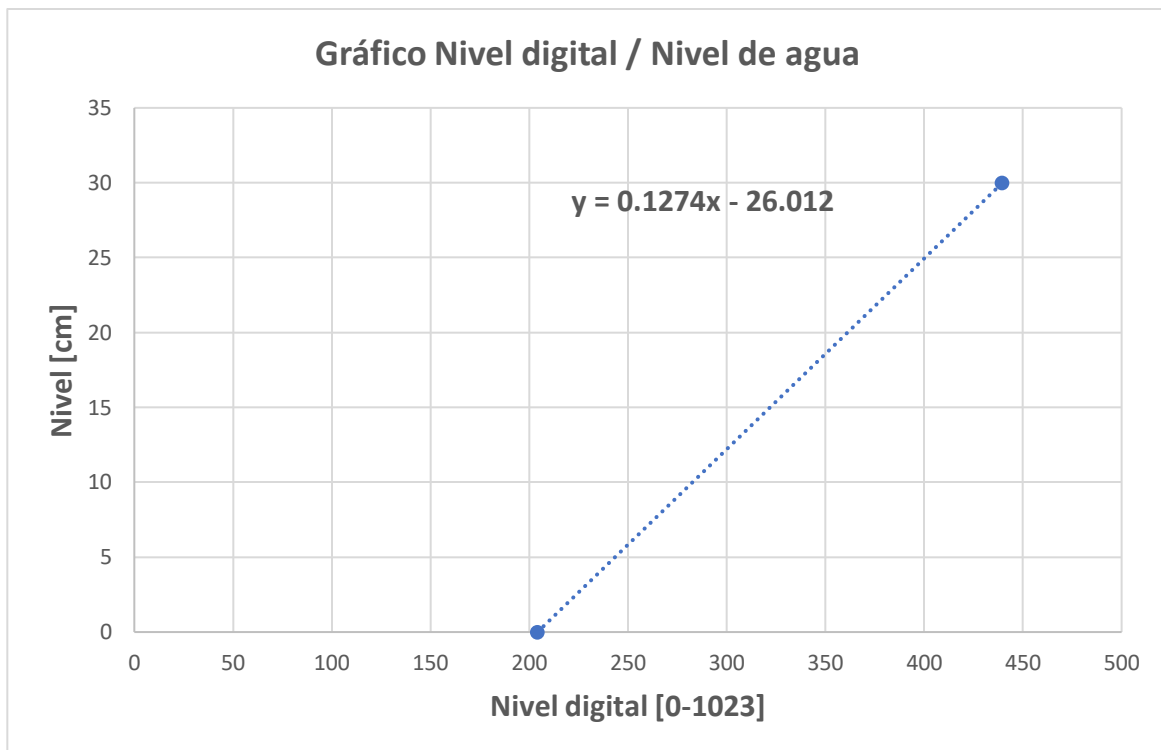
Expandiendo la tabla 2-3, se tienen los datos experimentales junto al nivel digital en la tabla 2-4:

Tabla 2-4. Datos experimentales de sensor de nivel.

Medición	Nivel [cm]	Voltaje [V]	Nivel digital
1	0	0.998	204
2	30	2.149	440

Fuente: Elaboración propia. Basada en datos experimentales en laboratorio y cálculos.

Se insertan los datos de nivel del estanque y nivel digital en Excel para obtener la ecuación de la recta, la cual se ocupará en el desarrollo del código y cuyo gráfico se encuentra en la figura 2-6.



Fuente: Elaboración propia. Basada en datos experimentales en laboratorio.

Figura 2-6. Gráfico nivel digital/nivel de agua.

2.4.2. Obtención de datos en lazo abierto

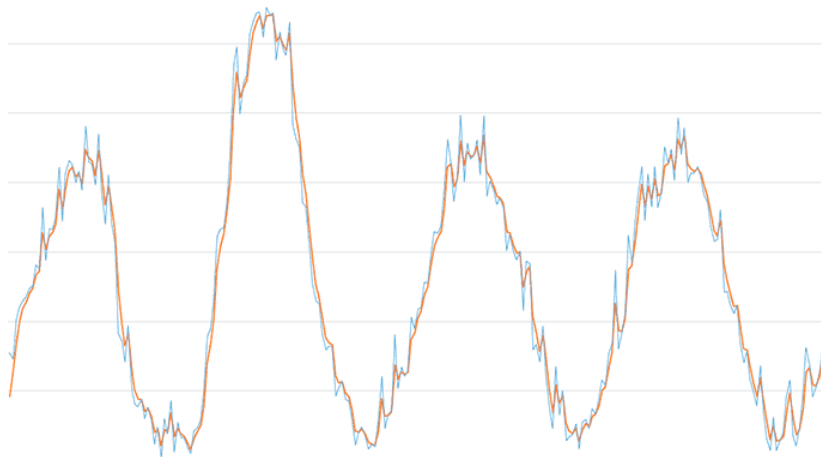
Para la obtención de los datos de nivel se utiliza el software CoolTerm, el cual permite el intercambio de datos con hardware conectado a puertos serie. Con esto es posible registrar los datos que son escritos por Arduino UNO en el puerto serie, para posteriormente realizar el análisis de estos.

Además del filtro pasabajos utilizado en la adaptación del circuito, se decide implementar un filtro digital EMA. Este filtro es utilizado para suavizar señales, reduciendo el ruido y dando más peso a las mediciones recientes. Se calcula con la siguiente ecuación:

$$A_n = \alpha M + (1 - \alpha)A_{n-1}$$

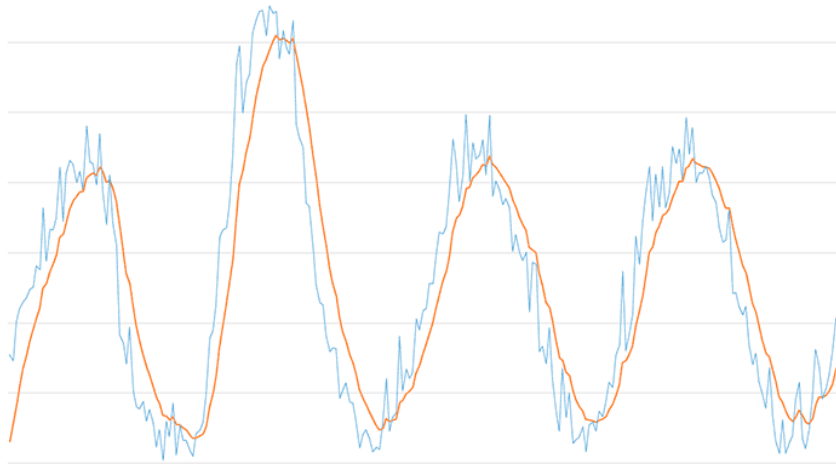
Donde A_n es el valor filtrado, A_{n-1} el valor filtrado anterior, M el valor muestreado de la señal que se desea filtrar y α es un factor entre 0 y 1. Por ejemplo, al asignar el valor $\alpha=1$, la señal filtrada es igual a la señal muestreada; en cambio si se asigna el valor $\alpha=0$, el valor filtrado es igual a 0, ya que no utiliza información nueva aportada desde la medición.

Las siguientes figuras muestran el comportamiento del filtro EMA con diferentes valores de α . La figura 2-7 tiene un factor $\alpha=0.6$, la figura 2-8 un $\alpha=0.2$ y la figura 2-9 un $\alpha=0.05$.



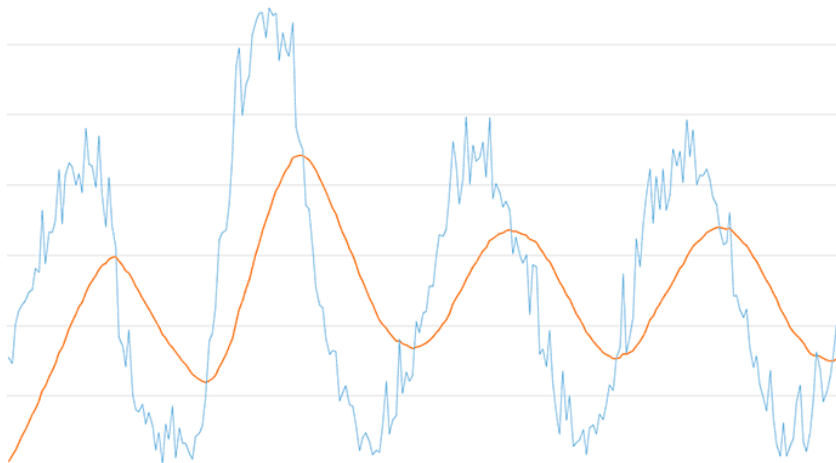
Fuente: <https://www.luisllamas.es/arduino-paso-bajo-exponencial/#filtro-paso-alto>.

Figura 2-7. Señal filtrada con Alpha 0.6.



Fuente: <https://www.luisllamas.es/arduino-paso-bajo-exponencial/#filtro-paso-alto>.

Figura 2-8. Señal filtrada con Alpha 0.2.



Fuente: <https://www.luisllamas.es/arduino-paso-bajo-exponencial/#filtro-paso-alto>.

Figura 2-9. Señal filtrada con Alpha 0.05.

Como se aprecia en las figuras 2-8 y 2-9, un menor valor de α genera mejor filtrado pero un retraso en la señal, lo que puede provocar problemas dependiendo el sistema en el que se trabaja. Para el caso del presente trabajo, como el sistema es lento, se usa un valor $\alpha=0.2$, obteniendo una señal correctamente filtrada, fiel a la original y con poco retraso.

Para la obtención de datos se utilizó el código mostrado en la figura 2-10.

```

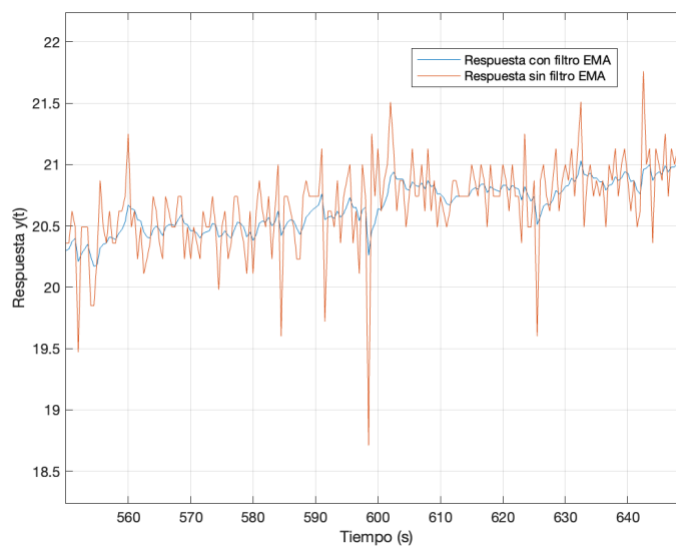
1  int sensor=0;           //Nivel digital de la señal de entrada//
2  float nivel=0.0;        //Nivel digital escalado a cm.//
3  float nivelEMA=0.0;     //Nivel con filtro EMA//
4  float alpha=0.2;        //Factor de suavizado//
5  int dato=0;             //Nivel digital para activación de bomba (de 0 a 255)//
6  float voltaje=0.0;      //Voltaje en salida Arduino//
7
8  void setup() {
9    Serial.begin(9600);   //Iniciar comunicación serial//
10 }
11
12 void loop() {
13   sensor=analogRead(5);  //Asignación pin análogo de lectura//
14   nivel=(0.1274*sensor)-26.012; //Escalamiento del nivel
15   nivelEMA=alpha*nivel+(1-alpha)*nivelEMA; //Cálculo de nivel con filtro EMA//
16   if (Serial.available()>0){
17     dato=Serial.parseInt(); //Asignación manual de nivel digital por puerto serial//
18     analogWrite(A0,dato);   //Escritura de nivel digital en salida análoga//
19   }
20   voltaje=0.0194*dato;     //Cálculo del voltaje en salida Arduino//
21   Serial.print(dato);
22   Serial.print(" , ");
23   Serial.print(voltaje);
24   Serial.print(" , ");
25   Serial.print(sensor);
26   Serial.print(" , ");
27   Serial.println(nivelEMA);
28   delay(500);             //Tiempo de muestreo//
29 }

```

Fuente: Elaboración propia. Captura de pantalla desde programa Arduino IDE.

