UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERIA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA MECATRONICA

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"OPTIMIZACIÓN DEL FILTRO DE KALMAN EN SISTEMAS DE CONTROL EN RED SINCRONIZADOS EN ACTUACIÓN NCS"

Trabajo de titulación previo a la obtención del título de Ingeniero en Mecatrónica

Línea de investigación: (Producción industrial y tecnología sostenible)

AUTOR:

Anthony Sebastián Perugachi Chulde

DIRECTOR:

Dr. Carlos Xavier Rosero Chandi

Ibarra – Ecuador 2026

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1004955827		
APELLIDOS Y NOMBRES:	Perugachi Chulde Anthony Sebastián		
DIRECCIÓN:	Calle Polivio Navarrete, Barrio Pucahuaico, San Antonio - Ibarra		
EMAIL:	asperugachic@utn.edu.ec / anthonychulde2002@gmail.com		
TELÉFONO FIJO:		**TELÉFONO MÓVIL:**	0989821991

DATOS DE LA OBRA	
TÍTULO:	"Optimización del filtro de Kalman en Sistemas de Control en Red Sincronizados en Actuación NCS"
AUTOR (ES):	Perugachi Chulde Anthony Sebastián
FECHA: DD/MM/AAAA	06/03/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniero en Mecatrónica
ASESOR /DIRECTOR:	Dr. Carlos Xavier Rosero Chandi MSc. Luz María Tobar Subia Contento

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 6 días del mes de marzo de 2026

EL AUTOR:

Firmado electrónicamente por:
ANTHONY SEBASTIAN PERUGACHI CHULDE
Validar únicamente con FirmaEC

(Firma)……
Nombre: Perugachi Chulde Anthony Sebastián

Universidad Técnica del Norte

Facultad de Ingeniería en Ciencias Aplicadas

Certificación del director del trabajo de grado

En mi calidad de director del trabajo de grado "Optimización del filtro de Kalman en Sistemas de Control en Red Sincronizados en Actuación NCS", presentado por el egresado Anthony Sebastián Perugachi Chulde, que opta por el título de ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, 6 de marzo de 2026

Dr. Carlos Xavier Rosero Chandi

Director de Tesis

Universidad Técnica del Norte

Facultad de Ingeniería en Ciencias Aplicadas

Aprovación del comité calificador

El Comité Calificador del trabajo de Integración Curricular “Optimización del filtro de Kalman en Sistemas de Control en Red Sincronizados en Actuación NCS”, elaborado Anthony Sebastián Perugachi Chulde, previo a la obtención del titulo de ingeniero en Mecatrónica, aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

Ibarra, 6 de marzo de 2026

Dr. Carlos Xavier Rosero Chandi

Director

MSc. Luz María Tobar Subia Contento

Asesor

Dedicatorias

Dedico este trabajo de grado a mis padres, Susana Chulde y José Perugachi. A ellos les debo mi más profundo agradecimiento por los sacrificios realizados a lo largo de estos años, los cuales me brindaron el invaluable regalo de la educación y el apoyo incondicional para alcanzar mis metas.

A mis hermanas, por su cuidado y guía constante, por ser un ejemplo de perseverancia y por enseñarme a seguir mis sueños.

A mi hermano menor, con el anhelo de que pronto alcance también sus propias metas.

Agradecimientos

Al ingeniero Xavier Rosero por su tiempo y paciencia brindada a lo largo del desarrollo de todo este trabajo de grado, ya que con su gran experiencia supo guiarme correctamente para corregir los errores y lograr concluirlo exitosamente.

A los docentes de la carrera de ingeniería en Mecatrónica de la Universidad Técnica del Norte por todos los conocimientos impartidos en cada materia a lo largo de todos los semestres y a mis compañeros de carrera que compartimos momentos de camaradería, pero que al final siempre logramos cumplir con las tareas y proyectos finales.

Índice general

Índice de figuras

Índice de tablas

Resumen

Los Sistemas de Control en Red (NCS, por sus siglas en inglés) constituyen arquitecturas de control distribuido donde los lazos de sensado, control y actuación se cierran a través de canales de comunicación compartidos. Esta configuración introduce desafíos críticos derivados de las latencias variables entre el sensado y la actuación (SAVD) y los tiempos de procesamiento fluctuantes. Para mitigar estos efectos, el presente trabajo analiza una estrategia de control sincronizada en la actuación (NCS-SAA) que emplea un esquema de muestreo aperiódico y actuación periódica, asistida por un estimador de estados basado en el filtro de Kalman.

El objetivo central de esta investigación es evaluar la eficiencia y robustez del filtro de Kalman dentro de un entorno NCS bajo la estrategia de sincronización mencionada. Para ello, se diseña e implementa una plataforma experimental distribuida compuesta por tres nodos de procesamiento basados en microcontroladores Arduino UNO, interconectados mediante una red LAN a través de módulos Ethernet ENC28J60. El desarrollo incluye una versión optimizada del algoritmo de Kalman, adaptada específicamente a las restricciones de memoria y capacidad de cómputo de los microcontroladores de baja gama.

Los resultados experimentales demuestran que la integración del filtro de Kalman adaptativo mejora significativamente la precisión en la estimación de los estados y preserva el desempeño dinámico del sistema frente a las variaciones e incertidumbres de la red. Se concluye que el filtro de Kalman, con las adecuaciones implementadas, representa una solución técnica efectiva y de bajo costo para fortalecer la estabilidad y solidez de los sistemas NCS con sincronización en la actuación.

Palabras clave: Sistemas de Control en Red (NCS), sincronización en la actuación, filtro de Kalman, Arduino, Ethernet.

Abstract

Networked Control Systems (NCS) are distributed control architectures where sensing, control, and actuation loops are closed through shared communication channels. This configuration introduces critical challenges derived from variable sensing-to-actuation delays (SAVD) and fluctuating processing times. To mitigate these effects, this work analyzes a synchronized-actuation control strategy (NCS-SAA) that employs an aperiodic sampling and periodic actuation scheme, assisted by a Kalman Filter-based state estimator.

The main objective of this research is to evaluate the efficiency and robustness of the Kalman Filter within an NCS environment under the aforementioned synchronization strategy. To this end, a distributed experimental platform is designed and implemented, consisting of three processing nodes based on Arduino UNO microcontrollers, interconnected via a LAN network using ENC28J60 Ethernet modules. The development includes an optimized version of the Kalman algorithm, specifically adapted to the memory and computational capacity constraints of resource-limited microcontrollers.

Experimental results demonstrate that the integration of the adaptive Kalman Filter significantly improves state estimation accuracy and preserves the dynamic performance of the system against network variations and uncertainties. It is concluded that the Kalman Filter, with the implemented adaptations, represents an effective and low-cost technical solution to strengthen the stability and robustness of actuation-synchronized NCS.

Keywords: Network Control Systems (NCS), synchronization in actuation, Kalman filter, Arduino, Ethernet.

Capítulo I

Introducción

1.1. Planteamiento del problema

Los sistemas de control en red NCS, constituyen una clase de sistemas de control en los que los lazos de control se cierran mediante una red de comunicación que transmite paquetes de información en tiempo real.

El desempeño de estos sistemas se ve afectado por el tráfico y la latencia inherentes a la red de comunicación, así como por el tiempo de procesamiento de los algoritmos de control. Estos factores introducen retardos variables entre el muestreo y la actuación (Sampling-Actuation Variable Delay o SAVD, por sus siglas en inglés), los cuales comprometen el rendimiento y la estabilidad de los lazos de control. Como respuesta a esta problemática, se ha propuesto un esquema de sincronización en la actuación que emplea muestreos aperiódicos y actuaciones periódicas.

En el trabajo de [1] se presenta un sistema de control distribuido en el que interactúan los nodos de sensor, controlador y actuador. A diferencia de otros enfoques donde el sensado se realiza de forma periódica, este trabajo propone un sensado aperiódico, centrando la sincronización en los períodos de actuación. Esta estrategia permite reducir el efecto de la incertidumbre temporal entre el sensado y la actuación, considerando que se dispone de un observador que, en el instante de muestreo, predice el valor que tendrán las variables de estado en el momento de actuación.

Por su parte, [2] propone la implementación de un filtro de Kalman para realizar la estimación óptima de estados en el instante de sensado. Sin embargo, el análisis desarrollado en dicho trabajo no ha sido validado experimentalmente sobre una plataforma de hardware real. Por esta razón, el presente trabajo propone el diseño y montaje de una plataforma experimental que permita someter este tipo de implementación a condiciones operativas reales.

1.2. Objetivos

1.2.1. Objetivo general

Evaluar el rendimiento computacional del filtro de Kalman en sistemas de control en red sincronizadas en actuación a través de su implementación en un sistema basado en microprocesadores y red de comunicación

1.2.2. Objetivos específicos

- Analizar la metodología de funcionamiento del filtro de Kalman en sistemas NCS, identificando los principales desafíos asociados a su implementación.
- Proponer una metodología de implementación que integre hardware y comunicaciones de forma coherente.
- Validar el sistema propuesto bajo condiciones operativas reales, analizando su desempeño en términos de eficiencia computacional y carga de red.

1.3. Alcance

En el trabajo previo de [1] se desarrolló un modelo matemático que permite la implementación del filtro de Kalman en sistemas NCS, comprobando su efectividad mediante simulación en Simulink. No obstante, en la práctica existen múltiples factores que pueden afectar el comportamiento del modelo, tales como la incertidumbre paramétrica, el ruido en los sensores y las perturbaciones del proceso. Estos elementos introducen incertidumbre adicional al sistema, generando discrepancias entre los resultados de simulación y el comportamiento real.

El sistema propuesto busca desarrollar un algoritmo de implementación mediante el uso de tres nodos distribuidos: sensor, controlador y actuador, cada uno basado en un microcontrolador e interconectados a través de una red de alta velocidad. Además, se implementará el filtro de

Kalman distribuido, sometiéndolo a condiciones reales de operación para evaluar su desempeño experimental.

1.4. Justificación

Desde el punto de vista personal y profesional, esta investigación permite profundizar en el entendimiento de técnicas avanzadas de control y procesamiento de señales, proporcionando las habilidades necesarias para enfrentar desafíos contemporáneos en el campo de la ingeniería. La optimización de algoritmos como el filtro de Kalman en sistemas NCS tiene el potencial de beneficiar directamente a industrias que dependen de sistemas de control precisos y confiables, tales como la robótica, las telecomunicaciones y la automatización industrial.

La elección de esta temática responde a la necesidad de perfeccionar los mecanismos de estimación en sistemas de control en red, aportando conocimientos y técnicas aplicables en múltiples áreas de la ingeniería. Esto contribuye al desarrollo de una sociedad que demanda cada vez mayor eficiencia y precisión en sus sistemas tecnológicos.

Capítulo II

Marco Referencial

En este capítulo se expone el sustento teórico y el estado del arte que fundamentan la presente investigación. Se analizan las particularidades de los sistemas de control en red, las arquitecturas de sincronización en la actuación y la integración del filtro de Kalman como mecanismo de estimación óptima en entornos caracterizados por un muestreo irregular.

2.1. Análisis de la literatura

2.1.1. Sistemas de control en red (NCS)

El surgimiento de los sistemas de control en red (NCS, por sus siglas en inglés) ha transformado la arquitectura del control industrial al permitir la interconexión de sensores, controladores y actuadores mediante canales de comunicación compartidos [3]. No obstante, la incorporación de una red digital rompe con el paradigma del control digital convencional, el cual asume un muestreo estrictamente periódico y una acción de control invariante en el tiempo.

