COMPARACIÓN DE CONTROLADORES AVANZADOS BASADO EN ARDUINO DE UN MOTOR DC

Alfonso Boceta Lasarte
Araceli Baena Baena
Antonio Bustos López

Curso 2024/2025

Escuela Técnica Superior de
INGENIERÍA DE SEVILLA

Índice

1. Introducción

Para el proyecto de la asignatura *Laboratorio de Control*, se diseñaron e implementaron tres tipos de controladores distintos:

- **Controlador Predictivo Basado en Modelo (MPC)**.
- **Controlador Adaptativo**.
- **Controlador PID en Cascada**.

Estos controladores fueron implementados en una plataforma *Arduino Due* con el propósito de controlar el motor DC Feedback 33-100, un dispositivo ampliamente empleado en las prácticas de la asignatura. El desarrollo incluyó tanto el diseño teórico como la implementación práctica de los algoritmos de control, evaluando su desempeño en el contexto del sistema real.

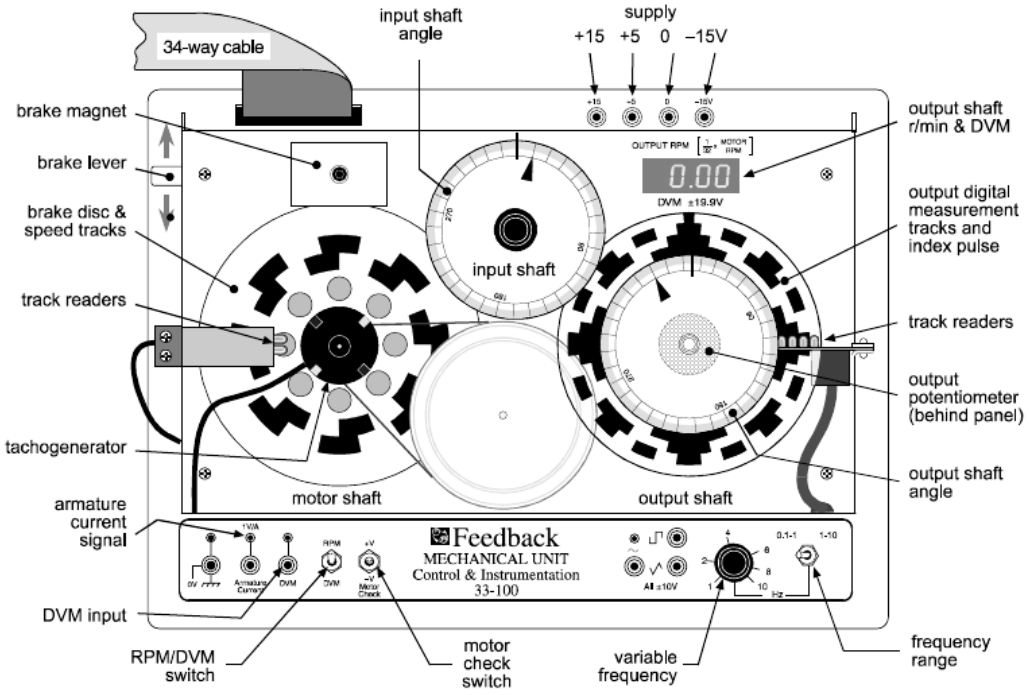

Figura 1: Motor Feedback 33-100

Para ello, se han diseñado dos referencias dinámicas (una correspondiente a la posición angular del motor y otra correspondiente a la velocidad angular) para estos controladores.De esta manera, se comparará el desempeño de los controladores avanzados, que deberán tratar de controlar ambas referencias de manera simultánea.

2. Modelo del Motor Empleado

Para diseñar controladores avanzados, necesitamos un modelo matemático del motor DC que describa su comportamiento dinámico. Este modelo puede ser expresado de dos formas:

- En espacio de estados, que describe las variables internas del sistema.
- En función de transferencia, que relaciona directamente la entrada y salida del sistema.

Espacio de estados

El modelo en espacio de estados está definido por las siguientes ecuaciones diferenciales lineales en forma matricial:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{1}$$
$$y(t) = Cx(t) + Du(t) \tag{2}$$

En este caso, las matrices se definen de la siguiente manera:

$$A = \begin{bmatrix} 0 & K_2 \\ 0 & -\frac{1}{\tau} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{K_1}{\tau} \end{bmatrix}, \quad C = I_2, \quad D = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

Donde:

- A: Matriz de estado
- B: Matriz de entrada
- C: Matriz de salida
- D: Matriz directa

Dado que el controlador se implementará en un sistema digital (Arduino), es necesario discretizar el sistema continuo. Esto se hace usando el método de transformación de espacio de estados con un tiempo de muestreo previamente calculado.En este caso, se realizó en MATLAB de manera similar a la práctica 5 de la asignatura, siendo las matrices resultantes las siguientes:

$$G = A_d, \quad H = B_d, \quad C_d = C, \quad D_d = D$$

Función de transferencia

Este modelo también se puede expresar en términos de su función de transferencia. Este es el diagrama de bloques del sistema donde u es la señal de control, ω es la velocidad angular del motor y θ es la posición angular.

Figura 2: Diagrama de bloques del sistema

Cálculo de parámetros

Los parámetros identificados del sistema se han calculado siguiendo el modelo de primer orden representado en la Figura 3, donde se muestra la respuesta de un sistema excitado por una señal cuadrada.

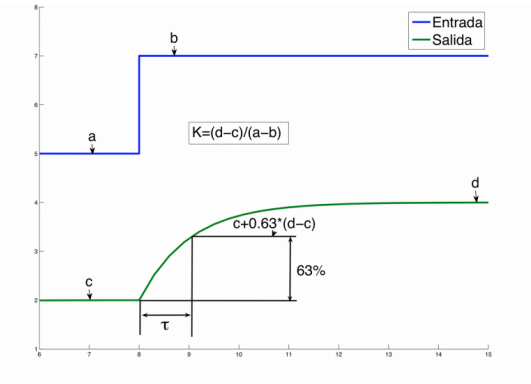

Figura 3: : Modelo de primer orden de un sistema excitado por una señal cuadrada

Constante de ganancia estática (K):

El parámetro K se calcula con la siguiente fórmula:

$$K = \frac{d-c}{a-b} \tag{3}$$

Donde:

- a y b son los valores de la señal de entrada en su estado inicial y final.
- c y d son los valores de la señal de salida en su estado inicial y final respectivamente.

Constante de tiempo (τ):

La constante de tiempo τ se determina como el tiempo necesario para que la respuesta del sistema alcance el 63 % de la diferencia entre su estado inicial (c) y su estado estacionario final (d). Este valor está marcado en la figura como el punto donde la salida alcanza el valor:

$$c + 0{,}63 \times (d-c) \tag{4}$$

La constante K2 se calcula a partir de la pendiente de la entrada de posición frente a una velocidad angular constante.

El resultado de los parámetros es el siguiente:

$$K_1 = 0.328, \quad K_2 = 9.24, \quad \tau = 0.13\,\text{s}, \quad T_s = \frac{\tau}{20} = 0.0065\,\text{s}$$

Nota: Los datos se obtuvieron en el laboratorio durante la realización de la segunda práctica de la asignatura.

3. Descripción del Controlador Avanzado desarrollado

3.1. Controlador PID en Cascada

Se ha implementado un controlador PID en cascada para controlar el servomotor mediante dos lazos:

Lazo externo (PID de posición)

Este lazo toma como entrada la referencia de posición y la posición actual del sistema. Calcula el error de posición y ajusta la salida para generar una referencia de velocidad, que sirve de entrada al lazo interno.

$$\text{Ref\_vel\_PID} = Kp \cdot \left(\text{error\_pos} + \frac{1}{Ti} \cdot \int \text{error\_pos}\ dt\ + Td\frac{\text{d(error\_pos)}}{dt}\right) \tag{5}$$

Lazo interno (PID de velocidad)

El lazo interno toma la referencia de velocidad generada por el lazo externo y la compara con la velocidad actual del sistema. Calcula el error de velocidad y genera la señal de control u, que se aplica al sistema.

$$\text{u} = Kp \cdot \left(\text{error\_vel} + \frac{1}{Ti} \cdot \int \text{error\_vel}\ dt\ + Td\frac{\text{d(error\_vel)}}{dt}\right) \tag{6}$$

La estructura en cascada permite desacoplar las dinámicas del sistema. El PID interno, más rápido, corrige los errores de velocidad, mientras que el PID externo, más lento, se enfoca en llevar la posición a la referencia deseada.

3.2. Controlador Adaptativo

El control adaptativo es un tipo de control para sistemas dinámicos el cual se caracteriza por el ajuste dinámico de las constantes de control. Es especialmente útil para sistemas cuyos parámetros cambian con el tiempo.

En general se acepta que el control adaptativo es un tipo de control no lineal en el que el estado del proceso puede ser separado en dos escalas de tiempo que evolucionan a diferente velocidad. La escala lenta corresponde a los cambios en los parámetros del regulador y la escala rápida a la dinámica del bucle ordinario de realimentación. En este caso se definen solo dos parámetros: la ganancia de realimentación K y la ganancia de alimentación directa L.

La evolución de estos parámetros, viene dictadas por las siguientes ecuaciones:

$$K(k+1) = K(k) + \gamma_K \cdot e(k) \cdot x(k)^T \tag{7}$$

$$L(k+1) = L(k) + \gamma_L \cdot e(k) \cdot r(k)^T \tag{8}$$

Donde:

- K, L son los vectores de ganancias de realimentación y de alimentación directa para el instante anterior respectivamente.
- γ_K, γ_L son los *learning rates* para la ganancia de realimentación y la ganancia de alimentación respectivamente.
- $e(k)$ es el vector de error de posición y velocidad.
- $x(k)$ es el vector del estado del sistema.
- $r(k)$ es el vector de referencia.