Figura 2-10. Código de Arduino IDE para obtención de datos.

Se toma un rango de 100 segundos en la recolección de datos para mostrar con la figura 2-11 la utilidad del filtro digital EMA en el sistema de nivel.



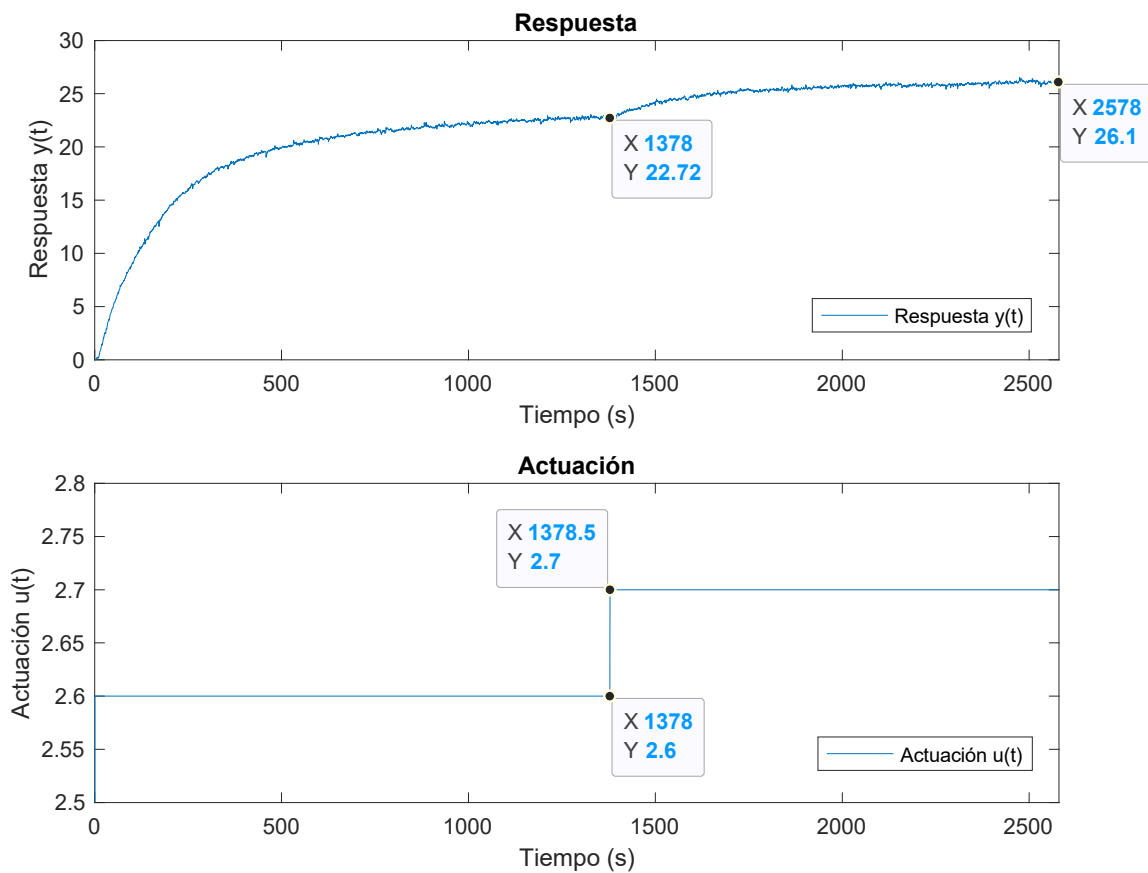
Fuente: Elaboración propia. Basado en recolección de datos en sistema de nivel.

Figura 2-11. Señales con y sin filtro EMA en sistema de nivel.

2.4.3. Punto de operación y función de transferencia

Dado que el sistema de nivel es no lineal, se debe trabajar en un punto de operación. Se aplica un delta escalón de 0.1 [V] en la salida del controlador, equivalente a 0.48 [V] en la bomba con el fin de obtener un modelo lineal del sistema en este punto.

La figura 2-12 muestra en la parte superior la respuesta del sistema y en su parte inferior la actuación. El salto de 2.6 [V] a 2.7 [V] en la salida del controlador equivale a un salto de 12.48 [V] a 12.96 [V] en la bomba. La tabla 2-5 resume los puntos importantes de actuación y respuesta.



Fuente: Elaboración propia. Basado en recolección de datos en sistema de nivel.

Figura 2-12. Gráficos de actuación y respuesta del sistema de nivel.

Tabla 2-5. Resumen de datos de actuación y respuesta.

Actuación (V)		Respuesta (cm)		Punto de operación
u_Q	2.6	y_Q	22.7	$(u_Q, y_Q) = (2.6, 22.7)$
u_∞	2.7	y_∞	26.1	

Fuente: Elaboración propia. Basado en recolección de datos en sistema de nivel.

$$\Delta u = 0.1 [V]$$

$$\Delta y = 3.4 [cm]$$

Luego de obtener la respuesta del sistema con el escalón de 0,1 [V], se procede a la obtención de la función de transferencia. Para esto, se ingresan los datos de la actuación y de la respuesta en MATLAB y se utiliza la herramienta IDENT para determinar $G(s)$ en el punto de operación. En la figura 2-13 se aprecia una captura de pantalla de la identificación por medio de la herramienta IDENT y la función de transferencia obtenida.

The screenshot displays the MATLAB IDENT tool interface. On the left, the 'Transfer Function' section shows the identified model: $\frac{K \exp(-T_d s)}{(1 + T_{p1} s)}$. Below this, the 'Poles' section indicates 1 pole, which is 'All real'. The 'Initial Guess' section shows 'Zero' unchecked, 'Delay' checked, and 'Integrator' unchecked. On the right, a table lists the parameters identified by the tool:

Par	Known	Value	Initial Guess	Bounds
K	☐	33.2935	Auto	[-Inf Inf]
Tp1	☐	249.9719	Auto	[0 260491.7]
Tp2	☐	0	0	[0 Inf]
Tp3	☐	0	0	[0 Inf]
Tz	☐	0	0	[-Inf Inf]
Td	☐	0	Auto	[0 15]

Below the table, the 'Initial Guess' section shows 'Auto-selected' and 'From existing model' as radio button options, and 'User-defined' as a selected radio button. A 'Value-->Initial Guess' button is also present.

Fuente: Elaboración propia. En base a herramienta IDENT de MATLAB.

Figura 2-13. Identificación de función de transferencia.

Sabiendo que el sistema de nivel es de primer orden con retardo, la función de transferencia $G(s)$ es de la siguiente forma:

$$G(s) = \frac{K}{Ts + 1} e^{-T_d s}$$

De los datos obtenidos de la identificación de sistema, se tiene que $K=33.2935$, $T=249.9719$ y $T_d=0$.

El tener un valor $T_d=0$ no permite ocupar los criterios de sintonización que se verán más adelante; para esto, se puede estimar el retardo como el 1% de T . En este caso $T=249.9719$, por lo que $T_d=2.4997$. Aproximando, se tiene que:

$$G(s) = \frac{33.29}{250s + 1} e^{-2.5s}$$

Para comprobar que la función de transferencia es una aproximación correcta del comportamiento del sistema, se puede usar el teorema del valor final. Este teorema establece que:

$$f(\infty) = \lim_{t \rightarrow \infty} f(t) = \lim_{s \rightarrow 0} sF(s)$$

También se sabe que:

$$G(s) = \frac{\Delta Y(s)}{\Delta U(s)}$$

$$\Delta Y(s) = G(s) \cdot \Delta U(s)$$

Para un escalón $\Delta u=0.1$, se tiene que:

$$\Delta y(\infty) = \lim_{s \rightarrow 0} \frac{33.29}{250s + 1} e^{-2.5s} \cdot \frac{0.1}{s} \cdot s = 3.329 \text{ [cm]}$$

En el infinito Δy toma un valor de 3.329 [cm], mientras que en la prueba práctica el valor se estaciona en un Δy de 3.4 [cm], siendo una diferencia de 0.071 [cm] entre el valor práctico y el estimado, comprobando así la fidelidad de la aproximación por MATLAB.

Se puede calcular el error de aproximación de la siguiente forma:

$$Error = \left| \frac{3.329 - 3.4}{3.329} \right| \cdot 100 = 2.1\%$$

2.5. DISEÑO DEL CONTROLADOR PID

Con la obtención de la función de transferencia en el punto de operación, se debe determinar el diseño de un controlador PID. Para esto primero se debe determinar el principio de sintonización, calcular las constantes K_p , T_i y T_d según corresponda y posteriormente desarrollar el código en Arduino IDE.

2.5.1. Criterios de sintonización

Existen varios criterios de sintonización para los controladores PID, algunos de ellos son:

- Lugar geométrico de las raíces: Método computacional de simulación para encontrar los polos de lazo cerrado dependientes del valor de K usando MATLAB, útil para compensar cambios en la referencia.
- Criterio de Ziegler and Nichols de la ganancia máxima: Aplicable solo a plantas en lazo abierto, con razón de amortiguamiento de 25% para perturbaciones.
- Criterio de Cohen Coon: Se basa en la obtención de la función de transferencia a partir de la respuesta a escalón del sistema. Tiene una razón de amortiguamiento de 25% para perturbaciones, mínimo tiempo de estabilización y minimización del error.

- Criterio de Ziegler and Nichols de la curva de reacción: Similar al de Cohen Coon porque también considera plantas con retardo y 25% de razón de amortiguamiento para perturbaciones.

Estos tres últimos criterios son para plantas estables y BIBO, además de útiles para compensar perturbaciones.

En este proyecto se usará el criterio de Ziegler and Nichols de la curva de reacción. En la tabla 2-6 se observan las fórmulas para la sintonización del controlador PID con este criterio.

Tabla 2-6. Constantes para criterio de Ziegler and Nichols.

Tipo de controlador	Kp	Ti	Td
P	$\frac{\gamma_0}{K_0 \cdot \tau_0}$		
PI	$\frac{0.9 \gamma_0}{K_0 \cdot \tau_0}$	$3.3 \tau_0$	
PID	$\frac{1.2 \gamma_0}{K_0 \cdot \tau_0}$	$2 \tau_0$	$0.5 \tau_0$

Fuente: Elaboración propia. En base a PPT "Principios de sintonización PID", asignatura Control de Procesos, profesor Guelis Montenegro Z.

Para el control de este sistema, se realizará un controlador PI. Con los valores obtenidos de la función de transferencia se tiene que $\gamma_0 = 250$, $K_0 = 33.29$ y $\tau_0 = 2.5$. Reemplazando en las fórmulas se tiene que para un controlador PI:

$$K_p = \frac{0.9 \cdot 250}{33.29 \cdot 2.5} = 2.7$$

$$T_i = 3.3 \cdot 2.5 = 8.25 \text{ [s]}$$

2.5.2. Discretización del controlador PID

La ecuación vista en el capítulo 1 de un controlador PID es válida para intervalos continuos de tiempo, pero el sistema ha de ser controlado en forma digital a intervalos discretos, por lo que

se hace necesaria la modificación de la fórmula vista, llevándola a una aproximación que permita el trabajo a estos intervalos. Esto se llama discretización del controlador PID.

Existe la aproximación estándar, con reemplazo de la integral por suma acumulativa de errores por medio de la suma de rectángulos (PID discreto estándar) y la aproximación con integración por el método del trapecio (promedio de valores sucesivos de error). La primera aproximación es suficiente ya que el sistema no sufre de variaciones rápidas.