El desempeño de un NCS se encuentra intrínsecamente vinculado a las características del canal de comunicación. En [4] demuestra que las latencias de red degradan la respuesta dinámica del sistema. Mientras que en sistemas de lazo abierto tales retardos suelen ser despreciables, en sistemas de lazo cerrado que dependen de señales de realimentación para la corrección del error la variabilidad temporal de la red compromete la estabilidad. Dado que estos retardos poseen una naturaleza estocástica y fluctúan aleatoriamente, se requiere de metodologías avanzadas para gestionar la incertidumbre y garantizar la convergencia del sistema [5].Figura 2.1

La literatura especializada identifica que la latencia, el tráfico de datos y el tiempo de procesamiento en los nodos introducen retardos variables , denominados comúnmente como retardos variables de sensor a actuador (SAVD, *Sensing-to-Actuation Variable Delays*) [6]. Estos fenó-

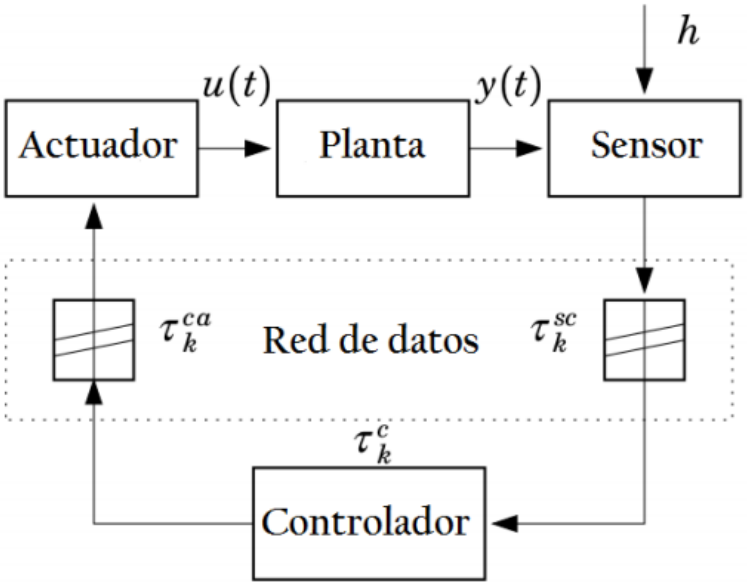

Figura 2.1: Arquitectura común de una NCS

menos provocan una degradación del desempeño que se manifiesta a través de oscilaciones en estado estacionario, errores transitorios y, en escenarios críticos, la inestabilidad global del lazo [7]. Diversas investigaciones, como las desarrolladas por Vascimini et al. [8], proponen estrategias de compensación basadas en predictores para mitigar dichos efectos.

2.1.2. Arquitecturas de sincronización

Las perturbaciones inherentes a un sistema de control automático provocan desviaciones respecto al comportamiento ideal. Por consiguiente, un diseño robusto debe contemplar medidas que aseguren un desempeño satisfactorio incluso ante señales de naturaleza determinística o aleatoria [9]. Tradicionalmente, las estrategias de mitigación se han centrado en compensar los retardos asumiendo que la sincronización ocurre en los instantes de muestreo. Sin embargo, investigaciones recientes [3] proponen un cambio de paradigma hacia la sincronización en los instantes de actuación (NCS-SAA). En esta arquitectura, el período de control h se redefine como el intervalo temporal entre dos acciones de control consecutivas en el actuador, permitiendo que el instante de muestreo sea aperiódico.

Este enfoque exige que el controlador estime el estado de la planta en el futuro instante de actuación, partiendo de la medición disponible y del retardo medido τ_k [10]. Trabajos previos demuestran que esta metodología es capaz de eliminar los efectos nocivos del *jitter* y la incertidumbre temporal al centralizar la base de tiempo del sistema en el nodo actuador [11].

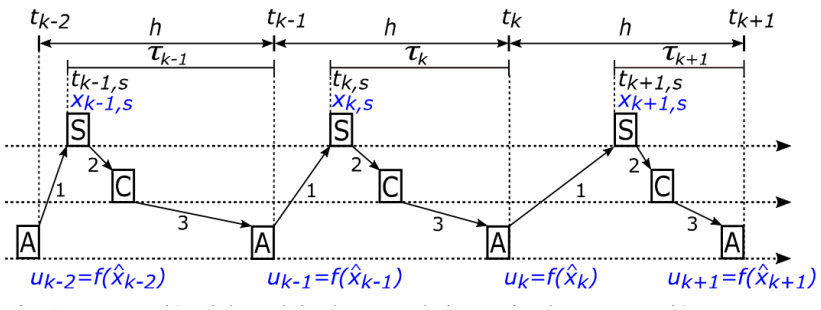

Figura 2.2: Operación del modelo de control sincronizado en actuación

La implementación física de la figura2.2 requiere protocolos de sincronización de reloj de alta precisión, tales como el IEEE 1588 (PTP), para asegurar una referencia temporal común entre los nodos. Se ha reportado que el esquema NCS-SAA es viable incluso en hardware de recursos limitados; por ejemplo, en [12] se presentan validaciones experimentales utilizando microcontroladores de bajo costo operando sobre redes CAN (*Controller Area Network*).

2.1.3. Aplicación del filtro de Kalman en entornos NCS

El filtro de Kalman es un algoritmo basado en el modelo de espacio de estados de un sistema que permite estimar de forma óptima el estado y la salida futura mediante el procesamiento de señales ruidosas. Dependiendo de la naturaleza de los retardos en las muestras de entrada, este puede operar como un estimador de parámetros o estrictamente como un filtro de estados [9].

A pesar de las ventajas operativas de la sincronización en la actuación, los sistemas reales enfrentan desafíos adicionales, tales como el ruido de medición y las perturbaciones del proceso. La integración del filtro de Kalman en sistemas NCS-SAA constituye una solución robusta para optimizar la estimación de estados bajo condiciones de incertidumbre estocástica.

La literatura técnica describe que la aplicación del filtro de Kalman en estos entornos se estructura generalmente en dos etapas: una fase de predicción y corrección que abarca desde la actuación previa hasta el muestreo actual, y una fase de predicción adicional que proyecta el estado desde el instante de muestreo hasta el instante de actuación programado [3].

Estudios comparativos indican que esta combinación no solo minimiza el error cuadrático medio de la estimación, sino que también reduce el esfuerzo de control al suavizar las acciones del actuador frente a señales ruidosas. Las aplicaciones de estas técnicas son críticas en áreas como la telecirugía, el control de vehículos aéreos no tripulados (UAV) y las redes de sensores móviles, donde la integridad de la comunicación es tan fundamental como la precisión algorítmica.

Se ha demostrado que el filtro de Kalman es una herramienta indispensable para inferir información a partir de mediciones indirectas y ruidosas. La contribución del presente trabajo consiste en integrar dicho filtro en el modelo de sincronización NCS-SAA. Cabe destacar que el enfoque estándar del filtro de Kalman discreto asume una periodicidad estricta; no obstante, el modelo NCS-SAA implica mediciones aperiódicas. Esto plantea la necesidad de adaptar el filtro convencional hacia una formulación asíncrona, redefiniendo las fases de predicción y corrección en función del tiempo de llegada de los datos [13].

Capítulo III

Marco Metodológico

En este capítulo se describe el diseño e implementación de un sistema de control en red sincronizado en actuación (NCS-SAA). Se detallan la arquitectura distribuida, el protocolo de sincronización temporal, el modelado matemático de la planta, la formulación del filtro de Kalman con retardo variable y la estrategia de control LQR empleada.

3.1. Antecedentes

El presente trabajo se fundamenta en la infraestructura tecnológica establecida en trabajos previos [2], que demostraron la viabilidad del uso de microcontroladores ATmega328P y comunicación ethernet para aplicaciones de control en red. En comparación con dichas implementaciones, este trabajo introduce las siguientes mejoras

- **Estimación de estados mediante filtro de Kalman:** Se incorpora un esquema de estimación óptima para compensar ruido de medición y perturbaciones del proceso, en reemplazo de enfoques basados únicamente en realimentación directa.
- **Adaptación dinámica del modelo:** Las matrices discretas $A_d(\tau_k)$ y $B_d(\tau_k)$ se recalculan en cada ciclo de control en función del retardo medido, evitando el uso de modelos discretos fijos o buffers de retardos históricos.
- **Sincronización activa periódica:** Se implementa un mecanismo de resincronización cada 1000 ms para compensar la deriva del reloj entre nodos.

Estas mejoras permiten atribuir los cambios de desempeño principalmente a la optimización algorítmica, manteniendo constante la plataforma de hardware.

3.2. Arquitectura del sistema NCS-SAA

Un sistema NCS requiere la ejecución coordinada de operaciones de muestreo y actuación a través de una red de comunicación. La incorporación de dicha red rompe el paradigma del control digital convencional, que asume muestreo estrictamente periódico y actuación invariante en el tiempo. En consecuencia, los retardos variables entre sensado y actuación (SAVD) degradan el desempeño dinámico y pueden comprometer la estabilidad del lazo cerrado.

Para mitigar este efecto, se adopta la arquitectura NCS-SAA propuesta en [6], en la cual el período de control h se define como el intervalo entre dos acciones consecutivas del actuador, permitiendo que el instante de muestreo sea aperiódico. En este esquema, el controlador debe estimar el estado que tendrá la planta en el futuro instante de actuación, a partir de la medición disponible y del retardo medido τ_k.

La asignación de responsabilidades entre nodos sigue el modelo NCS-SAA. Los Algoritmos 1, 2 y 3 describen la lógica de operación de cada nodo.

Algorithm 1 Nodo Actuador

Inicialización: Configurar Timer1 para interrupciones cada $h = 50$ ms.
Inicialización: Configurar Timer2 para PWM a 490 Hz.

while true **do**
    if mensaje UDP disponible del controlador **then**
        $u_k \leftarrow$ leer mensaje.
        Aplicar saturación según Ec. 3.18.
    end if
    if interrupción Timer1 (cada h ms) **then**
        $t_{\text{act},k} \leftarrow$ marca de tiempo local.
        Aplicar u_k a la planta mediante PWM.
        Enviar mensaje UDP $(t_{\text{act},k})$ al sensor.
    end if
    if mensaje UDP de sincronización del sensor **then**
        $t_s \leftarrow$ leer marca de tiempo del sensor.
        $t_a \leftarrow$ marca de tiempo local.
        Enviar mensaje UDP (t_a) al sensor.
    end if
end while

Algorithm 2 Nodo Sensor

Inicialización: Ejecutar sincronización inicial (promedio de $N = 10$ intercambios).
Inicialización: Calcular φ y θ según Ecs. 3.1 y 3.2.

while true **do**
    if mensaje UDP disponible del actuador **then**
        $t_{\text{act},k} \leftarrow$ leer mensaje.
        $y_k \leftarrow$ muestrear salida de la planta (ADC).
        $t_{s,k} \leftarrow$ marca de tiempo local.
        Calcular τ_k según Ec. 3.3.
        Enviar mensaje UDP (y_k, τ_k) al controlador.
    end if
    if temporizador de resincronización **then**
        $t_0 \leftarrow$ marca de tiempo local.
        Enviar solicitud de sincronización al actuador.
        Esperar respuesta del actuador.
        $t_1 \leftarrow$ leer marca de tiempo del actuador.
        $t_2 \leftarrow$ marca de tiempo local.
        Recalcular φ y θ según Ecs. 3.1 y 3.2.
    end if
end while

Algorithm 3 Nodo Controlador

Inicialización: $\hat{x}_{0|0} \leftarrow \mathbf{0}$, $P_{0|0} \leftarrow I$.
Inicialización: Cargar matrices Q, R y ganancia LQR L.
while true **do**
    if mensaje UDP disponible del sensor **then**
        $(y_k, \tau_k) \leftarrow$ leer mensaje.
        $A_d(\tau_k), B_d(\tau_k) \leftarrow$ calcular según Ec. 3.5.
        $\hat{x}_{k|k-1} \leftarrow$ calcular según Ec. 3.6.
        $P_{k|k-1} \leftarrow$ calcular según Ec. 3.7.
        $v_k \leftarrow$ calcular según Ec. 3.8.
        $S_k \leftarrow$ calcular según Ec. 3.9.
        if S_k es invertible **then**
            $K_k \leftarrow$ calcular según Ec. 3.10.
            $\hat{x}_{k|k} \leftarrow$ calcular según Ec. 3.11.
            $P_{k|k} \leftarrow$ calcular según Ec. 3.12.
        else
            $\hat{x}_{k|k} \leftarrow \hat{x}_{k|k-1}$.
            $P_{k|k} \leftarrow P_{k|k-1}$.
        end if
        $u_k \leftarrow$ calcular según Ec. 3.17.
        Aplicar saturación.
        Enviar mensaje UDP (u_k) al actuador.
    end if
end while

La operación del lazo de control se ejecuta mediante tres mensajes UDP por ciclo, como se ilustra en la Figura 3.1. El nodo actuador (A) inicia cada ciclo enviando su marca de tiempo local $t_{\text{act},k}$ al nodo sensor (S). A continuación, el sensor mide la salida de la planta, calcula el retardo τ_k y transmite el paquete $\{y_k, \tau_k\}$ al nodo controlador (C). Finalmente, el controlador ejecuta el filtro de Kalman, calcula la acción de control u_k y la envía al actuador, cerrando el ciclo.