Como se puede observar la evolución de estas constantes viene dictada por sus valores previos, así como del error; del estado x, (para K) o del valor de la referencia, r, (para L), además de los *learning rates*: γ_K, γ_L. Estos últimos son los que dictan la velocidad de variación de las constantes.

Se define la acción de control para este control adaptativo como:

$$u(k) = -K \cdot x(k) + L \cdot r(k) \tag{9}$$

Siendo $u(k)$ la entrada de control al sistema.

Control adpatativo: iterantes iniciales y velocidad de adaptación

Tal y como se ha explicado previamente, los *learning rates* indican la velocidad de adaptación de los parámetros adapatativos K y L, por lo que unos valores iniciales de estas pueden ser especialmente importantes para *learning rates* reducidos, puesto que los valores de las ganancias dependerán mayoritariamente de los valores que tenían en la iteración anterior.

3.3. Controlador Predictivo basado en Modelo (MPC)

El *Control Predictivo Basado en Modelo* (MPC, por sus siglas en inglés) es una metodología avanzada de control que utiliza un modelo matemático del sistema para predecir su comportamiento futuro y determinar las acciones de control óptimas. Este enfoque es particularmente útil para sistemas multivariables, como aquellos donde la posición y la velocidad deben ser controladas de manera simultánea y precisa, como es el caso de este proyecto.

El diseño del MPC está basado en la representación del sistema en espacio de estado, cuyas matrices ya han sido descritas anteriormente. Las ecuaciones que describen el sistema son:

$$x(k+1) = Ax(k) + Bu(k) \tag{10}$$
$$y(k) = Cx(k) + Du(k) \tag{11}$$

Donde:

- $x(k)$ es el vector de estados. Incluye las variables de posición y velocidad.
- $u(k)$ es el vector de las entradas de control.
- $y(k)$ representa las salidas observables del sistema (posición y velocidad buscadas).
- A, B, C, y D son las matrices que describen el sistema.

El MPC usa el modelo matemático para predecir el comportamiento de nuestro sistema sobre un horizonte de predicción definido N. Aquí se calcula la evolución futura de los estados y salidas en función de las acciones de control. Estas predicciones permiten anticipar el impacto de las decisiones de control actuales en el rendimiento futuro del sistema.

La función objetivo del MPC se diseña para minimizar el error entre las salidas predichas y las referencias deseadas de posición y velocidad. Una función clásica es de la forma:

$$J = \sum_{i=1}^{N} \left(\|y_{\text{pos}}(k+i) - r_{\text{pos}}(k+i)\|^2 + \|y_{\text{vel}}(k+i) - r_{\text{vel}}(k+i)\|^2 \right) + \lambda \sum_{i=1}^{N} \|u(k+i)\|^2 \tag{12}$$

Donde:

- r_{pos} y r_{vel} son las referencias deseadas para posición y velocidad.
- λ es un peso que penaliza el esfuerzo de control para evitar acciones abruptas.

El MPC optimiza las acciones de control $u(k)$ respetando restricciones específicas del sistema, tales como:

- Los límites físicos en la posición y la velocidad (calculados en las prácticas de la asignatura).

- Las restricciones en las entradas de control.

Este enfoque asegura que el sistema opere bajo condiciones seguras mientras se está optimizando el rendimiento.

El controlador MPC tiene una serie de ventajas muy útiles para el desarrollo de este proyecto:

- **Control Multivariable:** El MPC gestionará de manera simultánea múltiples variables, en este caso posición y velocidad, considerando sus interacciones.
- **Capacidad Predictiva:** Permite anticipar y minimizar las perturbaciones antes de que impacten negativamente en el sistema.
- **Adaptabilidad:** Al recalcular las acciones de control en cada instante, se adapta dinámicamente a cambios en el sistema o en las referencias deseadas.

4. Esquema del Software y Estructura de Control

4.1. Diseño analítico en *MATLAB*

Antes de proceder con la programación del microcontrolador Arduino, se llevaron a cabo simulaciones detalladas en *MATLAB*. Este paso permitió evaluar el desempeño de los controladores y las referencias diseñadas, asegurando su correcto funcionamiento antes de implementarlos físicamente.

Se estudió la estabilidad, controlabilidad y observabilidad del sistema en espacio de estados obtenido del motor. Para ello, se comprobó en MATLAB usando los comandos correspondientes (queda patente en el anexo de códigos). El sistema resultó ser **inestable**, aunque **controlable** y **observable**. Esto pone de manifiesto la necesidad de diseñar un controlador para garantizar su estabilidad, que es precisamente el objetivo de este trabajo.

La estabilidad del sistema se evaluó mediante los valores propios de la matriz A. Un sistema es estable si y solo si la parte real de todos los valores propios es negativa:

$$\mathrm{Re}(\lambda_i) < 0 \quad \forall i.$$

La controlabilidad se verificó utilizando la matriz de controlabilidad:

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix}.$$

El sistema es controlable si el rango de $\mathcal{C}$ es igual al número de estados del sistema.

La observabilidad se comprobó a través de la matriz de observabilidad:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}.$$

El sistema es observable si el rango de $\mathcal{O}$ es igual al número de estados.

El uso de MATLAB para las simulaciones ofreció varias ventajas significativas:

- **Identificación de problemas:** Las simulaciones permitieron detectar posibles problemas en los controladores y en las referencias antes de su implementación física, lo que ahorró tiempo y recursos.
- **Optimización de parámetros:** La plataforma de MATLAB facilitó la modificación y ajuste de los parámetros del sistema, logrando una configuración más precisa y efectiva de los controladores.

Estas simulaciones sirvieron como un paso fundamental en el proceso de desarrollo, asegurando que las etapas posteriores de implementación se llevaran a cabo con un menor riesgo de errores y con controladores bien optimizados.

4.2. Implementación en *ARDUINO*

Una vez programados y simulados los controladores en MATLAB, se procedió a implementarlos en Arduino. Durante este proceso surgieron diversos problemas al trasladar las funcionalidades de MATLAB a código en lenguaje C.

- **Cálculo de matrices:**En primer lugar, el cálculo de matrices en C es considerablemente más complejo que en MATLAB, lo que introdujo desafíos en términos de precisión numérica. Se observó que pequeñas modificaciones en los valores de las matrices podían provocar errores significativos en el comportamiento del sistema, probablemente debido a problemas de condicionamiento numérico o desbordamiento de datos.
- **Optimizador usado:** En segundo lugar, el controlador predictivo requería el uso del optimizador `quadprog` en MATLAB, que resultó difícil de implementar en Arduino debido a las limitaciones de memoria y capacidad de procesamiento del microcontrolador. Para superar esta dificultad, se optó por sustituir `quadprog` por un optimizador más sencillo basado en el método de descenso de gradiente.

Afortunadamente, esta solución funcionó de manera adecuada en Arduino y fue capaz de proporcionar los valores necesarios dentro del tiempo requerido para el control en tiempo real.

Realizada la estructura general del código (Ver Anexos A-C), se implementaron todos los controladores previamente simulados en MATLAB (además del PID en cascada, que se consideró que no merecía la pena ser previamente escrito en MATLAB, pues su utilidad es meramente comparativa con respecto a los otros controladores).

Algorithm 1 Cálculo de matrices Φ y Γ (necesarias para el cálculo del MPC)

Input: Matrices A_d, B_d, C_d, horizontes N_p, N_u
Output: Matrices Φ, Γ
procedure CALCULARMATRICES
  for $i \leftarrow 0$ to $N_p - 1$ **do**
    for $j \leftarrow 0$ to 2 **do**
      $\Phi[i \cdot 2 + j][0] \leftarrow C_d[j][0] \cdot A_d[0][0]^i$
      $\Phi[i \cdot 2 + j][1] \leftarrow C_d[j][1] \cdot A_d[1][1]^i$
    end for
    for $j \leftarrow 0$ to $N_u - 1$ **do**
      if $i \geq j$ **then**
        $\Gamma[i \cdot 2][j] \leftarrow C_d[0][0] \cdot A_d[0][0]^{i-j} \cdot B_d[0]$
        $\Gamma[i \cdot 2 + 1][j] \leftarrow C_d[1][1] \cdot A_d[1][1]^{i-j} \cdot B_d[1]$
      end if
    end for
  end for
end procedure

Figura 4: Cálculo de matrices Φ y Γ

Algorithm 2 Controlador MPC

Input: Matrices Φ, Γ, parámetros N_p, N_u, pesos w_x, restricciones u_{min}, u_{max}
Output: Señal de control u
procedure BLOQUECONTROLMPC
  Leer estado actual $x[0], x[1]$
  Generar referencias ref_pos, ref_vel
  Crear vector x_ref con ref_pos y ref_vel
  Calcular matrices H y f:
  for i, j en N_u **do**
    $H[i][j] \leftarrow \sum_k \Gamma[k][i] \cdot w_x \cdot \Gamma[k][j]$
  end for
  for i en N_u **do**
    $f[i] \leftarrow \sum_k \Gamma[k][i] \cdot w_x \cdot (\Phi[k][0] \cdot x[0] + \Phi[k][1] \cdot x[1] - x\_ref[k])$
  end for
  Inicializar u_opt en ceros
  for iter en max_iters **do**
    Calcular gradiente $grad[i] \leftarrow f[i] + \sum_j H[i][j] \cdot u\_opt[j]$
    Actualizar $u\_opt[i] \leftarrow \text{constrain}(u\_opt[i] - \alpha \cdot grad[i], u_{min}, u_{max})$
  end for
  Aplicar $u \leftarrow u\_opt[0]$
end procedure