La ecuación que representa a un controlador PID en su forma discreta estándar es:

$$u(k) = K_p e(k) + K_i T_s \sum_{h=0}^k e(h) + K_d \frac{e(k) - e(k-1)}{T_s}$$

Donde K_p es la ganancia proporcional, K_i es la ganancia integral igual a $\frac{K_p}{T_i}$, siendo T_i el tiempo integral, K_d es la ganancia derivativa igual a $K_p T_d$, siendo T_d el tiempo derivativo, T_s es el tiempo de muestreo, $e(k)$ es el error en el instante k y $e(k-1)$ es el error en el instante anterior.

Para este caso solo se usará un controlador PI, por lo que la ecuación se reduce y queda:

$$u(k) = K_p e(k) + K_i T_s \sum_{h=0}^k e(h)$$

Se determina un T_s de 0.5 [s], elegido en base a las características del sistema.

2.5.3. Código controlador PID

A continuación, se mostrará el código para el controlador PI detallando cada sección para su comprensión.

- Primera etapa: Constantes y variables (figura 2-14).

Se asignan las constantes K_p , K_i , T_s y se asignan las variables del controlador en "0" para evitar problemas al inicio del loop. Además, se agregan dos constantes de tiempo y finalmente se

establece el factor de filtrado EMA en 0.2, agregando una variable del nivel filtrado. Se debe notar que todas las variables y constantes son valores float, a excepción de sensor, que es un nivel digital entre 0 y 1023 convertido por Arduino; output, valor entero de la salida analógica de la placa que va de 0 a 255 por defecto (se puede ajustar la salida a 12 bits con `analogWriteResolution(bits)`); maxOut y minOut, que son las salidas máxima y mínima del controlador a 8 bits; setpointDefinido, que es un valor booleano de comprobación de datos; y ultimoTiempo, que entrega un valor entero sin signo y sin coma decimal, útil para posteriormente calcular con `millis()`.

```

1  //Constantes PI//
2  float Kp=2.7;           //Ganancia proporcional//
3  float Ki=0.33;          //Ganancia integral//
4  float Ts=0.5;           //Tiempo de muestreo en segundos//
5
6  //Variables y constantes PID//
7  float setpoint=0.0;      //Valor deseado de nivel en cm.//
8  int sensor=0;           //Valor de la señal de entrada//
9  float nivel=0.0;        //Nivel digital escalado a cm.//
10 int output=0;           //Salida del PID en nivel digital//
11 float error=0.0;        //Error actual//
12 float integral=0.0;      //Aproximación discreta de integral (error acumulado)//
13 int maxOut=255;         //Nivel digital máximo//
14 int minOut=0;           //Nivel digital mínimo//
15 bool setpointDefinido=false; //Constante booleana de escritura de setpoint//
16
17 //Variables de tiempo//
18 unsigned long ultimoTiempo=0; //Último conteo de tiempo//
19 float Tiempo=Ts*1000;    //Tiempo en milisegundos//
20
21 //Variables filtrado EMA//
22 float alpha=0.2;         //Factor de suavizado//
23 float nivelEMA=0.0;      //Nivel con filtro EMA//

```

Fuente: Elaboración propia. En base a código realizado en Arduino IDE.

Figura 2-14. Constantes y variables del programa.

- Segunda etapa: void setup() (figura 2-15).

Se realiza la configuración inicial por medio de la función void setup(). En esta etapa se inicializa el monitor serie, se asigna el pin A0 como salida y comienza a guardar el tiempo desde que parte la ejecución del programa.

```

25 void setup() {
26     Serial.begin(9600);           //Inicializar comunicación serial//
27     ultimoTiempo = millis();      //Guardar tiempo inicial//
28 }

```

Fuente: Elaboración propia. En base a código realizado en Arduino IDE.

Figura 2-15. Configuración inicial del programa.

- Tercera etapa: void loop() (figura 2-16).

```

30 void loop() {
31     sensor=analogRead(5);          //Asignación pin análogo//
32     nivel=(0.1274*sensor)-26.012;   //Escalamiento del nivel//
33     nivelEMA=alpha*nivel+(1-alpha)*nivelEMA; //Cálculo de nivel con filtro EMA//
34     if(Serial.available()>0){      //Primera condición//
35         float nuevoSetpoint=Serial.parseFloat(); //Lectura de setpoint deseado desde monitor serie//
36         if (nuevoSetpoint>0 && nuevoSetpoint<=30){ //Condición límite de nivel//
37             setpoint=nuevoSetpoint; //Cambio de setpoint a valor ingresado//
38             setpointDefinido=true;   //Cambio de constante booleana//
39         }
40     }
41     if(setpointDefinido){           //Segunda condición//
42         error=setpoint-nivelEMA;     //Cálculo de error//
43         integral+=error;              //Cálculo de integral//
44         output=((Kp*error)+(Ki*Ts*integral))/5)*255; //Cálculo de output//
45         output=constrain(output,minOut,maxOut); //Límitación de output a valores de saturación//
46         if (output>=maxOut){         //Condición salida saturada alta//
47             integral=(maxOut-Kp*error)/(Ki*Ts); //Antiwindup//
48         }
49         if (output<=minOut){         //Condición salida saturada baja//
50             integral=(minOut-Kp*error)/(Ki*Ts); //Antiwindup//
51         }
52         output=((Kp*error)+(Ki*Ts*integral))/5)*255; //Actualización de output//
53         output=constrain(output,minOut,maxOut); //Límitación de output actualizado//
54     }
55
56     analogWrite(A0,output);          //Escribir output en pin//
57
58     //Tiempo de muestreo//
59     while(millis()-ultimoTiempo<Tiempo){ //Espera Ts segundos antes de reiniciar loop (muestreo)//
60     }
61     ultimoTiempo=millis();          //Actualizar el tiempo//
62 }

```

Fuente: Elaboración propia. En base a código realizado en Arduino IDE.

Figura 2-16. Bucle del programa.

Se lee el nivel de agua desde el pin análogo A5. Recordar que la placa realiza una conversión análogo-digital, por lo que la lectura es desde 0 hasta 1023. La variable nivel se

encarga de escalar este dato de nivel digital a centímetros y finalmente nivelEMA es el nivel del estanque pasado por el filtro EMA. La figura 2-17 muestra esta parte del código.

```
sensor=analogRead(5);           //Asignación pin análogo//
nivel=(0.1274*sensor)-26.012;    //Escalamiento del nivel//
nivelEMA=alpha*nivel+(1-alpha)*nivelEMA; //Cálculo de nivel con filtro EMA//
```

Fuente: Elaboración propia. En base a código realizado en Arduino IDE.

Figura 2-17. Lectura nivel de agua del bucle.

Se establece la primera condición en el código; si hay datos en el monitor serie, se asigna el valor ingresado a la variable nuevoSetpoint. Posteriormente se condiciona la entrada del dato a los límites del estanque; si el nivel deseado ingresado está dentro de los márgenes, se asigna el valor nuevoSetpoint a setpoint y se define la variable booleana setpointDefinido como verdadera. La figura 2-18 muestra esta parte del código.

```
if(Serial.available()>0){           //Primera condición//
    float nuevoSetpoint=Serial.parseFloat(); //Lectura de setpoint deseado desde monitor serie//
    if (nuevoSetpoint>0 && nuevoSetpoint<=30){ //Condición límite de nivel//
        setpoint=nuevoSetpoint; //Cambio de setpoint a valor ingresado//
        setpointDefinido=true; //Cambio de constante booleana//
    }
}
```

Fuente: Elaboración propia. En base a código realizado en Arduino IDE.

Figura 2-18. Primera condición del bucle.

Posteriormente, el código se asegura que el setpoint ingresado sea válido. Si esto es así, se continúa con los cálculos. La variable error es la diferencia entre el valor deseado y el valor real; integral es la acumulación del error en cada ciclo; output es la salida del controlador dada por la fórmula del PID discreto. Puesto que la función de transferencia fue identificada en términos de voltaje, el controlador PI entrega una salida en la misma unidad. Se escala output para que su valor sea un nivel digital. Se limita la salida del controlador a los valores límites de la placa, que van de 0 a 255.

- Anti-windup

Después, se establecen dos condiciones para comprobar si la salida está saturada. Si esto es así, se ejecuta una acción anti-windup para que la acción integral no se acumule en exceso y posteriormente cause problemas en el control.

En este trabajo se implementó un anti-windup conocido como integración condicionada o anti-windup por clamping. Este esquema anti-windup detiene la acumulación de la acción integral cuando la salida del controlador alcanza sus límites de saturación. Gracias a este método, se previene la integración excesiva cuando el actuador no puede responder al incremento en la salida por encontrarse en sus límites de saturación, haciendo la respuesta del sistema más estable.

La condición de saturación para la salida del controlador es:

$$0 \leq K_p e(k) + K_i T_s \sum_{h=0}^k e(h) \leq 255$$

Esto significa que la salida es mayor o igual a 0 y menor o igual a 255. Si se despeja el término de la integral, se tiene:

$$\begin{aligned} \sum_{h=0}^k e(h) &\leq 255 - \frac{K_p e(k)}{K_i T_s} \\ \sum_{h=0}^k e(h) &\geq 0 - \frac{K_p e(k)}{K_i T_s} \end{aligned}$$

Este ajuste toma en cuenta la contribución de la acción proporcional en la limitación de la integral. Esto impide que la integral se acumule, ya que la suma de la parte proporcional y la integral será siempre menor o igual que 255 o mayor o igual que 0 cuando la salida se encuentre saturada. Se sobrescribe output con la misma ecuación y se limita a los valores de saturación de la salida.

La figura 2-19 muestra los cálculos de error, integral y output junto con las condiciones para la ejecución del anti-windup.

```

if(setpointDefinido){                                //Segunda condición//
    error=setpoint-nivelEMA;                          //Cálculo de error//
    integral+=error;                                  //Cálculo de integral//
    output=((Kp*error)+(Ki*Ts*integral))/5)*255;       //Cálculo de output//
    output=constrain(output,minOut,maxOut);           //Limitación de output a valores de saturación//
    if (output>=maxOut){                               //Condición salida saturada alta//
        integral=(maxOut-Kp*error)/(Ki*Ts);           //Antiwindup//
    }
    if (output<=minOut){                               //Condición salida saturada baja//
        integral=(minOut-Kp*error)/(Ki*Ts);           //Antiwindup//
    }
    output=((Kp*error)+(Ki*Ts*integral))/5)*255;       //Actualización de output//
    output=constrain(output,minOut,maxOut);           //Limitación de output actualizado//
}

```

Fuente: Elaboración propia. En base a código realizado en Arduino IDE.

Figura 2-19. Segunda condición del bucle y anti-windup.

Finalmente se escribe el valor de salida del controlador en el pin asignado, esperando el tiempo de muestreo deseado. Esta parte dice que mientras el tiempo desde que se encendió la placa menos la última vez que se ejecutó el programa (`millis()-ultimoTiempo`) sea menor a 0.5 segundos (tiempo de muestreo escogido), el código esperará. Cuando pase este tiempo, se avanzará y se guarda el tiempo actual de la placa en la variable `ultimoTiempo`; así, la próxima vez que se ejecute el `while`, esperará otros 0.5 segundos para continuar. Todo esto se muestra en la figura 2-20.

```

analogWrite(A0,output);                               //Escribir output en pin//

//Tiempo de muestreo//
while(millis()-ultimoTiempo<Tiempo){                 //Espera Ts segundos antes de reiniciar loop (muestreo)//
}
ultimoTiempo=millis();                                //Actualizar el tiempo//

```

Fuente: Elaboración propia. En base a código realizado en Arduino IDE.