En paralelo, el sensor y el actuador intercambian mensajes de sincronización a una frecuencia aproximada de 1 Hz, independientemente del lazo de control.

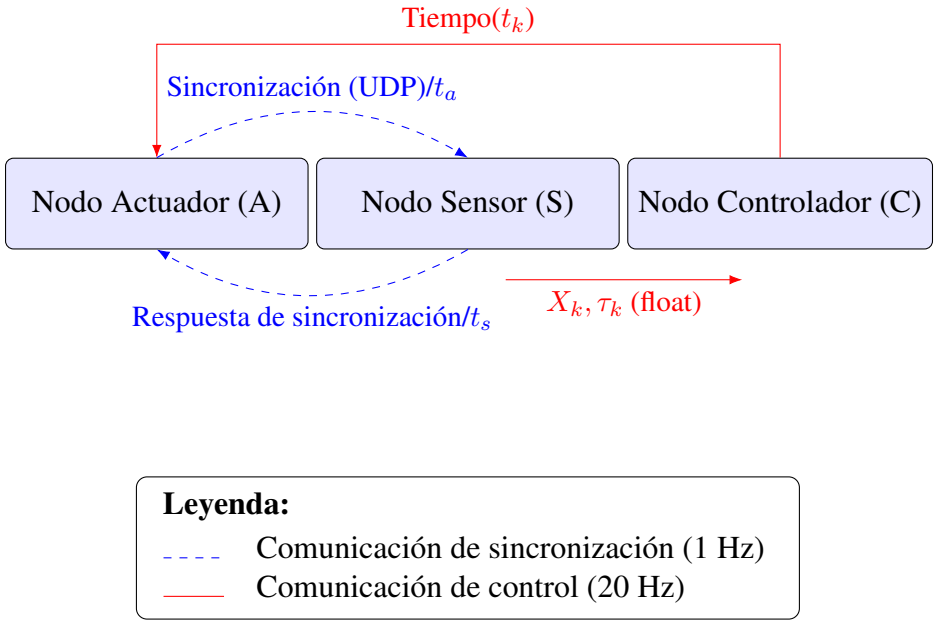

Figura 3.1: Diagrama de red y flujos de comunicación UDP entre nodos

El período de actuación se fijó en $h = 50$ ms mediante temporizadores por hardware en el nodo actuador. Este valor se seleccionó en función de la dinámica de la planta, del tiempo de procesamiento del estimador y de los retardos de comunicación esperados, garantizando la condición $\tau_k < h$ en operación normal.

3.3. Protocolo de sincronización de reloj

La precisión del filtro de Kalman en la arquitectura NCS-SAA depende de la medición del intervalo τ_k, definido como el tiempo transcurrido desde el instante de actuación hasta el instante de sensado dentro del mismo ciclo. Dado que el actuador y el sensor operan con osciladores independientes, se implementó un protocolo de sincronización basado en intercambio de marcas de tiempo (*timestamping*), inspirado en una versión simplificada de IEEE 1588 (PTP).[14]

3.3.1. Estimación de latencia y offset

El proceso de sincronización se define bajo un esquema maestro–esclavo, donde el actuador actúa como maestro y el sensor como esclavo. El procedimiento emplea tres marcas de tiempo

1. t_0: instante en que el sensor envía la solicitud de sincronización.
2. t_1: instante en que el actuador recibe la solicitud y registra su tiempo local.

3. t_2: instante en que el sensor recibe la respuesta del actuador.

Bajo la hipótesis de simetría del canal, la latencia de red φ y el desplazamiento de reloj θ se estiman como

$$\varphi = \frac{t_2 - t_0}{2}, \tag{3.1}$$

$$\theta = t_1 - (t_0 + \varphi)\,. \tag{3.2}$$

Para reducir el efecto del jitter, la sincronización inicial promedia $N = 10$ intercambios, obteniendo estimaciones estables de φ y θ antes de iniciar el lazo de control.

Una vez estimado el offset, el nodo actuador transmite su marca de tiempo local $t_{\text{act},k}$ en cada ciclo. El sensor determina el retardo efectivo en el ciclo k a partir de su reloj local $t_{s,k}$ y del desplazamiento previamente estimado

$$\tau_k = h - (t_{s,k} - t_{\text{act},k} + \theta)\,, \tag{3.3}$$

donde h es el período nominal. Este valor representa el tiempo efectivo durante el cual la planta evolucionó bajo la acción u_{k-1} antes del muestreo, y permite que el controlador actualice el modelo discreto en función del retardo real.

Debido a la deriva de los osciladores, se implementó una sincronización activa periódica: cada 1000 ms, el sensor reevalúa θ mediante un nuevo intercambio de mensajes. Este mecanismo mantiene la incertidumbre en la estimación de τ_k acotada durante la operación continua del sistema.

3.4. Implementación del filtro de Kalman

La estimación de estados se implementó en el nodo controlador para mejorar la robustez del lazo frente al ruido de medición y a la variabilidad del retardo τ_k. El filtro opera en las etapas clásicas de predicción y corrección.

3.4.1. Discretización del modelo de la planta

Para implementar el filtro de Kalman en el nodo controlador, es imperativo transformar el modelo en espacio de estados de tiempo continuo (presentado en la Ec. 4.1) a una representación de tiempo discreto. En el esquema NCS-SAA, esta discretización no es estática, sino que depende del retardo τ_k medido en cada ciclo, el cual define el intervalo de evolución de la planta desde la última acción de control hasta el instante de muestreo.

Partiendo de las matrices continuas del sistema de doble integrador

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \alpha \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \tag{3.4}$$

donde α representa la ganancia del circuito integrador analógico, se aplica la solución exacta de la ecuación de estado para un sistema lineal invariante en el tiempo bajo un retenedor de orden cero (ZOH). Dado que la matriz A es nilpotente de orden 2 ($A^2 = \mathbf{0}$), la matriz de transición de estados $e^{A\tau_k}$ se reduce a los dos primeros términos de la serie de Taylor, facilitando el recálculo en tiempo real dentro del ATmega328P.

Las matrices discretas adaptativas $A_d(\tau_k)$ y $B_d(\tau_k)$, empleadas en la predicción del estado del Algoritmo 3, se definen como

$$A_d(\tau_k) = I + A\tau_k = \begin{bmatrix} 1 & \tau_k \\ 0 & 1 \end{bmatrix}, \quad B_d(\tau_k) = \int_0^{\tau_k} e^{As} B \, ds = \begin{bmatrix} \frac{\alpha\tau_k^2}{2} \\ \alpha\tau_k \end{bmatrix}. \tag{3.5}$$

Esta formulación permite que el controlador actualice su creencia sobre el estado interno de la planta basándose en el tiempo exacto de vuelo del mensaje, eliminando el error de modelado que surgiría al asumir un retardo nominal constante. La Figura 3.2 ilustra la relación temporal entre el periodo de actuación h, el instante de muestreo y el retardo τ_k.

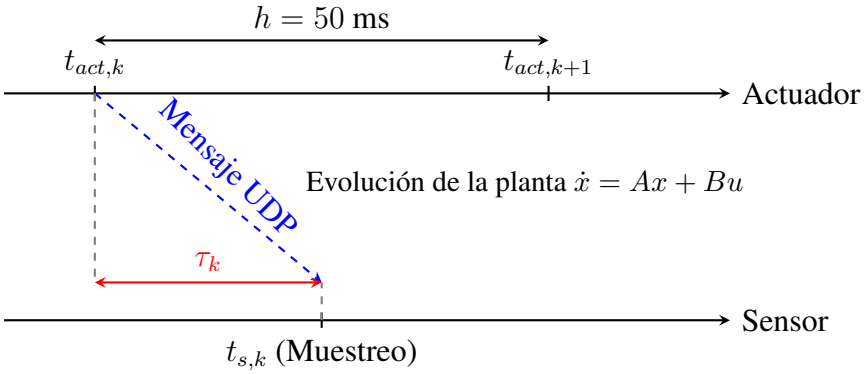

Figura 3.2: Relación temporal entre el evento de actuación y el instante de sensado para la definición de τ_k.

3.4.2. Etapa de predicción

A partir del estado estimado en el ciclo anterior $\hat{x}_{k-1|k-1}$ y de la acción de control aplicada u_{k-1}, se obtiene la estimación *a priori*

$$\hat{x}_{k|k-1} = A_d(\tau_k)\,\hat{x}_{k-1|k-1} + B_d(\tau_k)\,u_{k-1}, \tag{3.6}$$

$$P_{k|k-1} = A_d(\tau_k)\,P_{k-1|k-1}\,A_d^T(\tau_k) + Q, \tag{3.7}$$

donde $P_{k|k-1}$ es la covarianza del error de predicción y Q es la covarianza del ruido de proceso.

3.4.3. Etapa de corrección

Con la medición y_k recibida del sensor, se define la innovación

$$v_k = y_k - C\,\hat{x}_{k|k-1}, \tag{3.8}$$

y su covarianza

$$S_k = C\,P_{k|k-1}\,C^T + R, \tag{3.9}$$

donde R es la covarianza del ruido de medición. La ganancia de Kalman se calcula como

$$K_k = P_{k|k-1}\,C^T\,S_k^{-1}. \tag{3.10}$$

Finalmente, se actualizan el estado estimado y su covarianza

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\,v_k, \tag{3.11}$$

$$P_{k|k} = (I - K_k\,C)\,P_{k|k-1}. \tag{3.12}$$

En la implementación se verifica la invertibilidad de S_k antes de calcular S_k^{-1}. En caso de problemas numéricos, se conserva la estimación *a priori* $(\hat{x}_{k|k-1}, P_{k|k-1})$ para evitar inestabilidad computacional.

3.4.4. Sintonización de covarianzas Q y R

Las matrices de covarianza se seleccionaron mediante iteraciones controladas, evaluando el compromiso entre suavizado de ruido y capacidad de seguimiento de transitorios. Los valores finales empleados fueron.

$$Q = \begin{bmatrix} (4{,}0 \times 10^{-4})^2 & 0 \\ 0 & (5{,}0 \times 10^{-3})^2 \end{bmatrix}, \qquad R = (8{,}5 \times 10^{-5})^2. \tag{3.13}$$

donde R corresponde a la varianza del ruido de medición de la salida $y_k = x_1$. El análisis de desempeño del filtro con estos parámetros se presenta en el Capítulo IV.

3.5. Estrategia de control LQR

La ley de control se diseñó mediante regulación cuadrática lineal (LQR), con el objetivo de minimizar la función de costo

$$J = \sum_{k=0}^{\infty} \left(x_k^T \, Q_c \, x_k + u_k^T \, R_c \, u_k \right), \tag{3.14}$$

donde $Q_c \succeq 0$ pondera el error de estado y $R_c \succ 0$ penaliza el esfuerzo de control. Las matrices de ponderación se seleccionaron como

$$Q_c = \begin{bmatrix} 1 & 0 \\ 0 & 0{,}1 \end{bmatrix}, \qquad R_c = 0{,}5. \tag{3.15}$$

La ganancia óptima L se calculó fuera de línea en MATLAB mediante `lqrd`, resolviendo la ecuación de Riccati algebraica discreta (DARE) asociada al par (A_d, B_d) para un retardo nominal $\tau_{\text{nom}} = 50$ ms. La ganancia se fijó en el firmware del controlador para reducir la carga computacional en tiempo real

$$u_k = -L \, \hat{x}_{k|k}, \qquad L = \begin{bmatrix} 0{,}675 & -1{,}8541 \end{bmatrix}. \tag{3.16}$$

3.5.1. Seguimiento de referencia y saturación

Para permitir el seguimiento de una referencia escalar r_k sobre el primer estado, se incorporó una estructura de seguimiento por realimentación de estado con referencia explícita

$$u_k = -L \left(\hat{x}_{k|k} - x_{\text{ref},k} \right), \tag{3.17}$$

donde $x_{\text{ref},k} = [r_k \;\; 0]^T$. La referencia r_k se definió como una señal cuadrada bipolar de amplitud $\pm 1{,}5$ V y frecuencia 0.2 Hz para las pruebas experimentales descritas en el Capítulo IV.