Figura 5: Controlador MPC

Algorithm 3 Controlador PID en cascada

Input: Parámetros Kp, Ti, Td, restricciones u_{min}, u_{max}, v_{min}, v_{max}
Output: Señal de control u
procedure BLOQUECONTROLPIDCASCADA
  Leer estado actual pos_actual, vel_actual
  Generar referencias ref_pos, ref_vel
  PID de posición:
  $error\_pos \leftarrow ref\_pos - pos\_actual$
  $integral\_pos \leftarrow integral\_pos + error\_pos \cdot (Ts/1000)$
  $derivada\_pos \leftarrow (error\_pos - error\_pos\_prev)/(Ts/1000)$
  $error\_pos\_prev \leftarrow error\_pos$
  $ref\_vel\_PID \leftarrow Kp\_pos \cdot (error\_pos + (1/Ti\_pos) \cdot integral\_pos + Td\_pos \cdot derivada\_pos)$
  $ref\_vel\_PID \leftarrow \text{constrain}(ref\_vel\_PID, v_{min}, v_{max})$
  PID de velocidad:
  $error\_vel \leftarrow ref\_vel\_PID - vel\_actual$
  $integral\_vel \leftarrow integral\_vel + error\_vel \cdot (Ts/1000)$
  $derivada\_vel \leftarrow (error\_vel - error\_vel\_prev)/(Ts/1000)$
  $error\_vel\_prev \leftarrow error\_vel$
  $u \leftarrow Kp\_vel \cdot (error\_vel + (1/Ti\_vel) \cdot integral\_vel + Td\_vel \cdot derivada\_vel)$
  $u \leftarrow \text{constrain}(u, u_{min}, u_{max})$
  Aplicar control u
end procedure

Figura 6: Controlador PID en Cascada

Algorithm 4 Controlador Adaptativo

Input: Matrices A_d, B_d, C_d, ganancias K, L, parámetros γ_k, γ_l, λ, restricciones u_{min}, u_{max}
Output: Señal de control u
procedure BLOQUECONTROLADAPTATIVO
  Leer estado actual $x[0], x[1]$
  Generar referencias ref_pos, ref_vel
  Crear vector de referencia $r[0], r[1]$
  Calcular error $e[0], e[1]$
$e[i] \leftarrow r[i] - (C_d[i][0] \cdot x[0] + C_d[i][1] \cdot x[1])$
  Calcular señal de control:
$u \leftarrow -(K[0] \cdot x[0] + K[1] \cdot x[1]) + (L[0] \cdot r[0] + L[1] \cdot r[1])$
  $u \leftarrow \text{constrain}(u, u_{min}, u_{max})$
  Actualizar estado del sistema:
$x\_new[0] \leftarrow A_d[0][0] \cdot x[0] + A_d[0][1] \cdot x[1] + B_d[0] \cdot u$
$x\_new[1] \leftarrow A_d[1][0] \cdot x[0] + A_d[1][1] \cdot x[1] + B_d[1] \cdot u$
  Restringir $x\_new[i]$ al rango permitido y actualizar $x[i]$
  Actualizar ganancias adaptativas:
  for $i \leftarrow 0$ to 1 **do**
    $K[i] \leftarrow K[i] + \gamma_k \cdot (e[i] \cdot x[i]) - \lambda \cdot K[i]$
    $L[i] \leftarrow L[i] + \gamma_l \cdot (e[i] \cdot r[i]) - \lambda \cdot L[i]$
    Restringir $K[i]$ y $L[i]$ al rango permitido
  end for
  Aplicar control u
end procedure

Figura 7: Controlador Adaptativo

5. Procedimiento de ajuste del Controlador Desarrollado

El ajuste de los controladores desarrollados busca optimizar el rendimiento del sistema mediante la configuración adecuada de los parámetros. Para ello, se han empleado diferentes métodos de ajuste según el tipo de controlador: el método de Ziegler-Nichols para el PID en cascada y un método de búsqueda sistemático denominado *Grid Search* para el Control Predictivo basado en Modelo (MPC) y el Controlador Adaptativo. Este último método consiste en explorar un conjunto de combinaciones de parámetros para encontrar la configuración óptima.

Ajuste del Controlador PID en Cascada

El ajuste del controlador PID en cascada se realizó utilizando la metodología de Ziegler-Nichols. Se empleó el método de bucle abierto para el control PID en velocidad y el método de bucle cerrado para el control PID en posición, según la siguiente tabla:

PID	K_c	T_i	T_d
B.A.	$1.2/a$	$2L$	$L/2$
B.C.	$0.6\ K_u$	$0.5\ T_u$	$0.125\ T_u$

Figura 8: Sintonización de un PID según Ziegler-Nichols

- **Parámetros del PID para el control en velocidad**

$$K = \frac{Y_{rp}}{U_{rp}} = \frac{1.8}{6} = 0.3$$

$$\tau = 1.5 \times (t_{63} - t_{28}) = 1.5 \times (0.12 - 0.07) = 0.075$$

$$L = t_{63} - \tau = 0.12 - 0.075 = 0.045$$

$$a = \frac{K \times L}{\tau} = \frac{0.3 \times 0.045}{0.075} = 0.18$$

Los parámetros del PID en bucle abierto son:

$$K_c = \frac{1.2}{a} = 6.67$$

$$T_i = 2 \times L = 0.09$$

$$T_d = \frac{L}{2} = 0.0225$$

- **Parámetros del PID para el control en posición**

$$K_u = 15 \quad \text{y} \quad T_u = 0.95$$

Los parámetros del PID en bucle cerrado son:

$$K_c = 0.6 \times K_u = 9$$

$$T_i = 0.5 \times T_u = 0.475$$

$$T_d = 0.125 \times T_u = 0.11875$$

Nota: Los datos se obtuvieron en el laboratorio durante la realización de la segunda práctica de la asignatura.

Ajuste del Control Predictivo basado en Modelo

El ajuste del Control Predictivo basado en Modelo (MPC) se realizó utilizando el método de *Grid Search*. Se ajustaron los siguientes parámetros:

- Horizonte de predicción (Np): Define el número de pasos futuros considerados para la predicción del comportamiento del sistema.
- Horizonte de control (Nu): Especifica el número de pasos futuros en los que se permite la optimización del control.
- w_x: Penalización del error de posición.
- w_v: Penalización del error de velocidad.
- w_u: Penalización del esfuerzo de control.
- α: Tasa de aprendizaje.

Los rangos de valores explorados fueron los siguientes:

$$\begin{aligned} N_p &\in \{5, 10, 15\} \\ N_u &\in \{3, 5, 7\} \\ w_x, w_v &\in \{1, 5, 10, 20, 50, 100\} \\ w_u &\in \{0.01, 0.1, 1\} \\ \alpha &\in \{0.001, 0.005, 0.01, 0.05, 0.1\} \end{aligned}$$

Posteriormente, se realizaron simulaciones en MATLAB para cada combinación de parámetros. En cada iteración, se calcularon los errores cuadráticos medios (MSE) para posición y velocidad.

Como criterio de selección, se utilizó la métrica del error total, calculada como:

$$MSE_{\text{total}} = MSE_{\text{pos}} + MSE_{\text{vel}} \tag{13}$$

Solo se consideraron configuraciones que cumplieran con las restricciones de control.

Finalmente, se generó un listado con las mejores combinaciones de parámetros, ordenadas según el menor error total obtenido.

Combinación óptima

Los mejores resultados se lograron con la siguiente combinación:

$$N_u = 3, \quad N_p = 10, \quad w_x = 100, \quad w_v = 20, \quad w_u = 0.1$$

Con un error total de 2.31.

Ajuste del Controlador Adaptativo

Para el ajuste del Controlador Adaptativo se utilizó de nuevo el método de *Grid Search*. Se ajustaron los siguientes parámetros:

- K: Ganancia inicial de retroalimentación.
- L: Ganancia inicial de alimentación directa.
- α: Rango de las tasas de aprendizaje o *learning rates*.

Los rangos de valores explorados fueron los siguientes:

$$\begin{aligned} K &\in \{0.01, 0.05, 0.1, 0.5, 1, 10, 100\} \\ L &\in \{0.01, 0.05, 0.1, 0.5, 1, 10, 100\} \\ \alpha &\in \{0.001, 0.005, 0.01, 0.05, 0.1\} \end{aligned}$$

De la misma manera que se hizo con el ajuste del controlador anterior, se realizaron simulaciones en MATLAB para cada combinación de parámetros. En cada iteración, se calcularon los errores cuadráticos medios (MSE) para posición y velocidad.

Como criterio de selección, se volvió a utilizar la métrica del error total y solo se consideraron configuraciones que cumplieran con las restricciones de control.

Finalmente, se generó un listado con las mejores combinaciones de parámetros, ordenadas según el menor error total obtenido.