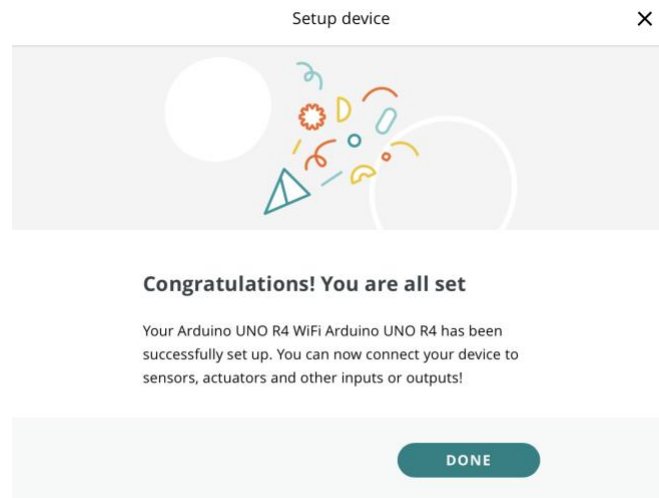
Figura 2-20. Salida y tiempo de muestreo del bucle.

2.5.4. Funcionamiento con Arduino Cloud

El objetivo de hacer funcionar el control PI con una plataforma IoT es la visualización remota de actuación y nivel, además de establecer un setpoint desde el mismo lugar, sin necesidad de utilizar el monitor serie. Para configurar el uso de la placa con la plataforma, hay que ejecutar algunos pasos previos.

La primera parte consiste en crear una cuenta gratis en Arduino Cloud para poder utilizar la plataforma en el botón “Get started for free”.

Ya creada la cuenta se debe vincular la placa en la pestaña “Devices” y el botón “Add device”. La plataforma solicitará agregar entre dispositivos de la marca Arduino, de terceros o realizar una configuración manual para cualquier dispositivo. En este caso se escoge la opción de agregar un dispositivo de la marca Arduino. Se pedirá la instalación de Arduino Cloud Agent, complemento necesario para la comunicación en serie entre la placa y Arduino Cloud. Posterior a este paso se debe conectar la placa, darle un nombre e instalar. La figura 2-21 muestra la pantalla mostrando un mensaje final de instalación completada.



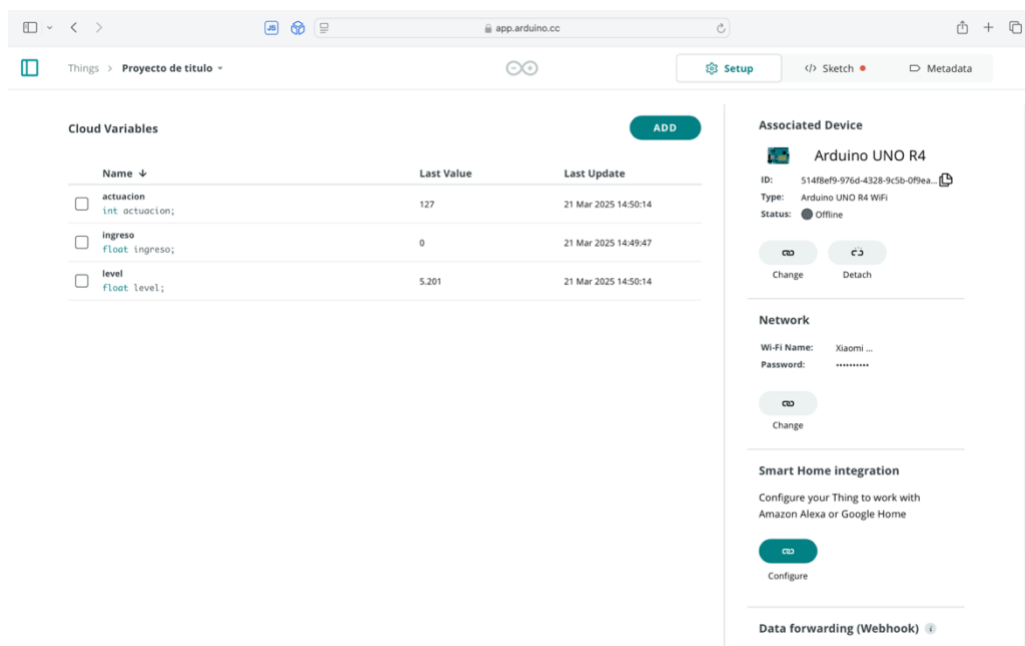
Fuente: Elaboración propia. En base a instalación de placa en Arduino Cloud.

Figura 2-21. Instalación completada de placa.

Con la instalación ya realizada se debe crear un “Thing”, un gemelo virtual de la placa física con la que se trabaja, en la que se realizan las configuraciones para el funcionamiento de los proyectos. En esta interfaz se puede:

- Crear las variables en la nube con las que se sincronizará el dispositivo.
- Seleccionar el dispositivo con el que se desea asociar.
- Introducir las credenciales de red para la conexión.
- Editar y subir “Sketches”, archivo donde se escribe el programa que se ejecutará en la placa.

Se crea un Thing con el nombre “Proyecto de título”, se asocia el dispositivo que se instaló previamente, se guardan las credenciales de red para la conexión y se crean tres variables; “actuacion”, que mostrará la actuación o salida del controlador; “ingreso”, que solicitará escribir el nivel de agua deseado; y “level”, que mostrará el nivel de agua en el estanque. Se muestra la pantalla Things junto a sus variables en la figura 2-22.

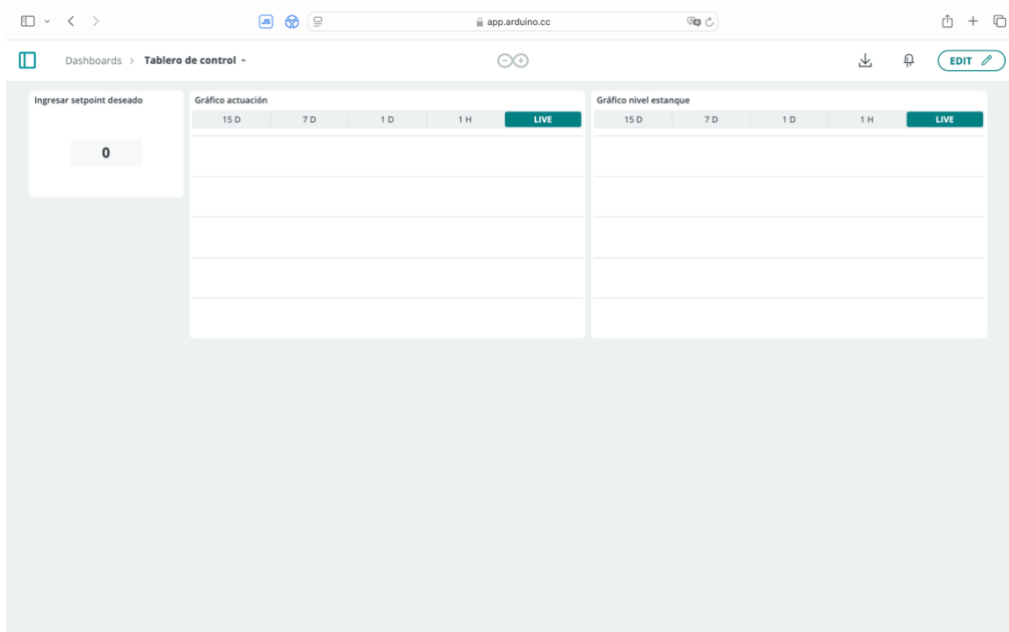


Fuente: Elaboración propia. En base a pantalla “Things” en Arduino Cloud.

Figura 2-22. Pantalla "Things" en Arduino Cloud.

Para poder visualizar las variables es necesario crear “Dashboards”, utilizados para monitorear y controlar fácilmente las placas asociadas desde una interfaz web en una computadora o smartphone. Estos se componen de “Widgets”, que están directamente vinculados con alguna variable creada. Cuando se actualiza una variable, el widget también lo hace.

Para este proyecto se crea un “Tablero de control” (figura 2-23) con tres widgets asociados a las variables creadas, “Ingresar setpoint deseado”, “Gráfico actuación” y “Gráfico nivel estanque”.



Fuente: Elaboración propia. En base a pantalla “Dashboards” en Arduino Cloud.

Figura 2-23. Dashboard con widgets asociados a variables.

Finalmente se debe crear un Sketch para el funcionamiento de la placa. Al crear variables en la nube, estas automáticamente aparecen creadas en el Sketch y se crean funciones asociadas a ellas con el nombre `void on“Variable”Change()`. Esta función se ejecuta cada vez que un nuevo valor de la variable es recibido desde la nube. El único valor que cambiará cuando se requiera es el del setpoint deseado, por lo que la parte del código para poder realizar esto se ejecutará en esta función (figura 2-24).

```

void onIngresoChange() {
  // Add your code here to act upon Ingreso change
  if (ingreso>=0 && ingreso<=30){
    setpointDefinido=true;
    setpoint=ingreso;
  }
}

```

Fuente: Elaboración propia. En base a pantalla "Sketches" en Arduino Cloud.

Figura 2-24. Uso de función para ingresar setpoint.

Se dejan las mismas variables, constantes y void setup() creados en el código original y se establece que level=nivelEMA y actuacion=output, para poder visualizar estas variables en el Dashboard. El resto del Sketch se deja tal y como aparece en Arduino Cloud (figura 2-25). Finalmente, se carga este Sketch a la placa asociada al principio y se puede trabajar con ella.

```

void loop() {
  ArduinoCloud.update();
  // Your code here
  sensor=analogRead(5);
  nivel=(0.1274*sensor)-26.012;
  nivelEMA=alpha*nivel+(1-alpha)*nivelEMA;
  level=nivelEMA;

  if(setpointDefinido){
    error=setpoint-nivelEMA;
    integral+=error;
    output=((Kp*error)+(Ki*Ts*integral))/5)*255;
    output=constrain(output,minOut,maxOut);
    actuacion=output;
    if (output>=maxOut){
      integral=(maxOut-Kp*error)/(Ki*Ts);
    }
    if (output<=minOut){
      integral=(minOut-Kp*error)/(Ki*Ts);
    }
    output=((Kp*error)+(Ki*Ts*integral))/5)*255;
    output=constrain(output,minOut,maxOut);
    actuacion=output;
  }

  analogWrite(A0,output);

  //Tiempo de muestreo//
  while(millis()-ultimoTiempo<Tiempo){
  }
  ultimoTiempo=millis();
}

```

Fuente: Elaboración propia. En base a pantalla "Sketches" en Arduino Cloud.

Figura 2-25. Bucle del Sketch en Arduino Cloud.

CAPÍTULO 3: RESULTADOS Y ANÁLISIS

3. RESULTADOS Y ANÁLISIS

Este capítulo tiene por finalidad realizar las pruebas del controlador PI en la planta de nivel y comparar los resultados obtenidos con las pruebas realizadas en lazo abierto.

Las pruebas consisten en aplicar perturbaciones en el momento que el nivel se encuentre en el setpoint requerido. Estas perturbaciones consisten en abrir la válvula FV 03 en un 50% y cerrar la válvula FV 02 un 25%.

Se mostrarán dos gráficas en las figuras, la del nivel de agua y la de la actuación de la bomba. Cabe señalar que el voltaje visto en las gráficas de actuación es el recibido por la bomba.

Además, para entender mejor el comportamiento de la componente integral, se agregarán solo con fines didácticos dos pruebas extras, eliminando K_i y el anti-windup, mostrando el funcionamiento ante un cambio de referencia.

Finalmente, se mostrará el dashboard de Arduino Cloud con la visualización del nivel y actuación en tiempo real, junto con poder cambiar el setpoint deseado.