Con el fin de respetar las limitaciones físicas del actuador, la acción de control se satura antes de ser transmitida por UDP

$$u_k \in [-1{,}65,\ 1{,}65]\ \text{V}. \tag{3.18}$$

Esta medida reduce el riesgo de recorte prolongado y mejora la repetibilidad experimental. El análisis del esfuerzo de control y las saturaciones observadas se presenta en el Capítulo IV.

Capítulo IV

Implementación y pruebas experimentales

Este capítulo detalla la fase de despliegue físico del sistema ncs-saa y la metodología experimental empleada para validar el desempeño del filtro de Kalman adaptativo. Se describe el montaje de la plataforma, la configuración de los nodos de cómputo y la infraestructura de red. Finalmente, se presentan y analizan los resultados obtenidos mediante métricas de error integral y eficiencia computacional, lo que permite cuantificar las mejoras introducidas por la estimación adaptativa en presencia de incertidumbre temporal.

Los datos experimentales se adquieren mediante una aplicación de supervisión desarrollada en Python, la cual establece comunicación con los nodos del sistema a través del protocolo UDP para el registro de variables en tiempo real. El procesamiento de datos, el cálculo de métricas de desempeño y la generación de la visualización gráfica se realizan mediante las bibliotecas científicas NumPy, Pandas y Matplotlib.

4.1. Descripción del entorno experimental

4.1.1. Topología de red y plataforma de hardware

La red de control se configura como una red de área local (lan) aislada mediante un conmutador ethernet de 100 Mbps, con el objetivo de evitar tráfico externo que introduzca retardos no determinísticos. Cada nodo del sistema se implementa sobre un microcontrolador ATmega328P (Arduino UNO R3) que opera a 16 MHz, con conectividad de red provista por módulos ENC28J60 mediante interfaz spi. La tabla 4.1 resume la configuración de direcciones ip y puertos asignados.

Tabla 4.1: Configuración de direcciones ip y puertos de comunicación.

Nodo	Dirección ip	Puerto	Función
sensor	192.168.1.60	8860	adquisición de estados
controlador	192.168.1.20	8820	estimación y control
actuador	192.168.1.40	8840	ejecución de acciones

Las características técnicas de la plataforma son:

- **Procesador:** ATmega328P, arquitectura avr de 8 bits, 16 MHz.
- **Memoria:** 32 KB flash, 2 KB sram, 1 KB eeprom.
- **Controlador ethernet:** ENC28J60, compatible con IEEE 802.3, interfaz spi a 8 MHz.
- **Pila de red:** `UIPEthernet` (implementación ligera de tcp/ip).
- **Protocolo de transporte:** UDP (*user datagram protocol*).

La selección de UDP se justifica por su baja latencia y por la ausencia de mecanismos de retransmisión, propiedades adecuadas para aplicaciones de control en tiempo real donde la información fuera de tiempo carece de valor operativo.

4.1.2. Configuración de red y distribución de nodos

La distribución de responsabilidades de cada nodo se mantiene fiel al diseño metodológico: el nodo sensor realiza la adquisición y el etiquetado temporal, el nodo controlador ejecuta el algoritmo de estimación y la ley lqr, y el nodo actuador aplica la señal de control a la planta. La interconexión física se ilustra en las figuras 4.1 y 4.2.

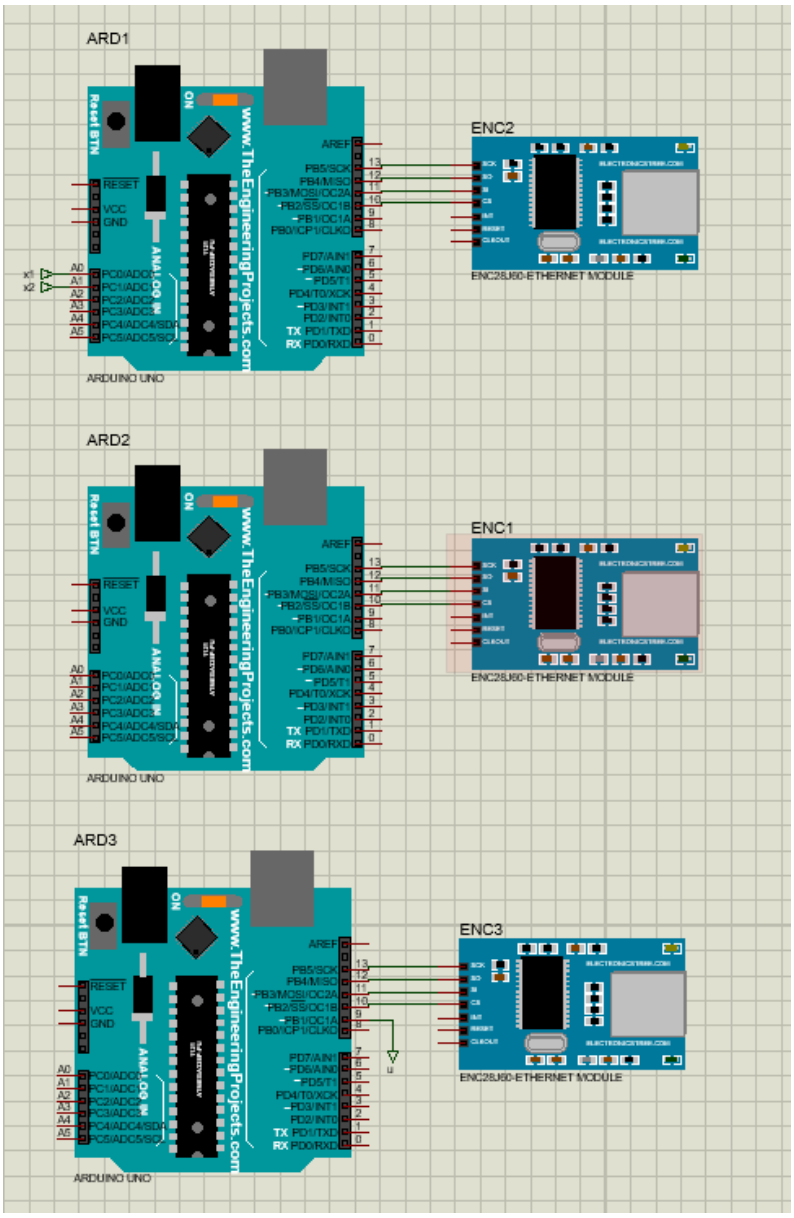

Figura 4.1: Diagrama esquemático del montaje experimental ncs-saa: flujo de datos y arquitectura de red.

Figura 4.2: Implementación física del sistema: nodos sensor, controlador y actuador vinculados mediante un switch ethernet.

4.1.3. Planta de doble integrador

Modelado de la planta y discretización

La planta bajo control corresponde a un doble integrador implementado mediante circuitos analógicos con amplificadores operacionales LM358, cuya representación en espacio de estados en tiempo continuo es

$$\dot{x}(t) = A_c\, x(t) + B_c\, u(t), \tag{4.1}$$

$$A_c = \begin{bmatrix} 0 & -23{,}81 \\ 0 & 0 \end{bmatrix}, \qquad B_c = \begin{bmatrix} 0 \\ -23{,}81 \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{4.2}$$

El vector de estados $x(t) = [x_1(t) \;\; x_2(t)]^T$ representa la salida del primer y segundo integrador, respectivamente. La señal de control $u(t)$ se genera mediante pwm filtrado y se limita al rango $[-1{,}65,\ 1{,}65]$ V para respetar las restricciones del actuador.

La planta consiste en un circuito analógico de doble integración críticamente estable. Esta se implementa mediante amplificadores operacionales de alta impedancia de entrada (LM358), resistencias de precisión con tolerancia del 1 % y capacitores electrolíticos de 100 μF (figura 4.3).

La señal de control u_k se inyecta mediante modulación por ancho de pulso (pwm) con una frecuencia portadora de 490 Hz, la cual es suavizada por un filtro de reconstrucción rc de primer orden con una frecuencia de corte de 15 Hz para eliminar componentes de alta frecuencia. La adquisición de los estados se realiza a través de los convertidores analógico-digitales (adc) de 10 bits del microcontrolador, utilizando una tensión de referencia de 5 V.

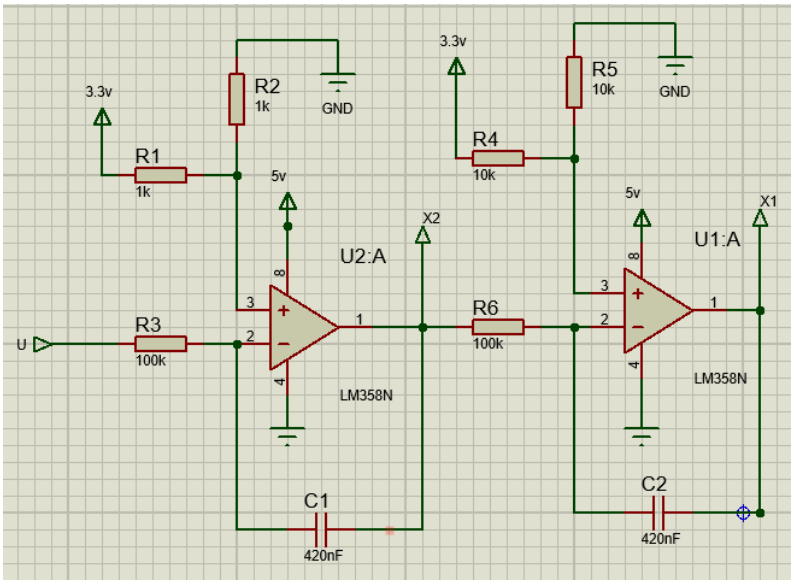

Figura 4.3: Circuito de la planta de doble integrador.

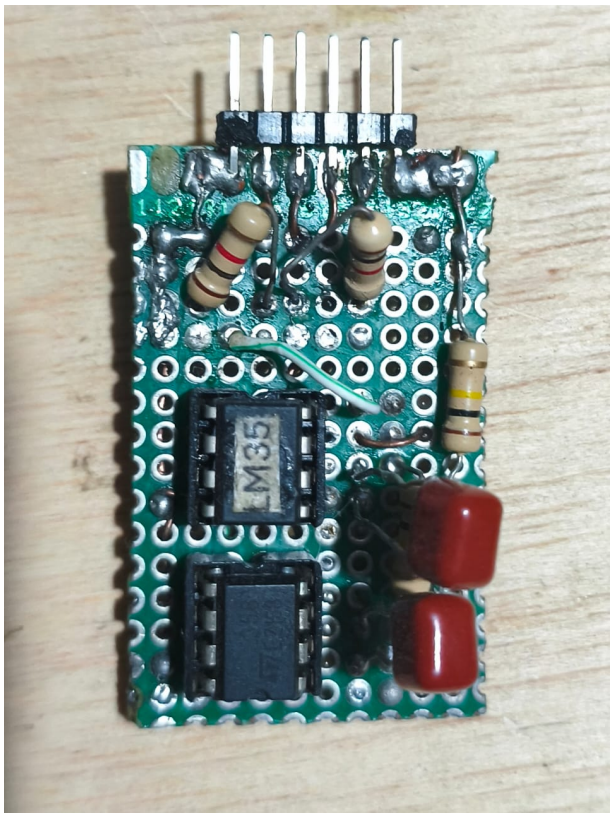

Figura 4.4: Implementación física de la planta de doble integrador con amplificadores LM358.

4.2. Eficiencia computacional en los nodos

La eficiencia computacional se determina midiendo el tiempo de ejecución requerido por el nodo controlador para procesar el algoritmo de estimación y la ley de control lqr. Este parámetro es crítico, ya que el retardo de procesamiento en el controlador contribuye directamente al retardo total del lazo (τ_k).