Combinación óptima

Los mejores resultados se lograron con la siguiente combinación:

$$K = 0.5, \quad L = 0.05$$

6. Referencias de posición y velocidad implementadas

Para estudiar el comportamiento de los distintos controladores se han empleado hasta 3 tipos de referencias:

- **Referencia sencilla:** Un vector de referencias con una componente senoidal y otra cosenoidal. Esta referencia fue la que se utilizó durante la mayor parte de los experimentos.

$$r_{pos} = A \cdot \sin(2\pi f t_{\text{cycle}})$$
$$r_{vel} = 2\pi f A \cdot \cos(2\pi f t_{\text{cycle}})$$

- **Referencia compuesta:** Un vector de referencias con una componente senoidal con armónicos y su derivada.

$$r_{\text{pos}}(x) = \left(\sin(x) + \frac{\cos(3x)}{2}\right) + \left(\sin(x) + \frac{\cos(3x)}{2}\right)$$
$$r_{\text{vel}}(x) = 2\cos(x) - 3\sin(3x)$$

- **Referencia mixta:** Un vector de referencias con una componente con tramos senoidales, lineales y constantes y su derivada.

$$r_{\text{pos}} = \begin{cases} \sin(2x) & \text{si } 0 \leq x < \frac{\pi}{2} \\ x - \frac{\pi}{2} & \text{si } \frac{\pi}{2} \leq x < \pi \\ -\sin(x) & \text{si } \pi \leq x < \frac{3\pi}{2} \\ -\frac{x}{2} + \frac{3\pi}{4} & \text{si } \frac{3\pi}{2} \leq x < 2\pi \end{cases}$$

$$r_{\text{vel}} = \begin{cases} 2\cos(2x) & \text{si } 0 \leq x < \frac{\pi}{2} \\ 1 & \text{si } \frac{\pi}{2} \leq x < \pi \\ -\cos(x) & \text{si } \pi \leq x < \frac{3\pi}{2} \\ -0{,}5 & \text{si } \frac{3\pi}{2} \leq x < 2\pi \end{cases}$$

Nota: Es de especial importancia el que la segunda componente del vector de referencias (referencia de velocidad) sea la derivada de la primera componente de este vector, es decir, de la referencia en posición, para así mantener la coherencia del sistema.

7. Resultado del Sistema de Control

7.1. Control predictivo: Referencia sencilla

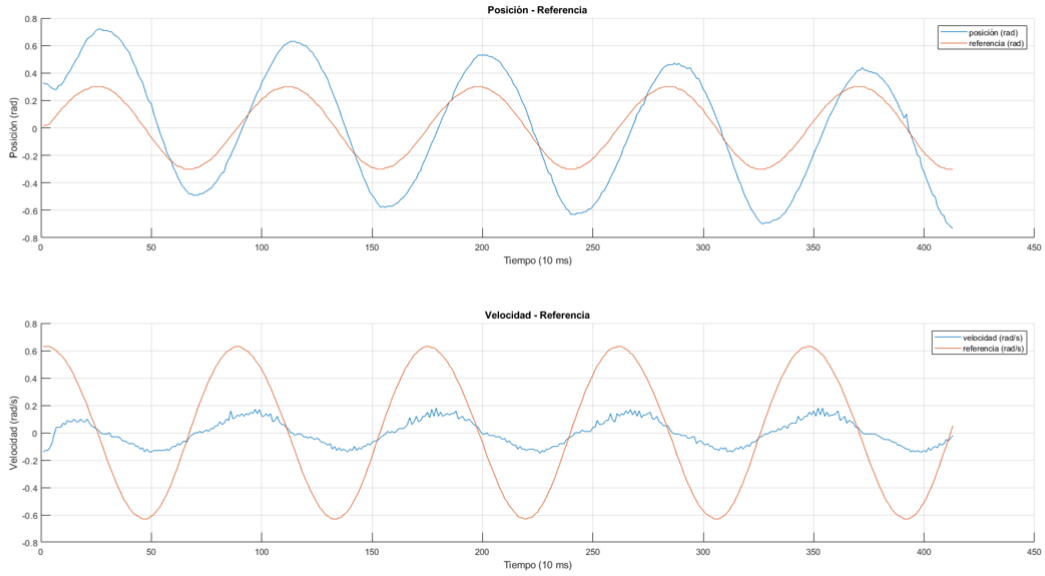

Figura 9: Control Predictivo ante referencia sencilla.

7.2. Control Predictivo: Referencia compuesta

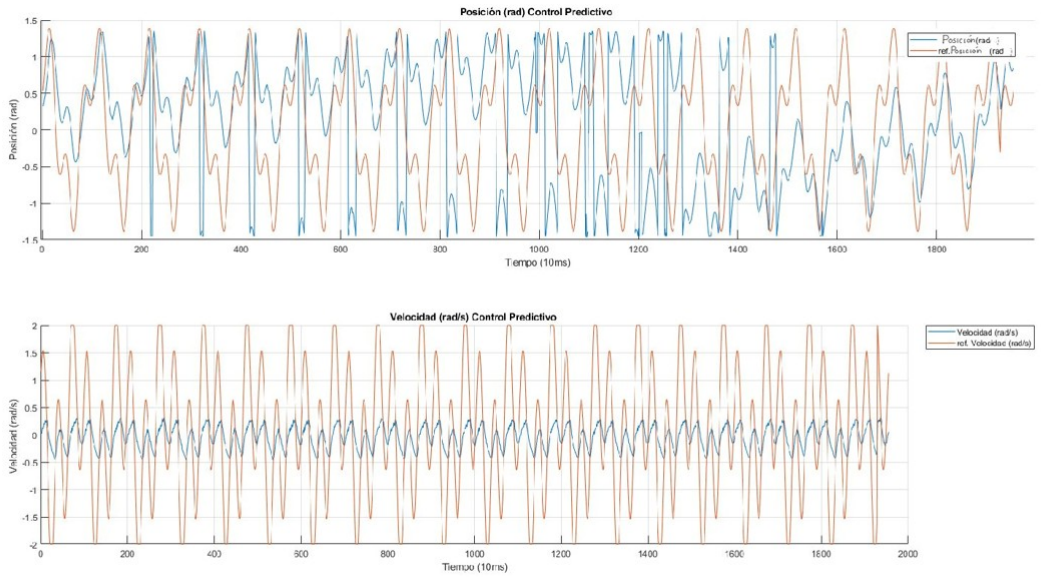

Figura 10: Control Predictivo ante referencia compuesta.

7.3. Control Predictivo: Referencia mixta

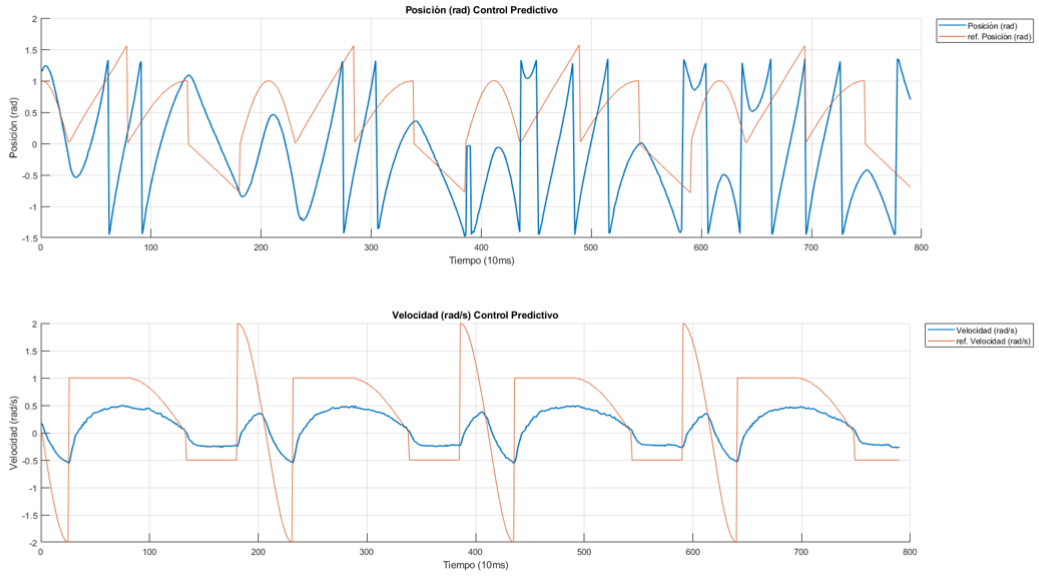

Figura 11: Control Predictivo ante referencia mixta.

7.4. Control Adaptativo: Referencia sencilla

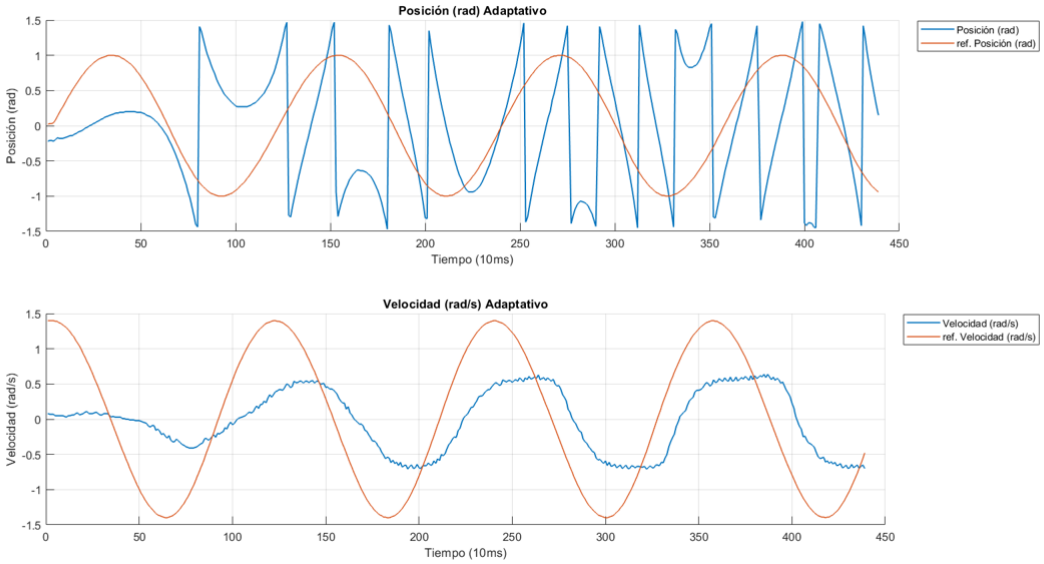

Figura 12: Control Adaptativo ante referencia sencilla.