3.1. PRUEBAS DEL CONTROLADOR PI

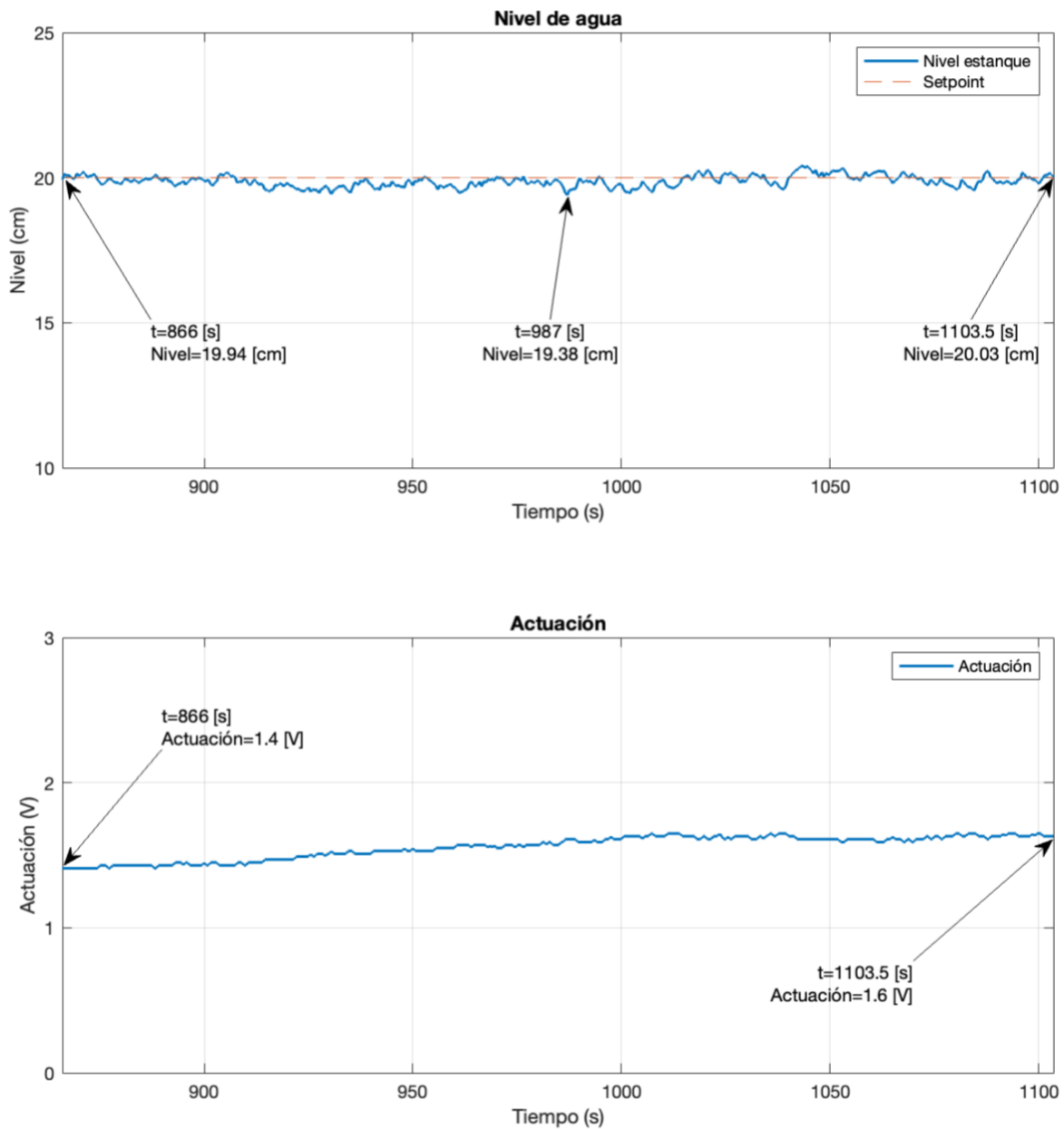
En este apartado se mostrarán las pruebas realizadas y los resultados obtenidos a través de gráficos hechos con MATLAB a partir de los datos recolectados.

Cabe destacar que si bien es cierto $y_Q = 22.7$ [cm], las pruebas del controlador se realizaron a un setpoint de 20 [cm] por ser un valor cercano al punto de operación y más práctico de trabajar. Katsuhiko Ogata declara “Este sistema lineal es equivalente al sistema no lineal, considerado dentro de un rango de operación limitado”.

Se consideró válido realizar las pruebas a 20 [cm] debido a que el sistema presenta un comportamiento suave y estable en ese rango y con condiciones similares de operación. La elección de este setpoint no compromete la validez de los resultados ni el desempeño del controlador PI.

3.1.1. Respuesta del controlador PI abriendo válvula FV 03

La primera prueba realizada (figura 3-1) consiste en mantener los valores teóricos obtenidos de $K_p=2.7$ y $K_i=0.33$. Cuando el nivel se encuentra en el setpoint deseado, se aplica una perturbación al sistema, abriendo la válvula manual FV 03 en un 50%.



Fuente: Elaboración propia. Basado en recolección de datos en sistema de nivel.

Figura 3-1. Prueba con valores teóricos y abriendo válvula FV 03 al 50%.

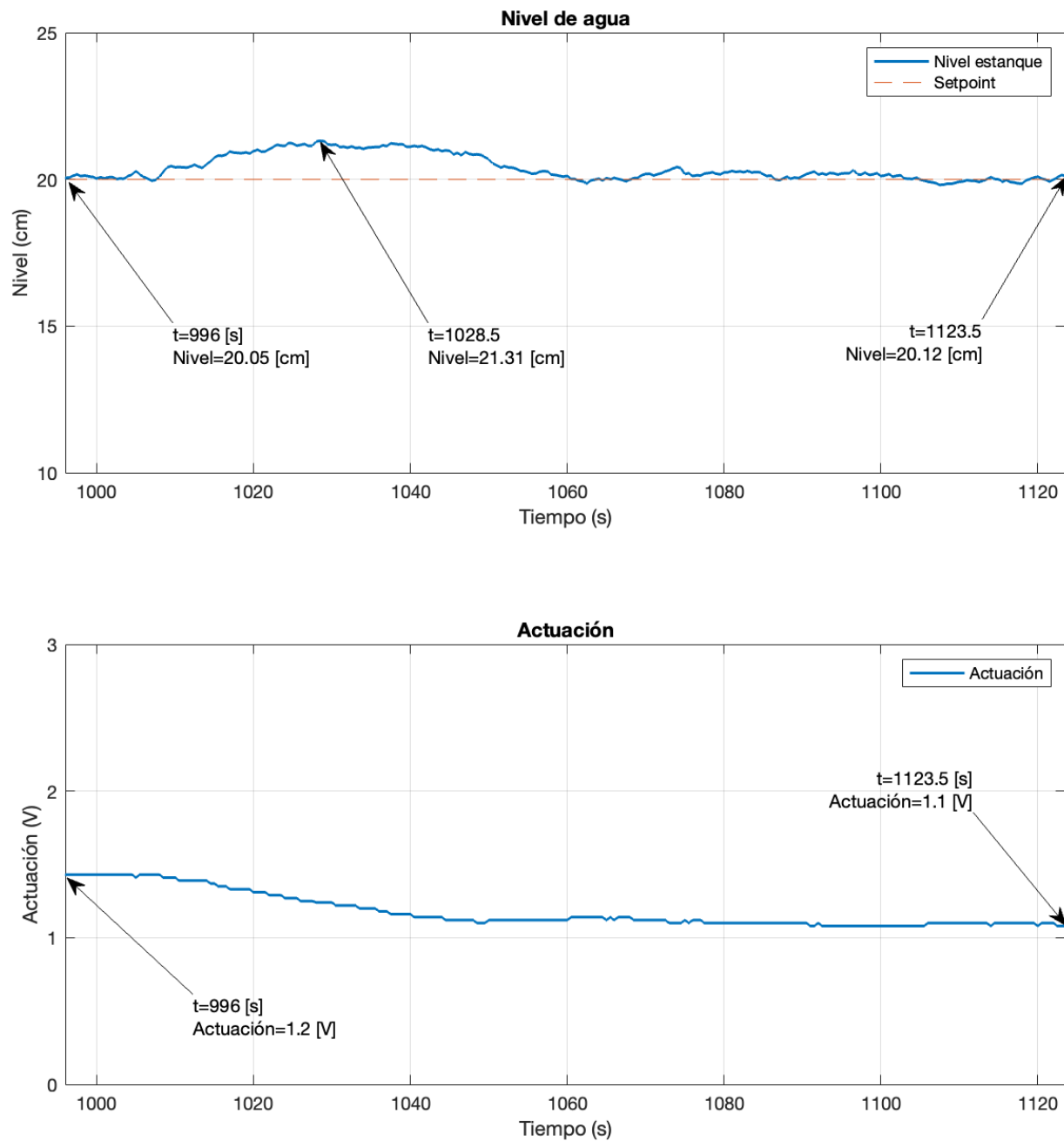
A los 866 [s] se aplica la perturbación, corrigiendo rápidamente y llegando a un nivel mínimo de 19.38 [cm] a los 987 [s], casi 2 minutos después de la perturbación. Pasados los 1000 [s] desde el inicio de la prueba, el sistema ya se encuentra completamente estabilizado. Pasan cerca de 4 minutos entre la perturbación y la estabilización completa del sistema.

Se sabe que el criterio de sintonización utilizado permite compensar perturbaciones con razón a/b menor a $\frac{1}{4}$, siendo b el primer overshoot y a el segundo overshoot (después de aplicada la perturbación). En este primer caso, el segundo overshoot es prácticamente 0, por lo que la relación a/b también es 0, comprobando la utilidad del criterio.

En la gráfica de actuación de la figura 3-1 se aprecia que la salida sube de 1.4 [V] en la salida de Arduino (6.8 [V] a la entrada de la bomba) a 1.6 [V] (7.8 [V] a la entrada de la bomba), producto de la mayor pérdida de agua del sistema.

3.1.2. Respuesta del controlador PI cerrando válvula FV 02

La segunda prueba realizada (figura 3-2) consiste en mantener los valores teóricos obtenidos y aplicar una perturbación al sistema, cerrando la válvula manual FV 02 en un 25%.



Fuente: Elaboración propia. Basado en recolección de datos en sistema de nivel.

Figura 3-2. Prueba con valores teóricos y cerrando válvula FV 02 al 25%.

A los 996 [s] se aplica la perturbación, corrigiendo rápidamente y llegando a un nivel máximo de 21.31 [cm] a los 1028.5 [s], casi medio minuto después de aplicada la perturbación.