Tabla 4.2: Tiempos de ejecución promedio en el nodo controlador.

Escenario de prueba	**Tiempo de cómputo promedio (μs)**
control por realimentación directa (sin Kalman)	372.29
ncs-saa con filtro de Kalman (jitter bajo)	561.47
ncs-saa con filtro de Kalman (jitter alto)	2191.51

Como se detalla en la tabla 4.2, el escenario sin filtro de Kalman presenta la menor carga computacional. Al integrar el filtro de Kalman en condiciones de jitter bajo, el tiempo se incrementa a $561{,}47\,\mu s$. En el escenario de alto jitter, el tiempo alcanza los $2191{,}51\,\mu s$ debido a la

naturaleza adaptativa del algoritmo: el microcontrolador recalcula las matrices $A_d(\tau_k)$ y $B_d(\tau_k)$ en cada ciclo según el retardo medido. A pesar de esto, el tiempo total representa solo el $4{,}38\,\%$ del periodo de control nominal ($h = 50\,ms$), lo que valida la capacidad del hardware.

4.2.1. Caracterización de la carga de red y jitter

El desempeño del sistema de comunicación se evalúa mediante el análisis del *jitter* de red, definido como la variación estocástica en los intervalos de llegada de los paquetes entre los nodos.

- **Escenario de bajo jitter:** se observa un promedio de $5{,}78\,ms$. Es un nivel típico en redes locales con tráfico ligero.
- **Escenario de alto jitter:** se induce un jitter promedio de $18{,}24\,ms$, con picos de hasta $25{,}00\,ms$ para estresar el sistema.

El jitter máximo alcanza la mitad del periodo de actuación (h), lo que supone un desafío para el control digital convencional. El sistema utiliza la sincronización activa para medir este retardo y permitir que el controlador compense la latencia mediante el modelo adaptativo del filtro de Kalman.

4.3. Escenarios de prueba y resultados

Para validar la robustez del filtro de Kalman adaptativo, el sistema se somete a cuatro escenarios operativos que representan desde condiciones ideales hasta situaciones críticas de congestión en la red. Estos casos permiten comparar el comportamiento del algoritmo propuesto frente a una implementación convencional sin estimación. En la tabla 4.3 se describen las condiciones específicas de cada escenario.

Tabla 4.3: Descripción de los escenarios de prueba experimentales.

Caso	Descripción	Condición de red	filtro de Kalman
0	simulación ideal	sin jitter / sin ruido	inactivo
1	ncs-saa jitter bajo	jitter promedio 5.78 ms	activo (adaptativo)
2	ncs-saa jitter alto	jitter promedio 18.24 ms	activo (adaptativo)
3	realimentación directa	jitter promedio 7.02 ms	inactivo

El desempeño del sistema en cada escenario se cuantifica mediante dos indicadores técnicos fundamentales:

- **Integral del error absoluto (IAE):** esta métrica se define como la acumulación del valor absoluto del error de seguimiento a lo largo del tiempo, expresada como $IAE = \int |r(t) - y(t)|dt$. Un valor bajo de IAE indica que el sistema sigue la referencia con alta precisión, minimizando tanto las desviaciones en estado estacionario como los errores durante los transitorios.

- **Costo de control:** este índice evalúa la eficiencia y suavidad de la señal de actuación generada por el controlador. Se calcula mediante la suma de las variaciones cuadráticas de la señal de control entre muestras sucesivas: $\sum(u_k - u_{k-1})^2$. Un costo de control elevado indica la presencia de oscilaciones de alta frecuencia o "jitter.[en] el actuador, lo cual es perjudicial para la vida útil de los componentes físicos de la planta.

4.3.1. Caso 0: simulación ideal

Este escenario se realiza en un entorno determinista sin ruido ni jitter. Funciona como la línea base (*benchmark*) para comparar el desempeño del controlador lqr diseñado bajo condiciones perfectas. Los resultados obtenidos en esta etapa se presentan en la figura 4.5.

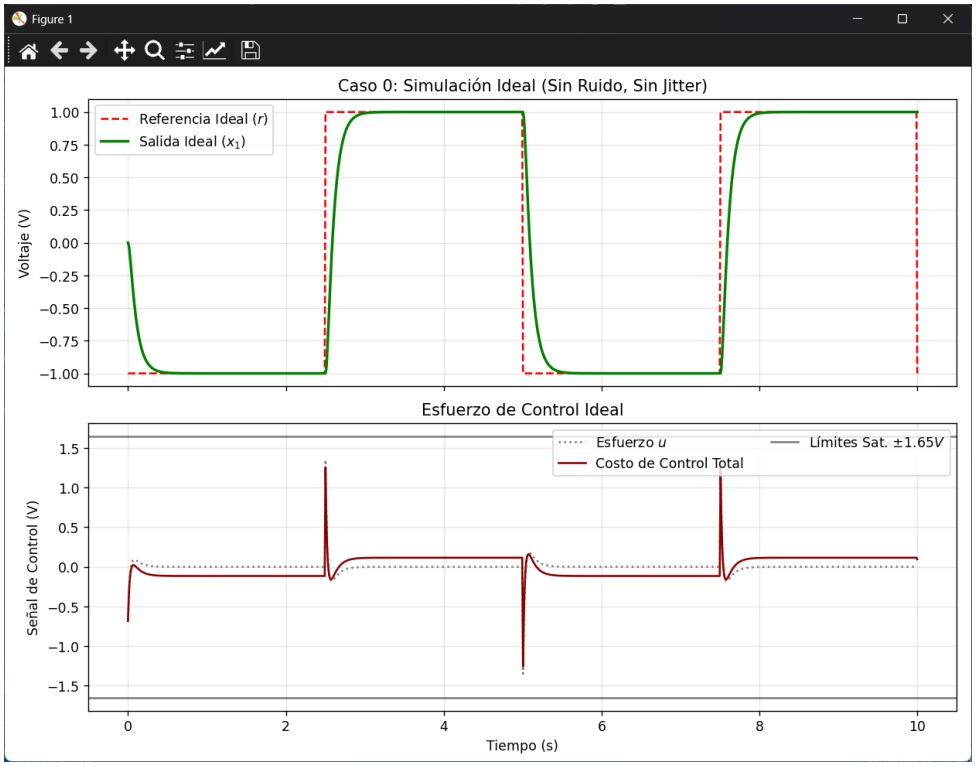

Figura 4.5: Resultados del caso 0: simulación ideal (sin ruido, sin jitter).

Como se observa en la figura 4.5, el seguimiento es preciso con un IAE de $0{,}8275$. El costo de control $(0{,}0953)$ indica un esfuerzo mínimo del actuador al no existir perturbaciones que corregir.

4.3.2. Caso 1: filtro de Kalman con jitter bajo

En esta prueba se utiliza el hardware real bajo una carga de red moderada. Intervienen el ruido electromagnético y la incertidumbre en la llegada de paquetes UDP. El comportamiento dinámico del sistema para este caso se ilustra en la figura 4.6.

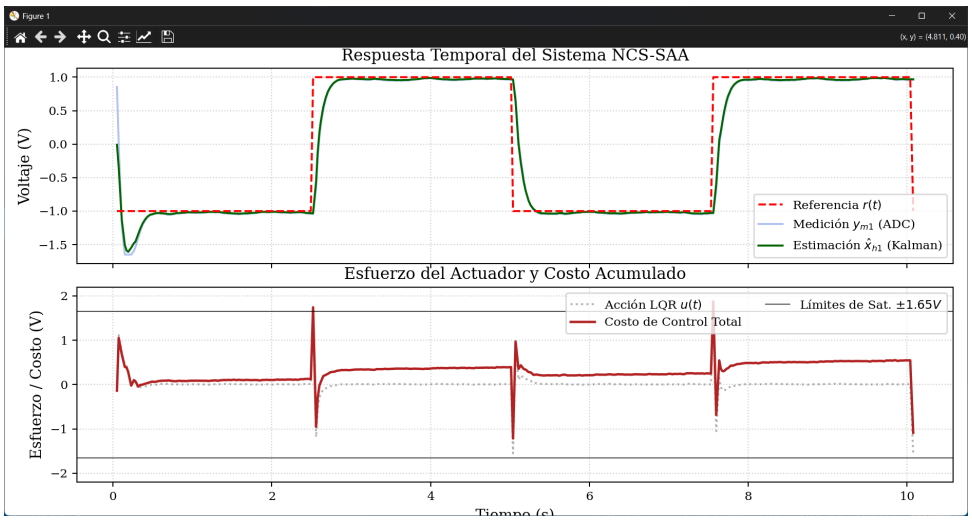

Figura 4.6: Resultados del caso 1: sistema con filtro de Kalman y jitter de $7{,}78\,ms$.

De acuerdo con la figura 4.6, el filtro de Kalman compensa el ruido del sensor eficazmente. El IAE se mantiene en $1{,}0546$, valor cercano al ideal y el costo de control se reduce a $0{,}0861$. El estimador predice la trayectoria de la planta de forma continua, lo que evita que las fluctuaciones de red afecten la señal de actuación.

4.3.3. Caso 2: filtro de Kalman con alto jitter

Se induce un jitter máximo de $25{,}00\,ms$ para evaluar el límite de estabilidad. El retardo de red iguala la mitad del periodo de control nominal, lo que representa una condición crítica. La respuesta del sistema ante este estrés temporal se muestra en la figura 4.7.

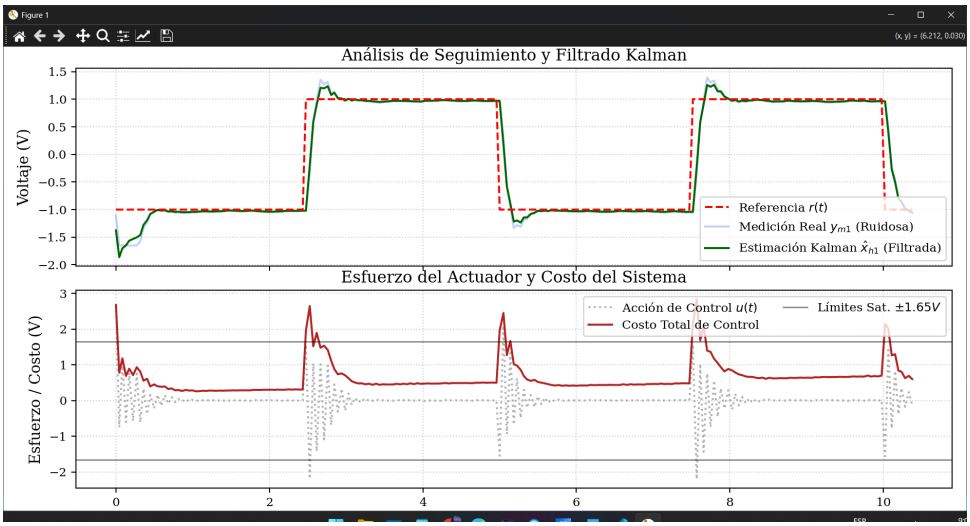

Figura 4.7: Resultados del caso 2: sistema con filtro de Kalman y alto jitter ($24{,}34\,ms$).

Bajo las condiciones observadas en la figura 4.7, el IAE aumenta a $1{,}4778$ y el costo de control sube a $0{,}6019$. Este incremento refleja el esfuerzo del algoritmo para corregir el error causado por los retardos. La actualización dinámica de las matrices de discretización permite conservar la estabilidad transitoria, algo imposible con un controlador de periodo fijo.

4.3.4. Caso 3: realimentación directa

Este escenario sirve como comparación técnica al eliminar la etapa de estimación. El controlador utiliza la lectura directa del sensor para cerrar el lazo bajo un jitter promedio de $7{,}02\,ms$.Los resultados de esta configuración se presentan en la figura 4.8.