7.5. Control PID en cascada: Referencia sencilla

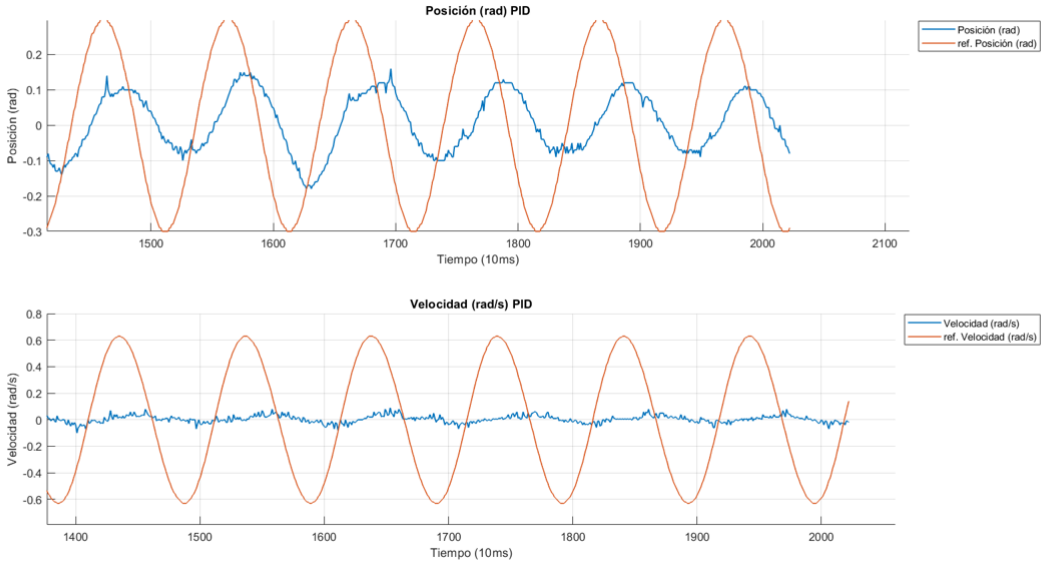

Figura 13: Control PID en cascada ante referencia sencilla.

7.6. Efectos del comportamiento mecánico del motor sobre resultados de los controladores

Si bien se observa un buen comportamiento de los controladores, se pueden notar ciertos picos o tramos similares a dientes de sierra, un ejemplo muy claro es en el control adaptativo:

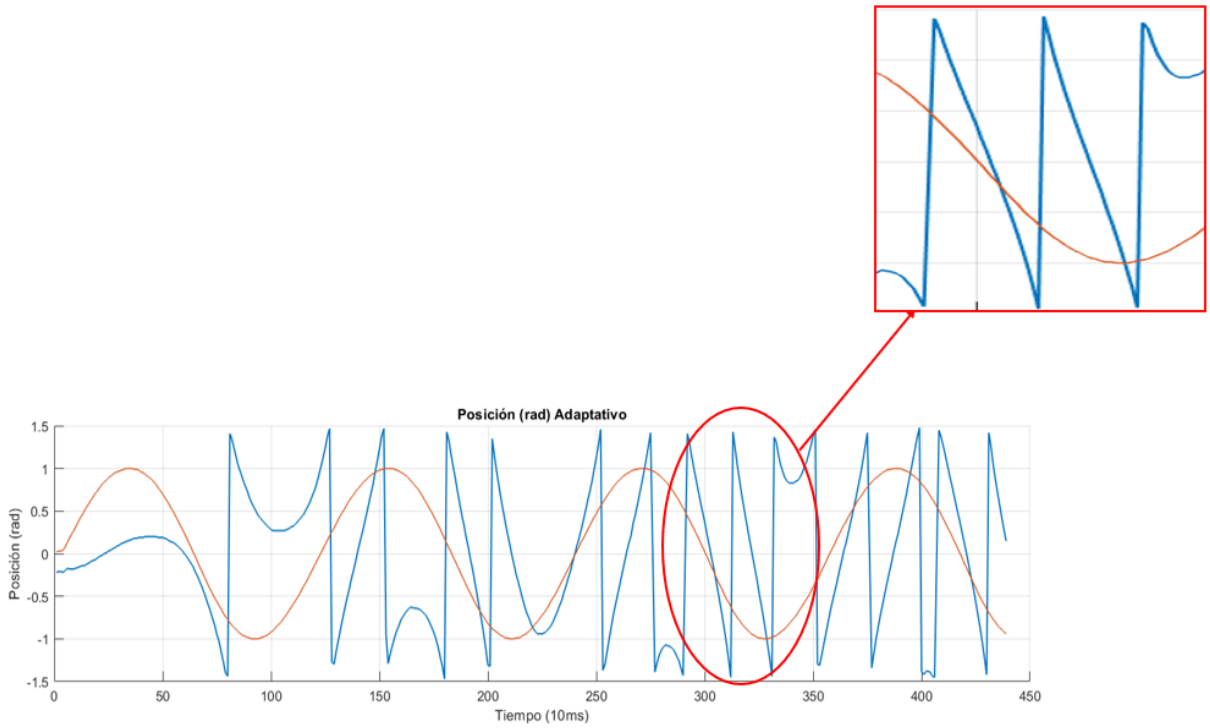

Figura 14: Diente de sierra en control adaptativo.

Esto se debe a **límite mecánico del motor** ya que la posición alcanza la saturación, haciendo que, tras

alcanzar el valor máximo de posición llegue al valor mínimo, dando ese aspecto de diente de sierra. Sin embargo, este comportamiento no se debe ver como un mal comportamiento del controlador, sino como un efecto de la naturaleza del sistema controlado.

8. Conclusiones

8.1. Control Predictivo: Conclusiones

Presenta un excelente desempeño, siempre y cuando no existan cambios bruscos (ya que las perturbaciones no han sido tenidas en cuenta en el modelo matemático empleado). Incluso con referencias complejas, como la mixta, demuestra ser el más efectivo de los tres controladores. Esto se debe a que es especialmente adecuado para sistemas donde la referencia presenta un comportamiento repetitivo, además de permitir un ajuste preciso del comportamiento gracias a la flexibilidad que otorga el poder modificar los pesos de control, así como el poder modificar los horizontes predictivos y de control.

Como inconveniente principal de este controlador, es el que la calidad de las predicciones depende del modelo del sistema empleado, el cual puede presentar problemas. En este caso se ha empleado un modelo sencillo del motor, como el usado en el control LQR que se ha visto en prácticas anteriores, en espacio de estados, obteniendo un buen comportamiento dentro de este marco de estudio.

8.2. Control Adaptativo: Conclusiones

Su desempeño no ha sido tan destacado en esta prueba debido a su alta dependencia de la velocidad de la referencia. Este tipo de controlador funciona mejor con referencias lentas ($\gamma_K = 0{,}005, \gamma_L = 0{,}05$). En nuestro caso, al tratarse de una referencia relativamente rápida, no ha logrado un rendimiento tan favorable en comparación con los otros dos controladores.

Este controlador, además, ha ajustado únicamente dos ganancias, lo cuál no permite un control tan preciso como el visto en el control predictivo, a pesar de tener varios parámetros que pueden ser ajustados para una mayor personalización del comportamiento del controlador. Aún así, se ha podido ver la adaptación de las constantes del control y el efecto que estas tienen sobre la posición y velocidad.

8.3. Control PID: Conclusiones

A pesar de ser un controlador sencillo, ha demostrado un buen comportamiento con la referencia utilizada, especialmente en el control de posición. Sin embargo, presenta limitaciones en el control de velocidad, ya que este depende de la salida del primer lazo de control de posición, restringiendo los valores alcanzables y resultando en un desempeño inferior en el control de esta segunda magnitud.

Las limitaciones presentadas por este controlador se deben a la naturaleza del problema, y el que se empleasen PIDs en cascada para el control de dos magnitudes a la vez, en lugar de emplear un PID por magnitud. Sin embargo, ha servido como una buena línea base frente a la cual comparar los otros dos controladores avanzados que se han empleado.