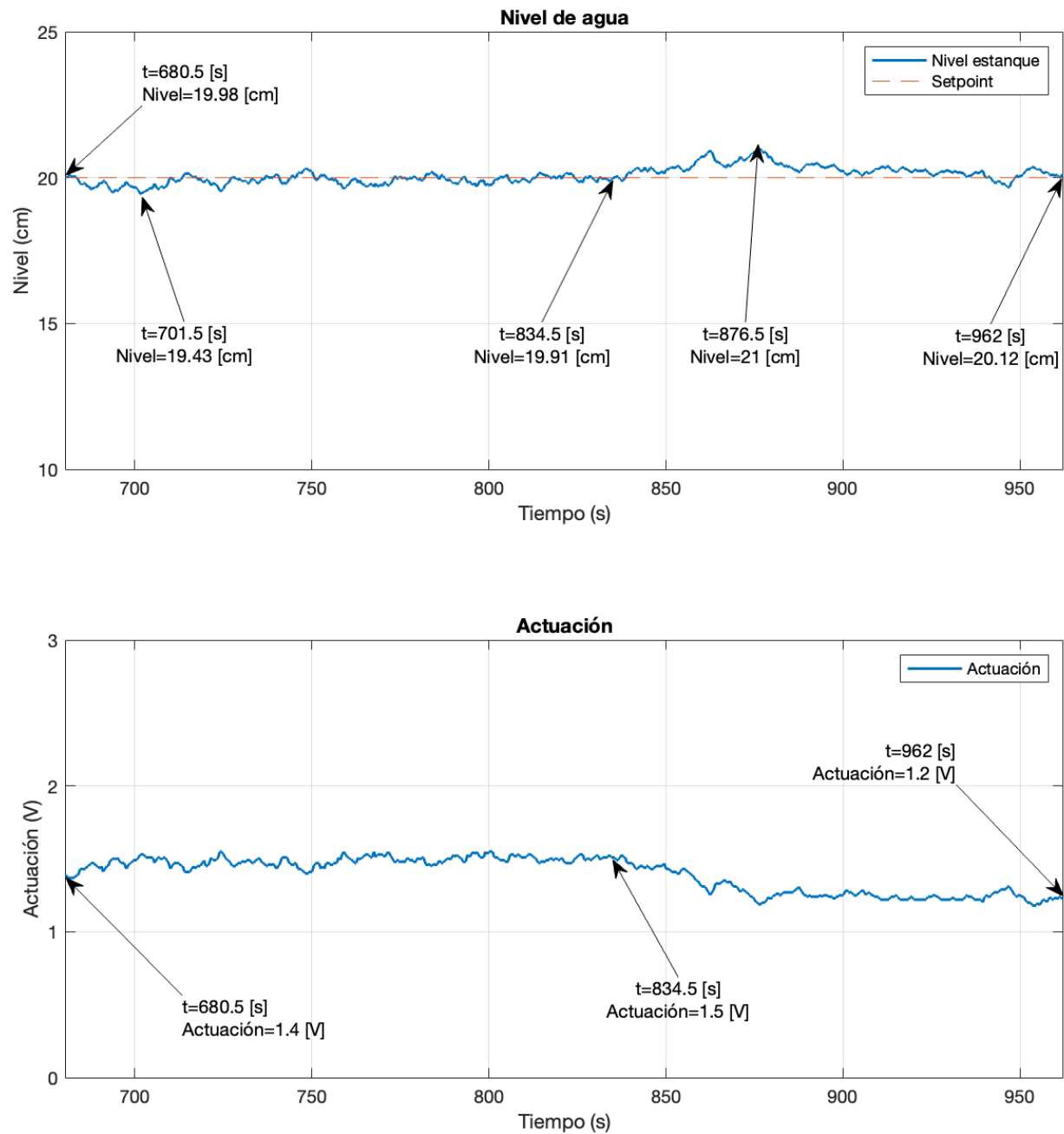
Pasados los 1000 [s] el sistema ya se encuentra completamente estabilizado. Pasan cerca de 2 minutos entre la perturbación y la estabilización completa del sistema.

En esta nueva prueba, el segundo overshoot también es prácticamente 0, por lo que la relación a/b también es 0, comprobando nuevamente la utilidad del criterio.

En la gráfica de actuación de la figura 3-2 se aprecia que la salida baja de 1.2 [V] en la salida de Arduino (6.9 [V] a la entrada de la bomba) a 1.1 [V] (5.2 [V] a la entrada de la bomba), producto de la menor pérdida de agua del sistema.

3.1.3. Respuesta del controlador PI con K_p alto

La tercera prueba (figura 3-3) consiste en variar la constante K_p , dándole un valor más alto del obtenido teóricamente. Se elige un valor $K_p=10$, se mantiene el valor de K_i igual al teórico y se realizan las dos perturbaciones anteriores en la misma prueba.



Fuente: Elaboración propia. Basado en recolección de datos en sistema de nivel.

Figura 3-3. Prueba con $K_p=10$.

Para la primera perturbación a los 680.5 [s], se alcanza el undershoot casi a los 20 [s] después de 19.43 [cm] y una estabilización cerca de 2.5 minutos después. Se aplica la segunda

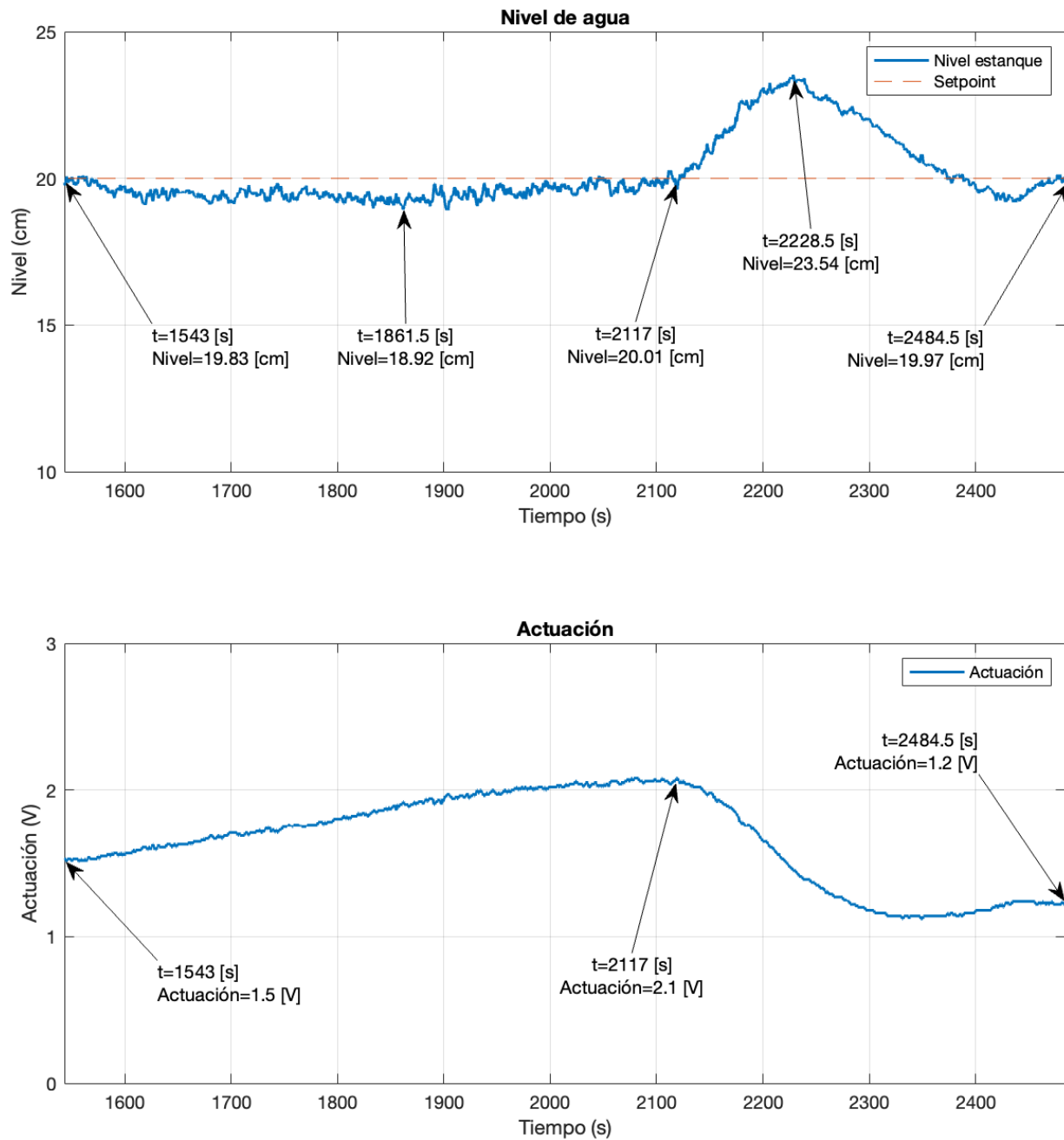
perturbación a los 834.5 [s], llegando a un overshoot de un 21 [cm] casi 40 [s] después y estabilizándose a los 2 minutos.

En ambas perturbaciones, el segundo overshoot es prácticamente 0, por lo que la relación a/b también es 0. El sistema tarda cerca de 280 [s] en estar estabilizado después de ambas perturbaciones.

La actuación tiene un comportamiento similar a las pruebas anteriores, subiendo el voltaje aplicado a la bomba cuando hay mayor pérdida de agua y bajando cuando la pérdida de agua es menor. Los 1.4 [V] en la salida de Arduino equivalen a 6.7 [V] a la entrada de la bomba, 1.5 [V] en Arduino equivalen a 7.2 [V] en la bomba y 1.2 [V] en la salida equivalen a 5.8 [V] a la entrada de la bomba.

3.1.4. Respuesta del controlador PI con K_i bajo

La cuarta prueba (figura 3-4) consiste en variar la constante K_i , dándole un valor más bajo del obtenido teóricamente. Se elige un valor $K_i=0.1$ y se mantiene el valor de K_p igual al teórico. Se realizan las dos perturbaciones en la misma prueba.



Fuente: Elaboración propia. Basado en recolección de datos en sistema de nivel.

Figura 3-4. Prueba con $K_i=0.1$.

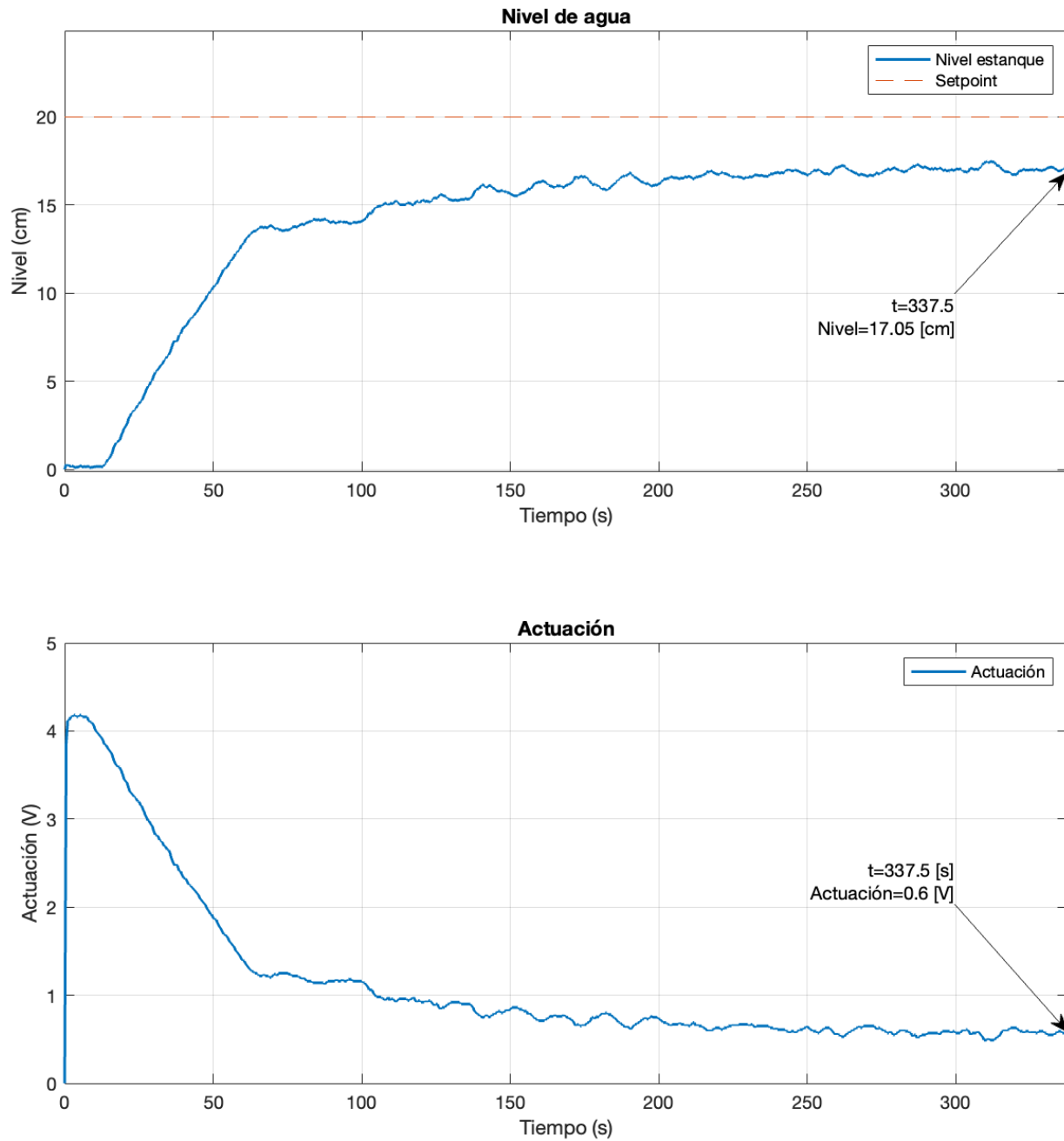
Para la primera perturbación a los 1543 [s], se alcanza el undershoot casi a los 5 minutos de 18,92 [cm] y una estabilización cerca de 4 minutos después de esto. Se aplica la segunda

perturbación a los 2117 [s], llegando a un overshoot de un 23.54 [cm] casi 2 minutos después y estabilizándose cerca de 6 minutos después de aplicada la perturbación.