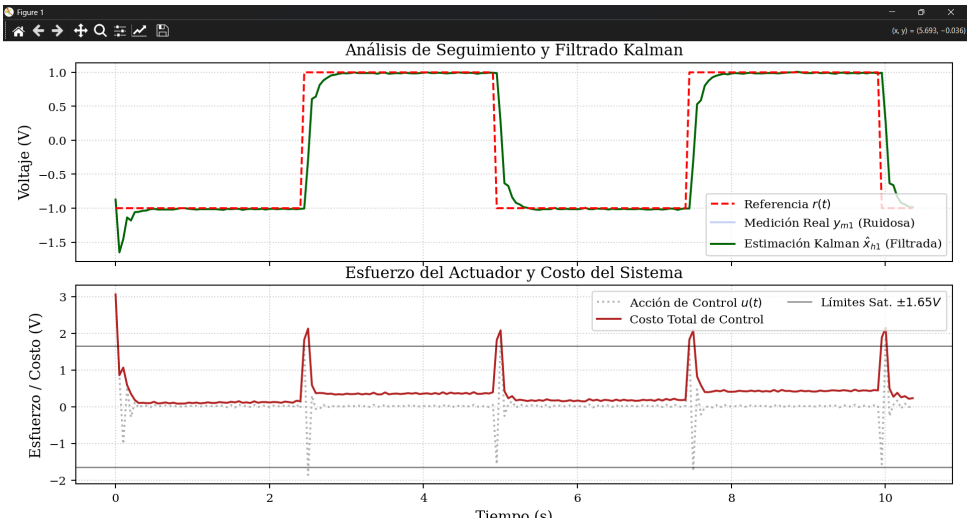

Figura 4.8: Resultados del caso 3: control por realimentación directa (sin filtro de Kalman).

Como se aprecia en la figura 4.8,el ruido del adc se traslada directamente a la señal de control, lo que provoca oscilaciones de alta frecuencia. Aunque el IAE es similar al caso con Kalman bajo, el costo de control se eleva a $0{,}2312$. La ausencia del filtro impide compensar los retardos variables y el ruido, lo que genera un mayor desgaste en los componentes físicos.

4.4. Analisis de robustez

La síntesis comparativa de los escenarios permite validar que el filtro de Kalman adaptativo mejora el desempeño del sistema ncs-saa. La tabla 4.4 resume las métricas para los cuatro casos de estudio.

Tabla 4.4: Comparativa de métricas IAE y costo de control.

Escenario	**IAE**	**Costo de control**
ideal (C0)	0.8275	0.0953
Kalman bajo jitter (C1)	1.0546	0.0861
Kalman alto jitter (C2)	1.4778	0.6019
sin Kalman (C3)	1.0724	0.2312

El filtro de Kalman con jitter moderado (C1) presenta un desempeño muy cercano al ideal (C0). Es notable que el costo de control en el caso 1 sea incluso inferior al de la simulación ideal, lo que demuestra que el filtrado suaviza la señal de actuación de forma más eficiente que el modelo teórico.

En el escenario de alto jitter (C2), se evidencia el éxito de la arquitectura sincronizada (saa). A pesar del retardo extremo, el sistema mantiene la estabilidad transitoria mediante la actualización del modelo en tiempo real. La comparativa con el caso 3 (sin Kalman) revela la necesidad del estimador: sin el filtro, el ruido de los sensores afecta directamente al actuador, lo que incrementa el costo de control de $0{,}0861$ a $0{,}2312$ bajo condiciones similares de red.

Los resultados validan los objetivos planteados:

- se mide y compensa la variabilidad del retardo (τ_k) mediante el protocolo de sincronización.
- la integración de hardware y comunicaciones UDP es coherente y estable.
- el tiempo de cómputo representa solo el $4{,}38\,\%$ del periodo de control, lo que confirma la eficiencia del sistema.

Capítulo V

Conclusiones y Trabajos Futuros

5.1. Conclusiones

Tras el diseño, implementación y validación experimental del sistema de control en red sincronizado en la actuación (NCS-SAA) con filtro de Kalman adaptativo, se presentan las siguientes conclusiones.

El estudio detallado de la metodología permitió identificar que el principal desafío en los sistemas de control en red es la degradación del desempeño causada por los retardos variables entre el sensado y la actuación ($SAVD$ o τ_k). Se concluye que el uso de un modelo discreto dependiente del retardo, integrado en el algoritmo de Kalman, es fundamental para compensar la incertidumbre temporal. A diferencia de los controladores digitales de periodo fijo, la propuesta adaptativa permite al sistema asimilar el *jitter* de red como una variable conocida del modelo, manteniendo la coherencia entre el estado estimado y la acción de control ejecutada.

Se demostró que la arquitectura distribuida basada en tres nodos (sensor, controlador y actuador) interconectados mediante el protocolo UDP sobre ethernet es técnicamente viable y eficiente. El protocolo de sincronización activa desarrollado permitió una medición precisa de la latencia de red en tiempo real, garantizando que el nodo controlador disponga de la información temporal necesaria para ejecutar la fase de predicción del filtro de Kalman. La implementación en microcontroladores de 8 bits validó que una arquitectura coherente de comunicaciones permite el despliegue de algoritmos avanzados de estimación en plataformas de bajo costo.

La validación experimental confirmó la robustez del sistema bajo condiciones críticas de operación. Se concluye que el filtro de Kalman mantiene la estabilidad del lazo cerrado incluso

cuando el *jitter* de red alcanza valores criticos, logrando un índice ISE de $1{,}3205$ frente a la degradación observada en el control por realimentación directa. En términos de eficiencia, el tiempo de cómputo máximo registrado ($2191{,}51\ \mu s$) representa únicamente el $4{,}38\,\%$ del periodo de control disponible, lo que certifica que la solución propuesta es altamente eficiente y no satura los recursos del hardware ni de la red de comunicaciones.

Finalmente, la investigación confirma que la integración de la arquitectura NCS-SAA con un filtro de Kalman adaptativo constituye una solución superior para mitigar los efectos negativos de la incertidumbre estocástica en las redes industriales modernas, optimizando el esfuerzo del actuador y garantizando la precisión en el seguimiento de trayectorias.

5.2. Trabajos futuros

A partir de los hallazgos obtenidos en esta investigación, se abren diversas líneas de desarrollo que permitirán consolidar la robustez de la arquitectura NCS-SAA en entornos industriales de mayor complejidad. En primer lugar, resulta pertinente extender el esquema de estimación adaptativa hacia plantas con dinámicas no lineales o de orden superior, evaluando la implementación de variantes como el filtro de Kalman Extendido (EKF) o el filtro de Kalman Unscented (UKF) para mitigar las incertidumbres de modelado, lo que podría simplificar los protocolos de sincronización actuales y reducir significativamente el tráfico de datos en aplicaciones multiplanta distribuidas

Estas extensiones permitirán consolidar la arquitectura NCS-SAA con filtro de Kalman adaptativo como una solución robusta y escalable para sistemas de control distribuido de bajo costo.

Bibliografía

[1] C. X. Rosero, C. Vaca, M. Gavilanez, I. Iglesias, L. T. Subia, and R. Rosero, "Kalman Filter for Networked Control Systems Synchronized at Actuation Instants," *Proceedings of the 2021 IEEE 5th Colombian Conference on Automatic Control, CCAC 2021*, no. 2, pp. 291–296, 2021.

[2] J. E. Quilumbango Vaca, "Lazos de control en red con sincronización en los instantes de actuación," trabajo de grado (ingeniero en mecatrónica), Universidad Técnica del Norte, Ibarra, Ecuador, mar 2019.

[3] J. Ancizar, C. Cárdenas, M. Antonio, N. Arias, and A. O. Bravo, "ANÁLISIS Y APLICACIÓN DEL FILTRO DE KALMAN A UNA SEÑAL CON RUIDO ALEATORIO Analysis and application of the Kalman filter to a signal with random noise," *Scientia et Technica Año XVIII*, vol. 18, no. 1, 2013.

[4] K. Ogata, *Ingenieria de Control Moderana*, vol. 5. 2010.

[5] V. K. Saini and A. Maity, "Adaptive decentralized kalman filters with non-common states for nonlinear systems," *European Journal of Control*, vol. 63, pp. 70–81, 2022.

[6] Q. Zhao and C. Han, "Optimal control and stabilization for ncs over multi-hop relay networks," in *2024 3rd Conference on Fully Actuated System Theory and Applications (FASTA)*, pp. 60–65, 2024.

[7] B. S. Solanki, "Optimal Control Strategy to Analyse the Networked Control System Stability," *International Journal of Current Science Research and Review*, vol. 07, no. 09, pp. 7160–7167, 2024.

[8] V. G. Vascimini, "Simulations Using the Kalman Filter," no. May, 2020.

[9] J. A. Castañeda Cárdenas, M. A. Nieto Arias, and V. A. Ortiz Bravo, "Análisis y aplicación del filtro de kalman a una señal con ruido aleatorio," *Scientia et Technica*, vol. 18, pp. 267–272, abr 2013.

[10] Y. Tipsuwan and M. Y. Chow, "Control methodologies in networked control systems," *Control Engineering Practice*, vol. 11, pp. 1099–1111, oct 2003.

[11] Á. Cuenca, W. Zhan, J. Salt, J. Alcaina, C. Tang, and M. Tomizuka, "A Remote Control Strategy for an Autonomous Vehicle with Slow Sensor Using Kalman Filtering and Dual-Rate Control," *Sensors*, vol. 19, p. 2983, jul 2019.

[12] C. Lozoya, P. Martí, M. Velasco, and J. M. Fuertes, "Analysis and design of networked control loops with synchronization at the actuation instants," in *The 34th Annual Conference of IEEE Industrial Electronics (IECON)*, (Orlando, FL, USA), pp. 2899–2904, nov 2008.

[13] C. X. Rosero, C. Vaca, I. Iglesias, L. Tobar-Subía, and M. Gavilanez, "On networked control systems under time, measurement, and process uncertainties," *Maskay*, vol. 11, pp. 7–13, nov 2021.

[14] F. A. José Gregorio Díaz, Ana María Mejías, "Aplicacion de los filtros de kalman a sistemas de control," *Revista INGENIERIA UC, vol 8*, vol. 08, no. 01, pp. 1–18, 2001.

Apéndice A

Códigos de Implementación

En este anexo se presentan los algoritmos desarrollados para la implementación del sistema NCS-SAA utilizando la plataforma Arduino. Se detallan los nodos sensor, controlador y actuador.