Anexos

Anexo A: Implementación del Controlador MPC en Arduino

```
#include <Scheduler.h>

// Par metros del sistema discreto
float Ad[2][2] = {{1, 0.0586}, {0, 0.9512}}; // Matriz de transici n
float Bd[2] = {0.0005, 0.0160};              // Matriz de control
float Cd[2][2] = {{1, 0}, {0, 1}};           // Matriz de salida
float Ts = 6;                                // Tiempo de muestreo en milisegundos

// Par metros del MPC
#define Np 10   // Horizonte de predicci n
#define Nu 3    // Horizonte de control
#define wx 100  // Peso posici n
#define wv 20   // Peso velocidad
#define wu 0.01 // Peso control

// Restricciones f sicas del sistema
#define u_min -10
#define u_max 10
#define x_min -3
#define x_max 3
#define v_min -2
#define v_max 2

// Par metros del descenso de gradiente (optimizador usado)
#define max_iters 10
#define alpha 0.1
#define tol 0.0001

// Variables globales
float x[2] = {0, 0};        // Condiciones iniciales
float ref_pos = 0;          // Referencia de posici n
float ref_vel = 0;          // Referencia de velocidad

// Matrices Phi y Gamma
float Phi[Np * 2][2];       // Relaciona estado actual con predicciones futuras
float Gamma[Np * 2][Nu];    // Relaciona control y estado futuros

// Variables de tiempo
unsigned long previous_time = 0;

// Inicializaci n del controlador
void calcularMatrices();

void setup() {
  Serial.begin(9600); // Inicializamos comunicaci n serial
  analogReadResolution(12); // Inicializamos lectura
  analogWriteResolution(12); // Inicializamos escritura
  calcularMatrices(); // Precalculamos Phi y Gamma
}

void loop() {
  unsigned long current_time = millis(); // millis() devuelve el tiempo desde que el
      programa empez  a ejecutarse
  if (current_time - previous_time >= Ts) {
    previous_time = current_time; // Actualizamos el tiempo previo, con esta
        condici n verificamos que el controlador se ejecuta cada Ts
    bloqueControlMPC(); // Ejecutamos el control MPC
  }
}
```

```
58
59  // Calcular Phi y Gamma
60  void calcularMatrices() {
61    for (int i = 0; i < Np; i++) {
62      for (int j = 0; j < 2; j++) {
63        Phi[i * 2 + j][0] = Cd[j][0] * pow(Ad[0][0], i); // pow calcula potencias
64        Phi[i * 2 + j][1] = Cd[j][1] * pow(Ad[1][1], i);
65      }
66      for (int j = 0; j < Nu; j++) {
67        if (i >= j) {
68          Gamma[i * 2][j] = Cd[0][0] * pow(Ad[0][0], i - j) * Bd[0];
69          Gamma[i * 2 + 1][j] = Cd[1][1] * pow(Ad[1][1], i - j) * Bd[1];
70        }
71      }
72    }
73  }
74
75  // Leemos el estado actual del servomotor
76  void leerEstado(float &pos, float &vel) {
77    int sensorValue = analogRead(A0);
78    pos = (sensorValue / 4095.0) * 6.0 - 3.0; // Escalar a rango [-3, 3]
79
80    sensorValue = analogRead(A1);
81    vel = (sensorValue / 4095.0) * 4.0 - 2.0; // Escalar a rango [-2, 2]
82  }
83
84  // Aplicar se al de control al servomotor
85  void aplicarControl(float u) {
86    int salida = (u + 10) * (4095 / 20); // Escalar a rango [0, 4095]
87    salida = constrain(salida, 0, 4095);
88    analogWrite(DAC0, salida);
89  }
90
91  // Generar referencias din micas
92  void generarReferencias() {
93    float T_cycle = 100 * Ts / 1000.0; // Duraci n del ciclo en segundos
94    float f = 1 / T_cycle;             // Frecuencia de la se al
95    float t = millis() / 1000.0;       // Tiempo actual en segundos
96
97    // Generar se ales de referencia din micas
98    ref_pos = 2.5 * sin(2 * PI * f * t);
99    ref_vel = 2 * PI * f * 2.5 * cos(2 * PI * f * t);
100
101   // Restringir valores de referencia al rango permitido
102   ref_pos = constrain(ref_pos, x_min, x_max);
103   ref_vel = constrain(ref_vel, v_min, v_max);
104 }
105
106 // Bucle de control MPC
107 void bloqueControlMPC() {
108   // Leer el estado actual
109   leerEstado(x[0], x[1]);
110
111   // Generar referencias din micas
112   generarReferencias();
113
114   // Crear vector de referencia
115   float x_ref[Np * 2];
116   for (int i = 0; i < Np; i++) {
117     x_ref[i * 2] = ref_pos;
118     x_ref[i * 2 + 1] = ref_vel;
119   }
120
```

```
  // Calcular H y f
  float H[Nu][Nu] = {0};
  float f[Nu] = {0};

  for (int i = 0; i < Nu; i++) {
    for (int j = 0; j < Nu; j++) {
      for (int k = 0; k < Np * 2; k++) {
        H[i][j] += Gamma[k][i] * wx * Gamma[k][j];
      }
    }
    for (int k = 0; k < Np * 2; k++) {
      f[i] += Gamma[k][i] * wx * (Phi[k][0] * x[0] + Phi[k][1] * x[1] - x_ref[k]);
    }
  }

  // Resolver usando descenso de gradiente
  float u_opt[Nu] = {0};
  for (int iter = 0; iter < max_iters; iter++) {
    float grad[Nu] = {0};
    for (int i = 0; i < Nu; i++) {
      grad[i] = f[i];
      for (int j = 0; j < Nu; j++) {
        grad[i] += H[i][j] * u_opt[j];
      }
    }

    // Actualizar controles
    for (int i = 0; i < Nu; i++) {
      u_opt[i] -= alpha * grad[i];
      u_opt[i] = constrain(u_opt[i], u_min, u_max);
    }
  }

  // Aplicar el primer control
  float u = u_opt[0];
  aplicarControl(u);

  // Mostrar datos
  Serial.print("Posici n:␣");
  Serial.print(x[0]);
  Serial.print(",␣Velocidad:␣");
  Serial.print(x[1]);
  Serial.print(",␣Ref␣Pos:␣");
  Serial.print(ref_pos);
  Serial.print(",␣Ref␣Vel:␣");
  Serial.print(ref_vel);
  Serial.print(",␣Control:␣");
  Serial.println(u);
}
```

Listing 1: Código en C

Anexo B: Implementación del Controlador PID en Cascada en Arduino

```
#include <Scheduler.h>

// Par metros del sistema discreto (no tocar bajo riesgo de ejecuci n)
float Ad[2][2] = {{1, 0.0586}, {0, 0.9512}}; // Matriz de transici n
float Bd[2] = {0.0005, 0.0160};              // Matriz de control
float Cd[2][2] = {{1, 0}, {0, 1}};           // Matriz de salida
float Ts = 6;                                // Tiempo de muestreo en milisegundos

```

```
// Restricciones f sicas del sistema
#define u_min -10
#define u_max 10
#define x_min -3
#define x_max 3
#define v_min -2
#define v_max 2

// Par metros de los PID
float Kp_pos = 0.6, Ti_pos = 0.5, Td_pos = 0.125;    // PID de posici n
float Kp_vel = 6.67, Ti_vel = 0.09, Td_vel = 0.0225; // PID de velocidad

// Variables globales
float x[2] = {0, 0};       // Condiciones iniciales
float ref_pos = 0;         // Referencia de posici n
float ref_vel = 0;         // Referencia de velocidad

// Variables del PID de posici n
float error_pos_prev = 0, integral_pos = 0, derivada_pos = 0;

// Variables del PID de velocidad
float error_vel_prev = 0, integral_vel = 0, derivada_vel = 0;

// Variables de tiempo
unsigned long previous_time = 0;

// Funci n de setup
void setup() {
  Serial.begin(9600); // Inicializamos comunicaci n serial
  analogReadResolution(12); // Inicializamos lectura
  analogWriteResolution(12); // Inicializamos escritura
}

// Bucle principal
void loop() {
  unsigned long current_time = millis(); // Tiempo actual en milisegundos
  if (current_time - previous_time >= Ts) {
    previous_time = current_time; // Actualizamos el tiempo previo
    bloqueControlPIDCascada(); // Ejecutamos el control PID en cascada
  }
}

// Leemos el estado actual del servomotor
void leerEstado(float &pos, float &vel) {
  int sensorValue = analogRead(A0);
  pos = (sensorValue / 4095.0) * 6.0 - 3.0; // Escalar a rango [-3, 3]

  sensorValue = analogRead(A1);
  vel = (sensorValue / 4095.0) * 4.0 - 2.0; // Escalar a rango [-2, 2]
}

// Aplicar se al de control al servomotor
void aplicarControl(float u) {
  int salida = (u + 10) * (4095 / 20); // Escalar a rango [0, 4095]
  salida = constrain(salida, 0, 4095);
  analogWrite(DAC0, salida);
}

// Generar referencias din micas
void generarReferencias() {
  float T_cycle = 100 * Ts / 1000.0; // Duraci n del ciclo en segundos
  float f = 1 / T_cycle;             // Frecuencia de la se al
  float t = millis() / 1000.0;       // Tiempo actual en segundos
```

```
72
73    // Generar se ales de referencia din micas
74    ref_pos = 2.5 * sin(2 * PI * f * t);
75    ref_vel = 2 * PI * f * 2.5 * cos(2 * PI * f * t);
76
77    // Restringir valores de referencia al rango permitido
78    ref_pos = constrain(ref_pos, x_min, x_max);
79    ref_vel = constrain(ref_vel, v_min, v_max);
80  }
81
82  // Bloque de control PID en cascada
83  void bloqueControlPIDCascada() {
84    // Leer el estado actual
85    float pos_actual, vel_actual;
86    leerEstado(pos_actual, vel_actual);
87
88    // Generar referencias din micas
89    generarReferencias();
90
91    // Control PID de posici n
92    float error_pos = ref_pos - pos_actual;
93    integral_pos += error_pos * (Ts / 1000.0);
94    derivada_pos = (error_pos - error_pos_prev) / (Ts / 1000.0);
95    error_pos_prev = error_pos;
96
97    float ref_vel_PID = Kp_pos * (error_pos + (1 / Ti_pos) * integral_pos + Td_pos *
          derivada_pos);
98
99    // Restringir la referencia de velocidad
100   ref_vel_PID = constrain(ref_vel_PID, v_min, v_max);
101
102   // Control PID de velocidad
103   float error_vel = ref_vel_PID - vel_actual;
104   integral_vel += error_vel * (Ts / 1000.0);
105   derivada_vel = (error_vel - error_vel_prev) / (Ts / 1000.0);
106   error_vel_prev = error_vel;
107
108   float u = Kp_vel * (error_vel + (1 / Ti_vel) * integral_vel + Td_vel *
          derivada_vel);
109
110   // Restringir la se al de control
111   u = constrain(u, u_min, u_max);
112
113   // Aplicar control
114   aplicarControl(u);
115
116   // Mostrar datos
117   Serial.print("Posici n:␣");
118   Serial.print(pos_actual);
119   Serial.print(",␣Velocidad:␣");
120   Serial.print(vel_actual);
121   Serial.print(",␣Ref␣Pos:␣");
122   Serial.print(ref_pos);
123   Serial.print(",␣Ref␣Vel:␣");
124   Serial.print(ref_vel);
125   Serial.print(",␣Control:␣");
126   Serial.println(u);
127 }
```