En ambas perturbaciones, el segundo overshoot es prácticamente 0, por lo que la relación a/b también es 0. El sistema tarda cerca de 941 [s] en estabilizarse, tiempo mucho mayor en comparación a la prueba anterior con K_p alto y K_i teórico. Los 1.5 [V] en la salida de Arduino equivalen a 7.3 [V] a la entrada de la bomba, 2.1 [V] en Arduino equivalen a 9.9 [V] en la bomba y 1.2 [V] en la salida equivalen a 5.8 [V] a la entrada de la bomba.

3.1.5. Respuesta del controlador PI sin acción integral

La quinta prueba (figura 3-5) consiste en eliminar por completo la acción integral, dejando $K_i=0$. Se decide utilizar un $K_p=10$, solo para mostrar el fenómeno ocurrido.



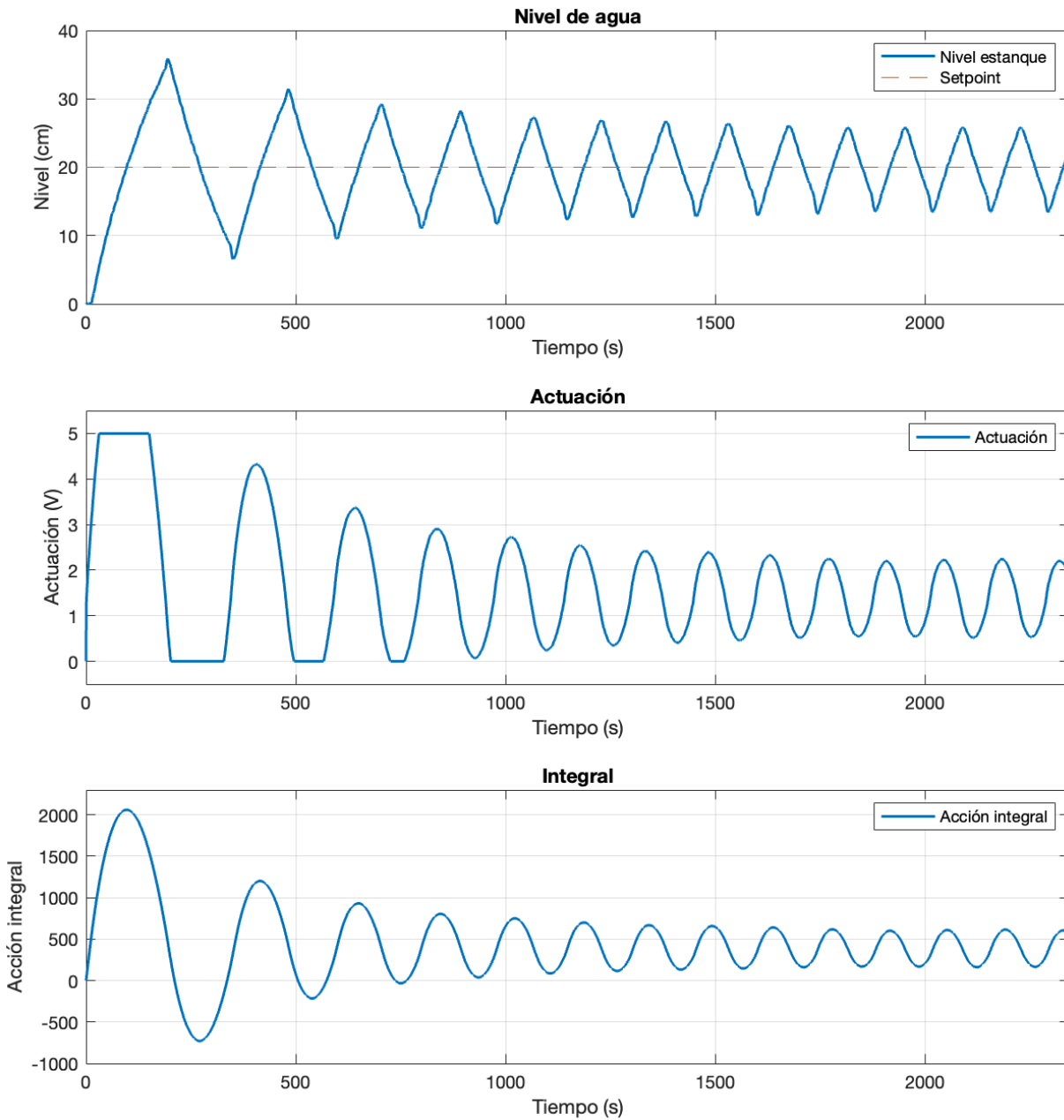
Fuente: Elaboración propia. Basado en recolección de datos en sistema de nivel.

Figura 3-5. Prueba sin acción integral.

Se aprecia en la figura 3-5 que al no existir acción integral y con un $K_p=10$, la salida no llega a saturarse, pero no se alcanza el setpoint requerido, por lo que el error nunca llega a ser 0. Los 0.6 [V] alcanzados en la salida de Arduino equivalen a 2.7 [V] en la entrada de la bomba.

3.1.6. Respuesta del controlador PI sin anti-windup

La última prueba (figura 3-6) consiste en eliminar el anti-windup y así ver cómo actúa el controlador.



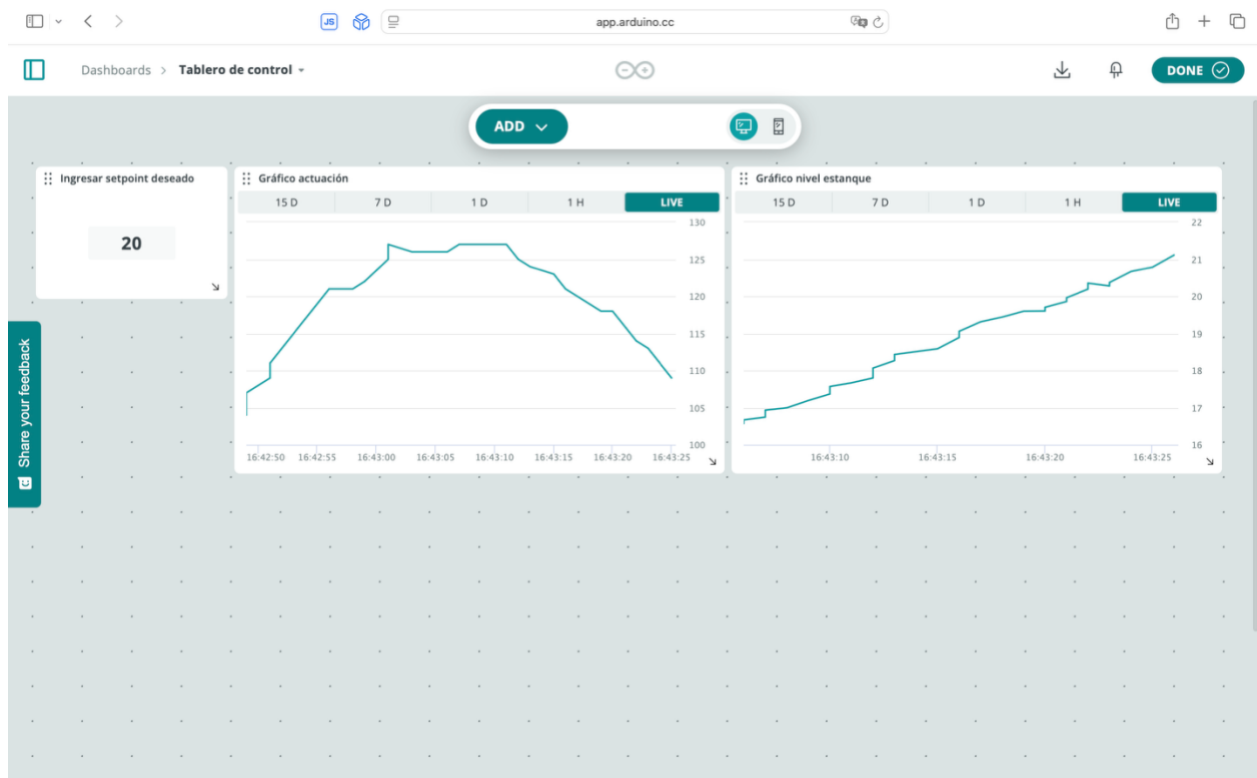
Fuente: Elaboración propia. Basado en recolección de datos en sistema de nivel.

Figura 3-6. Prueba sin anti-windup.

Producto de la acumulación excesiva de la acción integral, el sistema tiene dificultad para estabilizarse.

3.2. FUNCIONAMIENTO CON ARDUINO CLOUD

Posterior a las pruebas realizadas, se procede a la comunicación entre la placa y Arduino Cloud para ver el funcionamiento en forma remota.



Fuente: Elaboración propia. En base a pantalla "Dashboards" en Arduino Cloud.

Figura 3-7. Dashboard de Arduino Cloud en funcionamiento.

Se logra visualizar en la figura 3-7 el ingreso del setpoint requerido, el gráfico de la actuación y de nivel del estanque.

Finalmente también se obtiene una captura de pantalla desde la visualización remota en un smartphone (figura 3-8), donde se aprecia parte del Dashboard, como el setpoint y el gráfico de actuación.



Fuente: Elaboración propia. En base a pantalla "Dashboard" en Arduino Cloud desde smartphone.

Figura 3-8. Visualización de Dashboard desde smartphone.

3.3. OBSERVACIONES DE LOS RESULTADOS OBTENIDOS

Al finalizar las pruebas se observa que los resultados teóricos obtenidos para los valores K_p y K_i fueron adecuados. Al aplicar las perturbaciones se tiene que la relación a/b es prácticamente 0 producto de que no existe un segundo overshoot, demostrando la utilidad del criterio de sintonización utilizado.

Un K_p alto provocó que el sistema responda más rápido en presencia de perturbaciones. Como el sistema responde más rápido, el error se reduce más rápido y la acción integral se acumula menos, provocando que la salida (suma de P+I) sea más baja en comparación a la prueba con K_i bajo.

Con un K_i bajo, el sistema respondió más lento ante las perturbaciones. La parte integral del controlador provoca que el sistema tarde más en corregir el error y como consecuencia de esto, el error se mantiene durante más tiempo, generando que la acción integral se acumule más, haciendo que su aporte sea significativamente mayor y generando que la salida también aumente, llegando a un valor de 2.1 [V] en la salida de Arduino (casi 10 [V] en la bomba).

Como se mencionó en la introducción de este capítulo, las últimas dos pruebas fueron realizadas con fines solamente didácticos. Se visualiza en la prueba sin acción integral que no se llega al setpoint requerido. Esto coincide con la teoría, ya que la acción integral hace que el error sea siempre cero. Junto con esto se aprecia también que la actuación disminuye, ya que la acción proporcional depende directamente del error y este al ser cada vez menor, también lo es la salida del controlador.

También se aprecia en la figura 3-6 el aporte del anti-windup. Cuando se eliminó esta parte importante, el sistema comenzó a oscilar, deteniéndose la prueba después de un rato, ya que de igual forma el nivel en algún instante de tiempo debe llegar a un error igual a cero.