1.1. Código del Nodo Sensor

```
// ================================================================
// NODO SENSOR - NCS-SAA
// IP: 192.168.1.60
//
// Responsabilidades:
//   1. Escuchar el pulso SYNC del Actuador (inicio de ciclo).
//   2. Al recibir SYNC:
//        - Calcular tau_k = t_llegada_sensor - tA_envio_actuador.
//        - Medir x1 (A1) y x2 (A0) de la planta.
//        - Enviar {x1, x2, tau_k} al Controlador.
//
// LED:
//   Pin 9 -> Parpadea al recibir SYNC y enviar datos.
//
// Entradas Analogicas:
//   A1 -> x1 (Posicion/Voltaje 1)
//   A0 -> x2 (Velocidad/Voltaje 2)
// ================================================================

#include <UIPEthernet.h>
#include <UIPUdp.h>
```

```
22
23 // ---------------------------------------------------------------
24 // CONFIG RED
25 // ---------------------------------------------------------------
26 byte      mac[]          = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0x60 };
27 IPAddress ipSensor       (192, 168, 1, 60);
28 IPAddress ipControlador  (192, 168, 1, 20);
29
30 const uint16_t PORT_SYNC   = 5005;  // Recibe del Actuador
31 const uint16_t PORT_ESTADO = 5006;  // Env a al Controlador
32
33 // ---------------------------------------------------------------
34 // HARDWARE
35 // ---------------------------------------------------------------
36 const int PIN_LED_ACTIVIDAD = 9;
37 const int PIN_X1            = A1;
38 const int PIN_X2            = A0;
39
40 // ---------------------------------------------------------------
41 // PAR METROS DE LA PLANTA (Calibrados previamente)
42 // ---------------------------------------------------------------
43 const float V_REF    = 5.0f;
44 const float V_OFFSET = 1.65f;   // Punto de equilibrio x1_eq [V]
45
46 // ---------------------------------------------------------------
47 // ESTRUCTURAS DE DATOS
48 // ---------------------------------------------------------------
49 struct SyncPacket {
50   unsigned long tA_envio;       // millis() del Actuador al enviar
51 };
52
53 struct SensorPacket {
54   float         x1;             // Medici n x1 (adesviaci n)
55   float         x2;             // Medici n x2 (desviaci n)
56   unsigned long tau_k;          // Retardo A(k-1) -> S(k) [ms]
57 };
58
59 EthernetUDP udpSync;   // Escucha SYNC
60 EthernetUDP udpEstado; // Env a datos al Controlador
61
62 // ---------------------------------------------------------------
63 // PROTOTIPOS
```

```
64 // ----------------------------------------------------------------
65 void enviarDatos(float x1, float x2, unsigned long tau);
66 void pulsarLed(int pin, unsigned long duracion_ms);
67
68 // ================================================================
69 // SETUP
70 // ================================================================
71 void setup() {
72   Serial.begin(115200);
73
74   pinMode(PIN_LED_ACTIVIDAD, OUTPUT);
75   digitalWrite(PIN_LED_ACTIVIDAD, LOW);
76
77   // Inicializar Ethernet
78   Ethernet.begin(mac, ipSensor);
79   udpSync.begin(PORT_SYNC);
80   udpEstado.begin(PORT_ESTADO);
81
82   delay(500);
83   Serial.println(F("[SENSOR] Listo. Esperando SYNC del Actuador..."));
84 }
85
86 // ================================================================
87 // LOOP
88 // ================================================================
89 void loop() {
90   // 1. Escuchar paquete SYNC del Actuador
91   int size = udpSync.parsePacket();
92
93   if (size >= (int)sizeof(SyncPacket)) {
94     unsigned long t_llegada = millis(); // Tiempo local de llegada
95
96     SyncPacket syncPkt;
97     udpSync.read((uint8_t*)&syncPkt, sizeof(SyncPacket));
98
99     // 2. Calcular tau_k (Retardo de red + jitter)
100    //    tau_k = t_llegada_sensor - t_envio_actuador
101    unsigned long tau_k =50;//random(30, 51);// t_llegada - syncPkt.
     tA_envio;
102
103    // 3. Medir la planta inmediatamente
104    //    Convertimos a voltajes relativos al punto de operaci n
```

```
105     float x1_raw = (analogRead(PIN_X1) * V_REF) / 1023.0f;
106     float x2_raw = (analogRead(PIN_X2) * V_REF) / 1023.0f;
107
108     float x1_dev = x1_raw - V_OFFSET;
109     float x2_dev = x2_raw - V_OFFSET; // Offset x2_eq aproximado
110
111     // 4. Enviar datos al Controlador
112     enviarDatos(x1_dev, x2_dev, tau_k);
113
114     // 5. Feedback visual
115     pulsarLed(PIN_LED_ACTIVIDAD, 10);
116
117     // Debug por Serial
118     Serial.print(F("[SENSOR] SYNC Recibido | tau_k="));
119     Serial.print(tau_k);
120     Serial.print(F("ms | x1="));
121     Serial.print(x1_dev, 3);
122     Serial.print(F(" | x2="));
123     Serial.println(x2_dev, 3);
124   }
125 }
126
127 // ================================================================
128 // FUNCIONES
129 // ================================================================
130
131 /**
132 * @brief Empaqueta las mediciones y el retardo tau_k y los env a
133 *        al nodo Controlador por UDP.
134 */
135 void enviarDatos(float x1, float x2, unsigned long tau) {
136   SensorPacket dataPkt;
137   dataPkt.x1    = x1;
138   dataPkt.x2    = x2;
139   dataPkt.tau_k = tau;
140
141   udpEstado.beginPacket(ipControlador, PORT_ESTADO);
142   udpEstado.write((const uint8_t*)&dataPkt, sizeof(SensorPacket));
143   udpEstado.endPacket();
144 }
145
146 /**
```

```
147 * @brief Enciende un LED brevemente.
148 */
149 void pulsarLed(int pin, unsigned long duracion_ms) {
150   digitalWrite(pin, HIGH);
151   delay(duracion_ms);
152   digitalWrite(pin, LOW);
153 }
```

Listing A.1: Implementación del Nodo Sensor en Arduino.

1.2. Código del Nodo Controlador

```
// ================================================================
// NODO CONTROLADOR - NCS-SAA
// IP: 192.168.1.20
//
// Flujo por ciclo k:
//   1. Recibe {x1, x2, tau_k} del Sensor
//   2. Kalman paso 1: predice dt1 = tau_k  (hasta instante de medici n)
//   3. Kalman paso 2: corrige con medici n
//   4. Kalman paso 3: predice dt2 = h-tau_k (hasta fin de ciclo)
//   5. Calcula u_k = LQR sobre estado al fin de ciclo
//   6. Env a u_k al Actuador
//   7. Mide tiempo de c mputo y lo imprime
//
// LED:
//   Pin 8     Parpadea al recibir datos del Sensor y enviar u_k
//
// Serial Plotter (etiquetas):
//   r | x1_med | x1_kal | x2_med | x2_kal | u_k | tau_ms
// ================================================================

#include <UIPEthernet.h>
#include <UIPUdp.h>
#include <BasicLinearAlgebra.h>
using namespace BLA;

// ----------------------------------------------------------------
// CONFIG RED
// ----------------------------------------------------------------
byte      mac[]         = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0x20 };
IPAddress ipControlador (192, 168, 1, 20);
IPAddress ipActuador    (192, 168, 1, 40);

const uint16_t PORT_ESTADO = 5006;  // Sensor        Controlador
const uint16_t PORT_U      = 5007;  // Controlador     Actuador

EthernetUDP udpRx;
EthernetUDP udpTx;

// ----------------------------------------------------------------
// HARDWARE
```

```
41  // ---------------------------------------------------------------
42  const int PIN_LED = 8;
43
44  // ---------------------------------------------------------------
45  // TEMPORIZACI N
46  // ---------------------------------------------------------------
47  const unsigned long H_MS = 50;
48  const float         H_S  = H_MS / 1000.0f;
49
50  // ---------------------------------------------------------------
51  // MODELO DE LA PLANTA
52  // Ac = [0, a; 0, 0],  Bc = [0; a]   a = -23.8095
53  // ---------------------------------------------------------------
54  const float A_PARAM = -23.8095f;
55
56  const Matrix<2,2> C_obs = {1, 0,
57    0, 1};
58
59  // ---------------------------------------------------------------
60  // KALMAN
61  // ---------------------------------------------------------------
62  Matrix<2,1> x_end = {0.0f, 0.0f};  // Estimaci n al FIN del ciclo
     anterior
63  Matrix<2,2> P_end = {sq(2.0e-4f), 0.0f,
64    0.0f,        sq(2.0e-1f)};
65
66  const Matrix<2,2> Q_base = {sq(4.0e-2f), 0.0f,
67    0.0f,        sq(5.0e-3f)};
68
69  const Matrix<2,2> R_noise = {sq(8.5e-3f), 0.0f,
70    0.0f,        sq(9.6e-3f)};
71
72  // ---------------------------------------------------------------
73  // LQR  u = -K1 x1 - K2 x2 + Kr r
74  // ---------------------------------------------------------------
75  const float K1   =  0.668f;
76  const float K2   = -1.8541f;
77  const float Kr   =  0.668f;
78  const float U_SAT = 1.65f;
79
80  float u_prev = 0.0f;
81
```

```
82  // ---------------------------------------------------------------
83  // ESTRUCTURAS UDP
84  // ---------------------------------------------------------------
85  struct SensorPacket {
86    float         x1;
87    float         x2;
88    unsigned long tau_k;
89  };
90
91  struct ControlPacket {
92    float         u_k;
93    unsigned long tC_envio;
94  };
95
96  // ---------------------------------------------------------------
97  // PROTOTIPOS
98  // ---------------------------------------------------------------
99  void  discretizar(float dt, Matrix<2,2> &Ad, Matrix<2,1> &Bd);
100 float referenciaOndaCuadrada();
101 void  pulsarLed(unsigned long ms);
102
103 // ===============================================================
104 // SETUP
105 // ===============================================================
106 void setup() {
107   Serial.begin(115200);
108
109   pinMode(PIN_LED, OUTPUT);
110   digitalWrite(PIN_LED, LOW);
111
112   Ethernet.begin(mac, ipControlador);
113   udpRx.begin(PORT_ESTADO);
114   udpTx.begin(PORT_U);
115
116   delay(500);
117
118   // Cabecera del Serial Plotter (etiquetas fijas)
119   Serial.println(F("t_ms,r,x1_med,x1_kal,x2_med,x2_kal,u_k,tau_ms,proc_us
      ,sat,ref_event,jitter_peak"));
120
121   Serial.println(F("#[CTRL] Listo. Esperando datos del Sensor..."));
122 }
```

```
123
124 // ================================================================
125 // LOOP
126 // ================================================================
127 void loop() {
128   int size = udpRx.parsePacket();
129   if (size < (int)sizeof(SensorPacket)) return;
130
131   //        Marca de tiempo de recepci n
132   unsigned long t_rx = micros();
133
134   //        1. Leer paquete del Sensor
135   SensorPacket sp;
136   udpRx.read((uint8_t*)&sp, sizeof(SensorPacket));
137
138   //        2. Sanitizar tau_k
139   unsigned long tau_ms = sp.tau_k;
140   if (tau_ms >= H_MS) tau_ms = H_MS - 1;  // clamp: m nimo 1ms para dt2
141   if (tau_ms == 0)    tau_ms = 1;          // clamp: m nimo 1ms para dt1
142
143   const float dt1 = tau_ms / 1000.0f;           // tau_k  [s]
144   const float dt2 = (H_MS - tau_ms) / 1000.0f;  // h-tau_k [s]
145
146   //        3. Matrices discretas para dt1 y dt2
147   Matrix<2,2> Ad1, Ad2;
148   Matrix<2,1> Bd1, Bd2;
149   discretizar(dt1, Ad1, Bd1);
150   discretizar(dt2, Ad2, Bd2);
151
152   // Escalar Q proporcional al dt
153   const float qs1 = dt1 / H_S;
154   const float qs2 = dt2 / H_S;
155   const Matrix<2,2> Q1 = Q_base * qs1;
156   const Matrix<2,2> Q2 = Q_base * qs2;
157
```

```
158     //        4. Medici n del Sensor


159     Matrix<2,1> y;
160     y(0,0) = sp.x1;
161     y(1,0) = sp.x2;
162
163     //        5. KALMAN



164
165     // (a) Predicci n hasta instante de medici n (dt1 = tau_k)
166     Matrix<2,1> x_pr1 = Ad1 * x_end + Bd1 * u_prev;
167     Matrix<2,2> P_pr1 = Ad1 * P_end * ~Ad1 + Q1;
168
169     // (b) Correcci n con medici n
170     Matrix<2,1> innov = y - C_obs * x_pr1;
171     Matrix<2,2> S     = C_obs * P_pr1 * ~C_obs + R_noise;
172
173     Matrix<2,1> x_upd = x_pr1;
174     Matrix<2,2> P_upd = P_pr1;
175
176     if (Invert(S)) {
177       Matrix<2,2> K = P_pr1 * ~C_obs * S;
178       x_upd = x_pr1 + K * innov;
179       Matrix<2,2> I; I.Fill(0); I(0,0) = 1; I(1,1) = 1;
180       P_upd = (I - K * C_obs) * P_pr1;
181     }
182
183     // (c) Predicci n hasta fin de ciclo (dt2 = h - tau_k)
184     Matrix<2,1> x_next = Ad2 * x_upd + Bd2 * u_prev;
185     Matrix<2,2> P_next = Ad2 * P_upd * ~Ad2 + Q2;
186
187     //        6. LQR sobre estado al fin de ciclo

188     float r_ref = referenciaOndaCuadrada();
189     float u_k   = -K1 * x_next(0,0) - K2 * x_next(1,0) + Kr * r_ref;
190
191     if (u_k >  U_SAT) u_k =  U_SAT;
192     if (u_k < -U_SAT) u_k = -U_SAT;
193
194     //        7. Enviar u_k al Actuador
```

```
195     ControlPacket cp;
196     cp.u_k      = u_k;
197     cp.tC_envio = millis();
198
199     udpTx.beginPacket(ipActuador, PORT_U);
200     udpTx.write((const uint8_t*)&cp, sizeof(ControlPacket));
201     udpTx.endPacket();
202
203     //        Marca de tiempo de env o
204     unsigned long t_tx  = micros();
205     unsigned long t_proc = t_tx - t_rx;  // Tiempo de c mputo [ s ]
206
207     //        8. Actualizar estado para el pr ximo ciclo
208     x_end  = x_next;
209     P_end  = P_next;
210     u_prev = u_k;
211
212
213     //        9. LED
214     pulsarLed(5);
215
216
217
218     //        10. SERIAL PLOTTER
219     // Formato: etiqueta:valor separados por coma
220     Serial.print(millis());        Serial.print(F(","));
221     Serial.print(r_ref, 3);        Serial.print(F(","));
222     Serial.print(sp.x1, 3);        Serial.print(F(","));
223     Serial.print(x_next(0,0), 3);  Serial.print(F(","));
224     Serial.print(sp.x2, 3);        Serial.print(F(","));
225     Serial.print(x_next(1,0), 3);  Serial.print(F(","));
226     Serial.print(u_k, 3);          Serial.print(F(","));
227     Serial.print(tau_ms , 2);      Serial.print(F(","));
```

```
228     Serial.println(t_proc);
229
230   }
231
232   // ================================================================
233   // FUNCIONES
234   // ================================================================
235
236   /**
237   * @brief Discretiza el modelo continuo con Euler para un dt dado.
238   *        Ad = I + Ac dt ,  Bd     ZOH aproximado
239   */
240   void discretizar(float dt, Matrix<2,2> &Ad, Matrix<2,1> &Bd) {
241     Ad = {1.0f, A_PARAM * dt,
242       0.0f, 1.0f};
243     Bd = {A_PARAM * A_PARAM * dt * dt,
244       A_PARAM * dt};
245   }
246
247   /**
248   * @brief Onda cuadrada bipolar  0 .5V / -1.5V a 0.2 Hz (T=5s).
249   */
250   float referenciaOndaCuadrada() {
251     const unsigned long T_MEDIO_MS = 2500;
252     static bool          nivel_alto = true;
253     static unsigned long t_cambio   = 0;
254
255     if (millis() - t_cambio >= T_MEDIO_MS) {
256       nivel_alto = !nivel_alto;
257       t_cambio   = millis();
258     }
259     return nivel_alto ? 0.5f : -0.5f;
260   }
261
262   /**
263   * @brief Pulso breve en el LED indicador.
264   */
265   void pulsarLed(unsigned long ms) {
266     digitalWrite(PIN_LED, HIGH);
267     delay(ms);
268     digitalWrite(PIN_LED, LOW);
```

```
269 }
```