Listing 2: Código en C

Anexo C: Implementación del Controlador Adaptativo en Arduino

```
#include <Scheduler.h>

// Par metros del sistema discreto
float Ad[2][2] = {{1, 0.0586}, {0, 0.9512}};
float Bd[2] = {0.0005, 0.0160};
float Cd[2][2] = {{1, 0}, {0, 1}};
float Ts = 6; // Tiempo de muestreo en milisegundos

// Par metros del controlador adaptativo
float gamma_k = 0.005;
float gamma_l = 0.05;
float lambda = 0.001;
float K[2] = {0.01, 0.01}; // Ganancias iniciales
float L[2] = {0.01, 0.01}; // Ganancias iniciales

// Restricciones f sicas del sistema
#define u_min -10
#define u_max 10
#define x_min -3
#define x_max 3
#define v_min -2
#define v_max 2

// Variables globales
float x[2] = {0, 0}; // Condiciones iniciales
float ref_pos = 0;    // Referencia de posici n
float ref_vel = 0;    // Referencia de velocidad
unsigned long previous_time = 0;

// Inicializaci n del controlador
void setup() {
  Serial.begin(9600);
  analogReadResolution(12);
  analogWriteResolution(12);
}

void loop() {
  unsigned long current_time = millis();
  if (current_time - previous_time >= Ts) {
    previous_time = current_time;
    bloqueControlAdaptativo();
  }
}

// Leer el estado actual del servomotor
void leerEstado(float &pos, float &vel) {
  int sensorValue = analogRead(A0);
  pos = (sensorValue / 4095.0) * 6.0 - 3.0; // Escalar a rango [-3, 3]

  sensorValue = analogRead(A1);
  vel = (sensorValue / 4095.0) * 4.0 - 2.0; // Escalar a rango [-2, 2]
}

// Aplicar se al de control al servomotor
void aplicarControl(float u) {
  int salida = (u + 10) * (4095 / 20); // Escalar a rango [0, 4095]
  salida = constrain(salida, 0, 4095);
  analogWrite(DAC0, salida);
}

// Generar referencias din micas
void generarReferencias() {
  float T_cycle = 100 * Ts / 1000.0; // Duraci n del ciclo en segundos
```

```
  float f = 1 / T_cycle;              // Frecuencia de la se al
  float t = millis() / 1000.0;        // Tiempo actual en segundos

  // Generar se ales de referencia din micas
  ref_pos = 2.5 * sin(2 * PI * f * t);
  ref_vel = 2 * PI * f * 2.5 * cos(2 * PI * f * t);

  // Restringir valores de referencia al rango permitido
  ref_pos = constrain(ref_pos, x_min, x_max);
  ref_vel = constrain(ref_vel, v_min, v_max);
}

// Bucle de control adaptativo
void bloqueControlAdaptativo() {
  // Leer el estado actual
  leerEstado(x[0], x[1]);

  // Generar referencias din micas
  generarReferencias();

  // Crear vector de referencia
  float r[2] = {ref_pos, ref_vel};

  // Calcular error
  float e[2] = {r[0] - (Cd[0][0] * x[0] + Cd[0][1] * x[1]),
                r[1] - (Cd[1][0] * x[0] + Cd[1][1] * x[1])};

  // Calcular se al de control
  float u = -(K[0] * x[0] + K[1] * x[1]) + (L[0] * r[0] + L[1] * r[1]);
  u = constrain(u, u_min, u_max);

  // Actualizar estado del sistema
  float x_new[2];
  x_new[0] = Ad[0][0] * x[0] + Ad[0][1] * x[1] + Bd[0] * u;
  x_new[1] = Ad[1][0] * x[0] + Ad[1][1] * x[1] + Bd[1] * u;
  x_new[0] = constrain(x_new[0], x_min, x_max);
  x_new[1] = constrain(x_new[1], v_min, v_max);
  x[0] = x_new[0];
  x[1] = x_new[1];

  // Actualizar ganancias adaptativas
  for (int i = 0; i < 2; i++) {
    K[i] += gamma_k * (e[i] * x[i]) - lambda * K[i];
    L[i] += gamma_l * (e[i] * r[i]) - lambda * L[i];
    K[i] = constrain(K[i], -10, 10);
    L[i] = constrain(L[i], -10, 10);
  }

  // Aplicar el control
  aplicarControl(u);

  // Mostrar datos
  Serial.print("Posici n:␣");
  Serial.print(x[0]);
  Serial.print(",␣Velocidad:␣");
  Serial.print(x[1]);
  Serial.print(",␣Ref␣Pos:␣");
  Serial.print(ref_pos);
  Serial.print(",␣Ref␣Vel:␣");
  Serial.print(ref_vel);
  Serial.print(",␣Control:␣");
  Serial.println(u);
}
```

Listing 3: Código en C

Anexo D: Código en MATLAB (Estabilidad, Controlabilidad y Observabilidad)

```
%% Sistema en espacio de estados (lo sacamos en la p5)

% Constantes sistema
K1 = 0.328;
K2 = 9.24;
Tau = 0.13;
Ts = Tau/20;
A = [0, K2; 0, -1/Tau];
B = [0; K1/Tau];
C = eye(2);
D = 0;

%%
%Necesitamos demostrar que el controlador es observable y controlable en
%toda la continuidad de n; me puse a hacerlo a mano pero resulta que matlab
%tiene funciones que te lo sacan jajajaja historia

% Estabilidad
autovalores = eig(A);
if all(real(autovalores) < 0)
    disp('El sistema continuo es estable.');
else
    disp('El sistema continuo NO es estable.');
end

% Controlabilidad
CM = ctrb(A, B);
if rank(CM) == size(A, 1)
    disp('El sistema es controlable.');
else
    disp('El sistema NO es controlable.');
end

% Observabilidad
OM = obsv(A, C);
if rank(OM) == size(A, 1)
    disp('El sistema es observable.');
else
    disp('El sistema NO es observable.');
end

%TRIUNFO! Esta demostrada la necesidad de un controlador dadas las
%propiedades del sistema (inestable, observable y controlable)

%%
% Discretizaci n igual que en la p5
Ts = Tau/20; % Tiempo de muestreo
sysc = ss(A, B, C, D); % Crear sistema continuo
sysd = c2d(sysc, Ts); % Discretizar con zero-order hold
[Ad, Bd, Cd, Dd] = ssdata(sysd); % Obtener matrices discretizadas
```

Listing 4: Código en MATLAB

Anexo E: Código en MATLAB (Código general para simular)

```
%Matrices discretas del sistema
Ad=[1 0.0586;0 0.9512];
Bd=[0.0005; 0.0160];
Cd=[1 0; 0 1];
Ts=0.0065;

%% Par metros del MPC
Np = 5; % Horizonte predicci n
Nu = 5;  % Horizonte control
wx = 100; % P.error posici n
wv = 100;  % P. error velocidad
wu = 0.01; % P. control

u_min = -10; % inf control
u_max = 10;  % inf control

% Debemos asegurarnos de que se evita la saturaci n
x_min = -3; % min pos
x_max = 3;  % max pos
v_min = -2; % min vel
v_max = 2;  % max vel

%Matrices para la penalizaci n (L gica del LQR)
Q = blkdiag(kron(eye(Np), wx), kron(eye(Np), wv)); % Penalizaci n de posici n y
    velocidad
%Q se divide en dos matrices, ambas de dimensi n Np, una tiene una diagonal
%con los valores de wx y otra una diagonal con los valores de wv;
%finalmente se unen en una matriz cuadrada de dimension 2NPx2NP cuya
%diagonal contiene los valores de wx y wv
R = wu * eye(Nu); % Penalizaci n de control

%% Construcci n de Phi y Gamma (para posici n y velocidad)
Phi = []; %Esta matriz va a ser la encargada de que se relacionen los estados
    futuros del sistema con el estado en el que se encuentra
Gamma = zeros(Np * 2, Nu); % Inicializar Gamma para posici n y velocidad, va a "
    asignar" a los estados futuros las se ales de control que le corresponder an
% Se construyen teniendo en cuenta que son matrices dependientes del
% espacio de estados discretizado
for i = 1:Np
    Phi = [Phi; Cd * (Ad^i)];
    for j = 1:min(i, Nu)
        row_start = (i-1)*2 + 1;
        row_end = i*2;
        col_start = (j-1) + 1;
        col_end = j;
        Gamma(row_start:row_end, col_start:col_end) = Cd * (Ad^(i-j)) * Bd;
    end
end

%% Generar referencias peri dicas suavizadas (senoidal) con limitaciones
T_cycle = 100 * Ts;     % Duraci n de un ciclo en segundos (100 iteraciones con Ts)
f = 1 / T_cycle;        % Frecuencia de la se al (1 ciclo cada 100 iteraciones)
A_pos = 0.35;            % Amplitud de la posici n ajustada para respetar l mites
    (\pm 3)
A_vel = 2;              % Amplitud m xima permitida para la velocidad (\pm 2)
steps = 1000;           % Total de iteraciones

% Vector de tiempo para un ciclo
t_cycle = 0:Ts:T_cycle-Ts;

% Generar referencia de posici n para un ciclo
ref_cycle_pos = 0.3 * sin(2 * pi * f * t_cycle);