CONCLUSIONES Y RECOMENDACIONES

La implementación de un control PI en Arduino UNO R4 WiFi es eficaz para el control automático del nivel de agua de la plataforma Riegel. Esto ayuda a generar una solución barata y sencilla a la problemática existente en la industria. El control por medio de la placa Arduino UNO R4 es capaz de mantener un nivel de agua adecuado, con bajo margen de error y capaz de actuar correctamente ante perturbaciones.

Para lograr esta implementación, el modelado del sistema fue fundamental para lograr los objetivos y diseñar correctamente el controlador. Esta etapa se encontró con importantes influencias del entorno real en el que se encuentra el sistema, tales como perturbación y ruido en la señal del transmisor de nivel debido a la caída y movimiento del agua y limitación del hardware, debiendo adaptar circuitos para tales propósitos.

Se evidenció con claridad lo enseñado durante el proceso de estudio en la carrera, consiguiendo diferentes resultados al variar los parámetros K_p y K_i . También se logró entender en forma gráfica la importancia del anti-windup en un control del tipo PID, lo que también concuerda con lo aprendido en clases.

Con todo esto, Arduino se comportó como una plataforma lo suficientemente accesible y robusta para implementar un control PI. Hay facilidad en la programación y conexión física de sensores y actuadores, permitiendo su uso no solo para fines educativos, sino que también es factible escalar hacia procesos industriales de mayor tamaño, agregando monitoreo remoto, alarmas y múltiples otras variables.

Se recomienda implementar a cada una de las plantas de la sala M-203 un filtro físico lo suficientemente robusto como para reducir las perturbaciones de la señal del transmisor de nivel o en su defecto, conseguir que la caída de agua sea lo suficientemente suave para evitar las excesivas perturbaciones en la señal. Mejoras en el diseño físico del sistema pueden impactar en forma significativa el resultado final y rendimiento del controlador.

Finalmente, todos los objetivos tanto general como específicos, se lograron.

BIBLIOGRAFÍA

Arduino. What is Arduino? [en línea]. 05 de febrero de 2018. <<https://www.arduino.cc/en/Guide/Introduction>> [consulta: octubre 2024].

Arduino. Arduino Hardware [en línea]. 11 de abril de 2022. <<https://www.arduino.cc/en/hardware>> [consulta: octubre 2024].

CASTAÑO, Sergio. Controladores PID Discreto [en línea]. [s. a.]. <<https://controlautomaticoeducacion.com/control-realimentado/controladores-pid-discreto/>> [consulta: marzo 2025].

DOS20. Somos [en línea] <<https://dos20.cl/somos/>> [consulta: octubre 2024].

Ingeniería MCI Ltda. Software de Arduino [en línea]. [s. a.]. <<https://arduino.cl/programacion/>> [consulta: octubre 2024].

Keller Pressure. Transmisor piezoresistivo de nivel low-cost, Serie 26Y. 06/2015. Winterthur, Suiza: 2015. 2 p.

LLAMAS, Luis. Filtro paso bajo y paso alto exponencial (EMA) en Arduino. [En línea]. [s. a.]. <<https://www.luisllamas.es/arduino-paso-bajo-exponencial/#filtro-paso-alto>> [consulta: noviembre 2024].

MathWorks. Visión general [en línea]. [s. a.]. <<https://la.mathworks.com/products/matlab.html>> [consulta: octubre 2024].

MONTENEGRO, Guelis. Control de procesos. Introducción [diapositiva]. Viña del Mar, Chile: Departamento de electrotecnia e informática. 2023a. 29 diapositivas.

MONTENEGRO, Guelis. Control de procesos. Principios de sintonización PID [diapositiva]. Viña del Mar, Chile: Departamento de electrotecnia e informática. 2023b. 90 diapositivas.

OGATA, Katsuhiko. Sistemas de control en tiempo discreto. 2a ed. Edo. de México: Prentice Hall Hispanoamérica S.A., 1996. 745 p. ISBN: 968-880-539-4.

OGATA, Katsuhiko. Ingeniería de control moderna. 5a ed. Madrid: Person Educación, S.A., 2010. 895 p. ISBN: 978-84-8322-660-5.

SPX Flow Technology. Heavy Duty, Magnetic driven, Seal-less, Circulating pumps. IB-301 R06. 04/2012. Örebro, Suecia: 2012. 22 p.

ANEXOS

ANEXO A: CÓDIGO DE ARDUINO PARA RECOLECCIÓN DE DATOS

```

int sensor=0;      //Nivel digital de la señal de entrada//
float nivel=0.0;   //Nivel digital escalado a cm.//
float nivelEMA=0.0; //Nivel con filtro EMA//
float alpha=0.2;   //Factor de suavizado//
int dato=0;        //Nivel digital para activación de bomba (de 0 a 255)//
float voltaje=0.0; //Voltaje en salida Arduino//

void setup() {
  Serial.begin(9600); //Iniciar comunicación serial//
}

void loop() {
  sensor=analogRead(5);           //Asignación pin análogo de lectura//
  nivel=(0.1274*sensor)-26.012;    //Escalamiento del nivel//
  nivelEMA=alpha*nivel+(1-alpha)*nivelEMA; //Cálculo de nivel con filtro EMA//
  if (Serial.available()>0){
    dato=Serial.parseInt(); //Asignación manual de nivel digital por puerto serial//
    analogWrite(A0,dato);   //Escritura de nivel digital en salida análoga//
  }
  voltaje=0.0194*dato; //Cálculo del voltaje en salida Arduino//
  Serial.print(dato);
  Serial.print(" , ");
  Serial.print(voltaje);
  Serial.print(" , ");
  Serial.print(sensor);
  Serial.print(" , ");
  Serial.println(nivelEMA);
  delay(500); //Tiempo de muestreo//
}

```

ANEXO B: CÓDIGO DE ARDUINO PARA CONTROLADOR PID

```
//Constantes PI//
float Kp=2.7;           //Ganancia proporcional//
float Ki=0.33;          //Ganancia integral//
float Ts=0.5;           //Tiempo de muestreo en segundos//

//Variables y constantes PID//
float setpoint=0.0;     //Valor deseado de nivel en cm.//
int sensor=0;           //Valor de la señal de entrada//
float nivel=0.0;        //Nivel digital escalado a cm.//
int output=0;           //Salida del PID en nivel digital//
float error=0.0;        //Error actual//
float integral=0.0;     //Aproximación discreta de integral (error acumulado)//
int maxOut=255;         //Nivel digital máximo//
int minOut=0;           //Nivel digital mínimo//
bool setpointDefinido=false; //Constante booleana de escritura de setpoint//

//Variables de tiempo//
unsigned long ultimoTiempo=0; //Último conteo de tiempo//
float Tiempo=Ts*1000;        //Tiempo en milisegundos//

//Variables filtrado EMA//
float alpha=0.2;           //Factor de suavizado//
float nivelEMA=0.0;        //Nivel con filtro EMA//

void setup() {
  Serial.begin(9600);       //Iniciar comunicación serial//
  ultimoTiempo = millis();  //Guardar tiempo inicial//
}

void loop() {
  sensor=analogRead(5);     //Asignación pin análogo//
  nivel=(0.1274*sensor)-26.012; //Escalamiento del nivel//
  nivelEMA=alpha*nivel+(1-alpha)*nivelEMA; //Cálculo de nivel con filtro EMA//
```

```

if(Serial.available(>0){           //Primera condición//
    float nuevoSetpoint=Serial.parseFloat();    //Lectura de setpoint deseado desde monitor serie//
    if (nuevoSetpoint>0 && nuevoSetpoint<=30){    //Condición límite de nivel//
        setpoint=nuevoSetpoint;                //Cambio de setpoint a valor ingresado//
        setpointDefinido=true;                  //Cambio de constante booleana//
    }
}

if(setpointDefinido){              //Segunda condición//
    error=setpoint-nivelEMA;        //Cálculo de error//
    integral+=error;                //Cálculo de integral//
    output=((Kp*error)+(Ki*Ts*integral))/5)*255; //Cálculo de output//
    output=constrain(output,minOut,maxOut);    //Limitación de output a valores de saturación//
    if (output>=maxOut){            //Condición salida saturada alta//
        integral=(maxOut-Kp*error)/(Ki*Ts);    //Antiwindup//
    }
    if (output<=minOut){            //Condición salida saturada baja//
        integral=(minOut-Kp*error)/(Ki*Ts);    //Antiwindup//
    }
    output=((Kp*error)+(Ki*Ts*integral))/5)*255; //Actualización de output//
    output=constrain(output,minOut,maxOut);    //Limitación de output actualizado//
}

analogWrite(A0,output);            //Escribir output en pin//

//Tiempo de muestreo//
while(millis()-ultimoTiempo<Tiempo){    //Espera Ts segundos antes de reiniciar loop (muestreo)//
}
ultimoTiempo=millis();                //Actualizar el tiempo//
}

```

ANEXO C: CÓDIGO DE ARDUINO CLOUD PARA CONTROLADOR PID

/*

Sketch generated by the Arduino IoT Cloud Thing "Untitled"

<https://create.arduino.cc/cloud/things/af79bd0d-e652-4166-9a60-7dd91bc6852a>

Arduino IoT Cloud Variables description

The following variables are automatically generated and updated when changes are made to the Thing

float ingreso;

float level;

int actuacion;

Variables which are marked as READ/WRITE in the Cloud Thing will also have functions which are called when their values are changed from the Dashboard.

These functions are generated with the Thing and added at the end of this sketch.

*/

#include "thingProperties.h"

float Kp=3;

float Ki=0.3;

float Ts=0.5;

float setpoint=0.0;

int sensor=0;

float nivel=0.0;

int output=0;

float error=0.0;

float integral=0.0;

int maxOut=255;

int minOut=0;

bool setpointDefinido=false;

unsigned long ultimoTiempo=0;

float Tiempo=Ts*1000;

```

float alpha=0.2;
float nivelEMA=0.0;

void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);
  ultimoTiempo = millis();

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  /*
   The following function allows you to obtain more information
   related to the state of network and IoT Cloud connection and errors
   the higher number the more granular information you'll get.
   The default is 0 (only errors).
   Maximum is 4
  */
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
}

void loop() {
  ArduinoCloud.update();
  // Your code here
  sensor=analogRead(5);
  nivel=(0.1274*sensor)-26.012;
  nivelEMA=alpha*nivel+(1-alpha)*nivelEMA;
  level=nivelEMA;

  if(setpointDefinido){

```

```

error=setpoint-nivelEMA;
integral+=error;
output=(((Kp*error)+(Ki*Ts*integral))/5)*255;
output=constrain(output,minOut,maxOut);
actuacion=output;
if (output>=maxOut){
    integral=(maxOut-Kp*error)/(Ki*Ts);
}
if (output<=minOut){
    integral=(minOut-Kp*error)/(Ki*Ts);
}
output=(((Kp*error)+(Ki*Ts*integral))/5)*255;
output=constrain(output,minOut,maxOut);
actuacion=output;
}

analogWrite(A0,output);

//Tiempo de muestreo//
while(millis()-ultimoTiempo<Tiempo){
}
ultimoTiempo=millis();
}

/*
Since Level is READ_WRITE variable, onLevelChange() is
executed every time a new value is received from IoT Cloud.
*/
void onLevelChange() {
    // Add your code here to act upon Level change
}

/*
Since Actuacion is READ_WRITE variable, onActuacionChange() is
executed every time a new value is received from IoT Cloud.
*/

```

```
void onActuacionChange() {  
    // Add your code here to act upon Actuacion change  
}  
  
/*  
    Since Ingreso is READ_WRITE variable, onIngresoChange() is  
    executed every time a new value is received from IoT Cloud.  
*/  
void onIngresoChange() {  
    // Add your code here to act upon Ingreso change  
    if (ingreso>=0 && ingreso<=30){  
        setpointDefinido=true;  
        setpoint=ingreso;  
    }  
}
```