Listing A.2: Implementación del Algoritmo de Control y Filtro de Kalman.

1.3. Código del Nodo Actuador

```
// ================================================================
// NODO ACTUADOR - NCS-SAA
// IP: 192.168.1.40
//
// Responsabilidades:
//   1. Generar el tick h = 50 ms (maestro de tiempo)
//   2. Al inicio de cada ciclo:
//        - Aplicar u_k pendiente al pin 9 (PWM)
//        - Enviar SYNC con timestamp al Sensor
//   3. Durante el ciclo:
//        - Escuchar u_k del Controlador
//        - Al recibirlo, almacenarlo para el siguiente tick
//
// LEDs:
//   Pin 5     Marca el tick h (enciende al inicio de cada ciclo)
//   Pin 8     Parpadea al RECIBIR u_k del Controlador
//           Parpadea al APLICAR u_k al inicio del ciclo
//
// Acci n de control:
//   Pin 9     PWM hacia la planta (u_k)
// ================================================================

#include <UIPEthernet.h>
#include <UIPUdp.h>

// ----------------------------------------------------------------
// CONFIG RED
// ----------------------------------------------------------------
byte      mac[]      = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0x40 };
IPAddress ipActuador (192, 168, 1, 40);
IPAddress ipSensor   (192, 168, 1, 60);

const uint16_t PORT_SYNC = 5005;  // Actuador     Sensor
const uint16_t PORT_U    = 5007;  // Controlador     Actuador

// ----------------------------------------------------------------
// HARDWARE
// ----------------------------------------------------------------
const int PIN_LED_RITMO    = 5;   // Marca el periodo h
const int PIN_LED_ESTADO   = 8;   // Recepci n / aplicaci n de u_k
```

```
41  const int PIN_CONTROL      = 9;   // Salida PWM hacia la planta
42
43  // ---------------------------------------------------------------
44  // TEMPORIZACI N
45  // ---------------------------------------------------------------
46  const unsigned long H_MS = 50;    // Periodo del ciclo [ms]
47
48  // ---------------------------------------------------------------
49  // ESTRUCTURA del paquete SYNC  (Actuador     Sensor)
50  // Incluye el timestamp del Actuador para que el Sensor
51  // pueda calcular       = t_llegada_sensor - tA_envio
52  // ---------------------------------------------------------------
53  struct SyncPacket {
54    unsigned long tA_envio;   // millis() del Actuador al enviar SYNC
55  };
56
57  // ---------------------------------------------------------------
58  // ESTRUCTURA del paquete u_k  (Controlador     Actuador)
59  // ---------------------------------------------------------------
60  struct ControlPacket {
61    float         u_k;        // Acci n de control calculada
62    unsigned long tC_envio;   // Timestamp del Controlador (diagn stico)
63  };
64
65  // ---------------------------------------------------------------
66  // ESTADO INTERNO
67  // ---------------------------------------------------------------
68  float         u_pendiente  = 0.0f;   // u_k a aplicar en el pr ximo tick
69  bool          u_disponible = false;  // Indica si lleg  u_k nuevo este
     ciclo
70  unsigned long t_tick_prev  = 0;      // Timestamp del  ltimo  tick h
71
72  EthernetUDP udpSync;   // Socket para enviar SYNC al Sensor
73  EthernetUDP udpU;      // Socket para recibir u_k del Controlador
74
75  // ---------------------------------------------------------------
76  // PROTOTIPOS
77  // ---------------------------------------------------------------
78  void  enviarSync(unsigned long tA);
79  void  recibirU();
80  void  aplicarControl(float u);
81  void  pulsarLed(int pin, unsigned long duracion_ms);
```

```
82
83  // ================================================================
84  // SETUP
85  // ================================================================
86  void setup() {
87    Serial.begin(115200);
88
89    // Configurar pines
90    pinMode(PIN_LED_RITMO,  OUTPUT);
91    pinMode(PIN_LED_ESTADO, OUTPUT);
92    pinMode(PIN_CONTROL,    OUTPUT);
93    digitalWrite(PIN_LED_RITMO,  LOW);
94    digitalWrite(PIN_LED_ESTADO, LOW);
95    digitalWrite(PIN_CONTROL,    LOW);
96
97    // Inicializar red
98    Ethernet.begin(mac, ipActuador);
99    udpSync.begin(PORT_SYNC);
100   udpU.begin(PORT_U);
101
102   delay(500);
103   Serial.println(F("[ACTUADOR] Listo."));
104   Serial.println(F("[ACTUADOR] h=50ms | PIN_CONTROL=9 | LED_RITMO=5 |
      LED_ESTADO=8"));
105
106   // Marcar inicio del primer ciclo
107   t_tick_prev = millis();
108 }
109
110 // ================================================================
111 // LOOP
112 // ================================================================
113 void loop() {
114   unsigned long t_actual = millis();
115
116   //        TICK h: inicio de nuevo ciclo


117   if (t_actual - t_tick_prev >= H_MS) {
118     t_tick_prev = t_actual;
119
120     // 1) Marcar ritmo h     LED pin 5
```

```
121       pulsarLed(PIN_LED_RITMO, 10);
122
123       // 2) Aplicar u_k pendiente a la planta     pin 9
124       if (u_disponible) {
125         aplicarControl(u_pendiente);
126         u_disponible = false;
127         Serial.print(F("[ACTUADOR] u_k aplicado: "));
128         Serial.println(u_pendiente, 4);
129       } else {
130         // ZOH: si no lleg  u_k nuevo, mantener el anterior
131         Serial.println(F("[ACTUADOR] ZOH: manteniendo u_k anterior"));
132       }
133
134       // 3) Confirmar aplicaci n     LED pin 8
135       pulsarLed(PIN_LED_ESTADO, 10);
136
137       // 4) Enviar SYNC con timestamp al Sensor
138       //    El Sensor usar  tA_envio para calcular
139       enviarSync(t_actual);
140       Serial.print(F("[ACTUADOR] SYNC enviado | tA="));
141       Serial.println(t_actual);
142     }
143
144     //        RECEPCI N CONTINUA de u_k del Controlador
145     //    Se ejecuta en cada iteraci n del loop (no bloqueante)
146     recibirU();
147   }
148
149   // ================================================================
150   // FUNCIONES
151   // ================================================================
152
153   /**
154   * @brief Env a el paquete SYNC al Sensor con el timestamp actual
155   *        del Actuador. El Sensor lo usar  para calcular      .
156   * @param tA Timestamp del Actuador en el momento del env o [ms].
157   */
158   void enviarSync(unsigned long tA) {
159     SyncPacket pkt;
160     pkt.tA_envio = tA;
161
```

```
162     udpSync.beginPacket(ipSensor, PORT_SYNC);
163     udpSync.write((const uint8_t*)&pkt, sizeof(SyncPacket));
164     udpSync.endPacket();
165   }
166
167   /**
168   * @brief Escucha el socket PORT_U de forma no bloqueante.
169   *        Si llega un paquete con u_k, lo almacena como pendiente
170   *        para aplicarlo en el pr ximo tick h.
171   *        Enciende LED pin 8 al recibir.
172   */
173   void recibirU() {
174     int size = udpU.parsePacket();
175     if (size >= (int)sizeof(ControlPacket)) {
176       ControlPacket pkt;
177       udpU.read((uint8_t*)&pkt, sizeof(ControlPacket));
178
179       u_pendiente  = pkt.u_k;
180       u_disponible = true;
181
182       // LED pin 8: confirmaci n de recepci n
183       pulsarLed(PIN_LED_ESTADO, 10);
184
185       Serial.print(F("[ACTUADOR] u_k recibido: "));
186       Serial.print(pkt.u_k, 4);
187       Serial.print(F(" | tC="));
188       Serial.println(pkt.tC_envio);
189     }
190   }
191
192   /**
193   * @brief Aplica la acci n de control u a la planta v a PWM en pin 9.
194   *        Mapea u     [-1.65, +1.65] V al rango PWM [0, 168]
195   *        centrado en el punto de operaci n (U_OFFSET = 1.65 V).
196   *        Usa OCR1AL directamente para mayor precisi n temporal.
197   * @param u Acci n de control en voltios (desviaci n respecto al offset)
      .
198   */
199   void aplicarControl(float u) {
200     const float U_OFFSET = 1.65f;
201     const float V_MAX    = 3.3f;
202
```

```
203     // Saturar
204     if (u >  V_MAX / 2.0f) u =  V_MAX / 2.0f;
205     if (u < -V_MAX / 2.0f) u = -V_MAX / 2.0f;
206
207     // Mapear a rango [0, 168] del Timer1 (8-bit, prescaler=8)
208     float pwm_val = (u / (V_MAX / 2.0f)) * 84.0f + 84.0f;
209     if (pwm_val > 168.0f) pwm_val = 168.0f;
210     if (pwm_val <   0.0f) pwm_val =   0.0f;
211
212     // Configurar Timer1 para PWM ~7.8 kHz
213     TCCR1A = (1 << COM1A1) | (1 << WGM10);
214     TCCR1B = (1 << WGM12)  | (1 << CS11);
215
216     OCR1AL = (unsigned char)(pwm_val);
217   }
218
219   /**
220   * @brief Enciende un LED durante duracion_ms milisegundos y lo apaga.
221   * @param pin         Pin del LED.
222   * @param duracion_ms Tiempo encendido en ms.
223   */
224   void pulsarLed(int pin, unsigned long duracion_ms) {
225     digitalWrite(pin, HIGH);
226     delay(duracion_ms);
227     digitalWrite(pin, LOW);
228   }
```

Listing A.3: Implementación del Nodo Actuador.