```

```
% Derivar la posici n para obtener la velocidad (coherente)
ref_cycle_vel = 2 * pi * f * 0.3 * cos(2 * pi * f* t_cycle);

% Escalar las referencias para cumplir con las limitaciones
ref_cycle_pos = min(max(ref_cycle_pos, -3), 3); % Limitar posici n entre \pm3
ref_cycle_vel = min(max(ref_cycle_vel, -2), 2); % Limitar velocidad entre \pm2

% Repetir el ciclo peri dicamente
cycles = ceil(steps / length(t_cycle));
ref_store_pos = repmat(ref_cycle_pos, 1, cycles);
ref_store_vel = repmat(ref_cycle_vel, 1, cycles);

% Ajustar las referencias al n mero total de iteraciones
ref_store_pos = ref_store_pos(1:steps);
ref_store_vel = ref_store_vel(1:steps);

%% Implementaci n de MPC
%Se ha acudido al descenso de gradiente
% Par metros del descenso de gradiente
max_iters = 10; % N mero m ximo de iteraciones
alpha = 0.05;   % Tasa de aprendizaje
tol = 0.0001;   % Tolerancia para convergencia

% Restricciones de control (se pasan a formato matricial)
u_lb = u_min * ones(Nu, 1); % L mite inf del control
u_ub = u_max * ones(Nu, 1); % L mite sup del control

x0 = [0; 0]; % Estado inicial
x = x0; % Estado inicial para la simulaci n

u_store = []; % esto funciona igual que el u_hist de la p5
x_store = x'; % esto funciona igual que el x_hist de la p5, la ' es innegociable

for k = 1:steps
    % Usar las referencias generadas
    ref_pos = ref_store_pos(k);
    ref_vel = ref_store_vel(k);

    % Generar el vector de referencia para el horizonte
    x_ref = repmat([ref_pos; ref_vel], Np, 1);

    % ESTE ES EL PROBLEMA DE OPTIMIZACION
    f = Gamma' * Q * (Phi * x - x_ref);
    H = Gamma' * Q * Gamma + R;
    H = 0.5 * (H + H'); % Asegurar simetr a del Hessiano, sino puede romperse en
        cualquier momento

    % Inicializaci n del vector de control
    u_opt = zeros(Nu, 1); % Inicializar en cero
    grad_prev = Inf; % Gradiente previo para verificar convergencia

    % Bucle para el descenso de gradiente
    for iter = 1:max_iters
        % Gradiente de la funci n de costo
        grad = H * u_opt + f;

        % Actualizar el vector de control
        u_opt = u_opt - alpha * grad;

        % Aplicar restricciones de control (proyecci n)
        u_opt = max(u_lb, min(u_ub, u_opt));

        % Verificar convergencia
```

```
        if norm(grad - grad_prev, 2) < tol
            break;
        end
        grad_prev = grad;
    end

    % Aplicar el primer control calculado
    u = u_opt(1);

    % Actualizar el estado
    x = Ad * x + Bd * u;

    % Guardar resultados
    u_store = [u_store; u];
    x_store = [x_store; x'];
end

%% Graficar resultados
figure;
subplot(3,1,1);
plot(x_store(:,1));
hold on;
plot(ref_store_pos, '--');
title('Posici n vs Referencia');
xlabel('Tiempo (pasos)');
ylabel('Posici n');
legend('Posici n', 'Referencia');

subplot(3,1,2);
plot(x_store(:,2));
hold on;
plot(ref_store_vel, '--');
title('Velocidad vs Referencia');
xlabel('Tiempo (pasos)');
ylabel('Velocidad');
legend('Velocidad', 'Referencia');

subplot(3,1,3);
plot(u_store);
title('Control Aplicado');
xlabel('Tiempo (pasos)');
ylabel('Control');
legend('u');
 %%
% % Inicializar par metros del sistema
% Ad = [1 0.0586; 0 0.9512];
% Bd = [0.0005; 0.0160];
% Cd = [1 0; 0 1];
% Ts = 0.0065;
%
% % Grid Search: definir valores de par metros
% Nu_values = [3, 5, 7];
% Np_values = [5, 10, 15];
% wu_values = [0.01,0.05,0.1,0.5,1];
% wx_values = [1,5,10,25,50,100];
% wv_values = [1,5,10,25,50,100];
% alpha_values = [0.01,0.05,0.1,0.5];
%
% % Referencias
% T_cycle = 100 * Ts;
% f = 1 / T_cycle;
% steps = 200;
%
```

```
% t_cycle = (0:steps-1) * Ts;
% ref_pos = 2.5 * sin(2 * pi * f * t_cycle);
% ref_vel = 2 * pi * f * 2.5 * cos(2 * pi * f * t_cycle);
%
% % Limitar las referencias
% ref_pos = min(max(ref_pos, -3), 3);
% ref_vel = min(max(ref_vel, -2), 2);
%
% % Variables para almacenar resultados
% results = [];
%
% % Grid Search
% for Nu = Nu_values
%     for Np = Np_values
%         for wu = wu_values
%             for wx = wx_values
%                 for wv = wv_values
%                     for alpha = alpha_values
%                         % Crear matrices Q y R
%                         Q = blkdiag(kron(eye(Np), wx), kron(eye(Np), wv));
%                         R = wu * eye(Nu);
%
%                         % Construcci n de Phi y Gamma
%                         Phi = [];
%                         Gamma = zeros(Np * 2, Nu);
%                         for i = 1:Np
%                             Phi = [Phi; Cd * (Ad^i)];
%                             for j = 1:min(i, Nu)
%                                 Gamma((i-1)*2+1:i*2, j) = Cd * (Ad^(i-j)) * Bd;
%                             end
%                         end
%
%                         % Simulaci n
%                         x = [0; 0]; % Estado inicial
%                         u_store = [];
%                         mse_pos = 0; mse_vel = 0; exceeded = false;
%
%                         for k = 1:steps
%                             % Referencia actual
%                             x_ref = repmat([ref_pos(k); ref_vel(k)], Np, 1);
%
%                             % Construcci n de H y f
%                             f = Gamma' * Q * (Phi * x - x_ref);
%                             H = Gamma' * Q * Gamma + R;
%                             H = 0.5 * (H + H'); % Asegurar simetr a
%
%                             % Descenso de gradiente
%                             u_opt = zeros(Nu, 1);
%                             for iter = 1:10
%                                 grad = H * u_opt + f;
%                                 u_opt = u_opt - alpha * grad;
%
%                                 % Restringir el control
%                                 u_opt = min(max(u_opt, -10), 10);
%                             end
%
%                             % Aplicar el primer control
%                             u = u_opt(1);
%
%                             % Verificar si excede l mites
%                             if abs(u) > 9
%                                 exceeded = true;
%                                 break;
```

```
%                                 end
%
%                                 % Actualizar el estado
%                                 x = Ad * x + Bd * u;
%
%                                 % Acumular errores
%                                 mse_pos = mse_pos + (x(1) - ref_pos(k))^2;
%                                 mse_vel = mse_vel + (x(2) - ref_vel(k))^2;
%
%                                 % Almacenar control
%                                 u_store = [u_store; u];
%                             end
%
%                             % Si no excedi  l mites , guardar resultado
%                             if ~exceeded
%                                 mse_pos = mse_pos / steps;
%                                 mse_vel = mse_vel / steps;
%                                 results = [results; Nu, Np, wu, wx, wv, alpha, mse_pos
    + mse_vel];
%                             end
%                         end
%                     end
%                 end
%             end
%     end
% end
%
% % Ordenar resultados por error (de menor a mayor)
% results = sortrows(results, 7);
%
% % Mostrar las 100 mejores combinaciones
% best_results = results(1:min(100, size(results, 1)), :);
%
% % Exportar resultados
% disp('Las 100 mejores configuraciones:');
% disp('Nu   Np   wu     wx     wv     alpha   Error');
% disp(best_results);
%
% % Guardar resultados en un archivo
% writematrix(best_results, 'mpc_best_results2.csv');
%%
% %% Par metros del controlador adaptativo
% gamma_k = 0.005; % Tasas de aprendizaje reducidas
% gamma_l = 0.05;
% lambda = 0.001; % Regularizaci n para evitar crecimiento excesivo de ganancias
% K = [0.01, 0.01]; % Ganancias iniciales m s bajas
% L = [0.01, 0.01];
%
%
% %% Implementaci n del controlador adaptativo
% x0 = [0; 0];
% x = x0;
%
% u_store = [];
% x_store = x';
% K_store = K;
% L_store = L;
%
% for k = 1:steps
%     ref_pos = ref_store_pos(k);
%     ref_vel = ref_store_vel(k);
%     r = [ref_pos; ref_vel];
%
```

```
%     e = r - Cd * x;
%
%     u = -K * x + L * r;
%     if numel(u) > 1
%         u = u(1);
%     end
%
%     % Saturar el control
%     u = min(max(u, -5), 5);
%
%     % Actualizar el estado del sistema
%     x = Ad * x + Bd * u;
%
%     % Saturar el estado
%     x(1) = min(max(x(1), -3), 3);
%     x(2) = min(max(x(2), -2), 2);
%
%     % Actualizar las ganancias adaptativas con regularizaci n
%     K = K + gamma_k * (e * x') - lambda * K;
%     L = L + gamma_l * (e * r') - lambda * L;
%
%     % Limitar las ganancias adaptativas
%     K = min(max(K, -10), 10);
%     L = min(max(L, -10), 10);
%
%     % Guardar resultados
%     u_store = [u_store; u];
%     x_store = [x_store; x'];
%     K_store = [K_store; K];
%     L_store = [L_store; L];
% end
```

Listing 5: Código en MATLAB

Diagrama de flujo

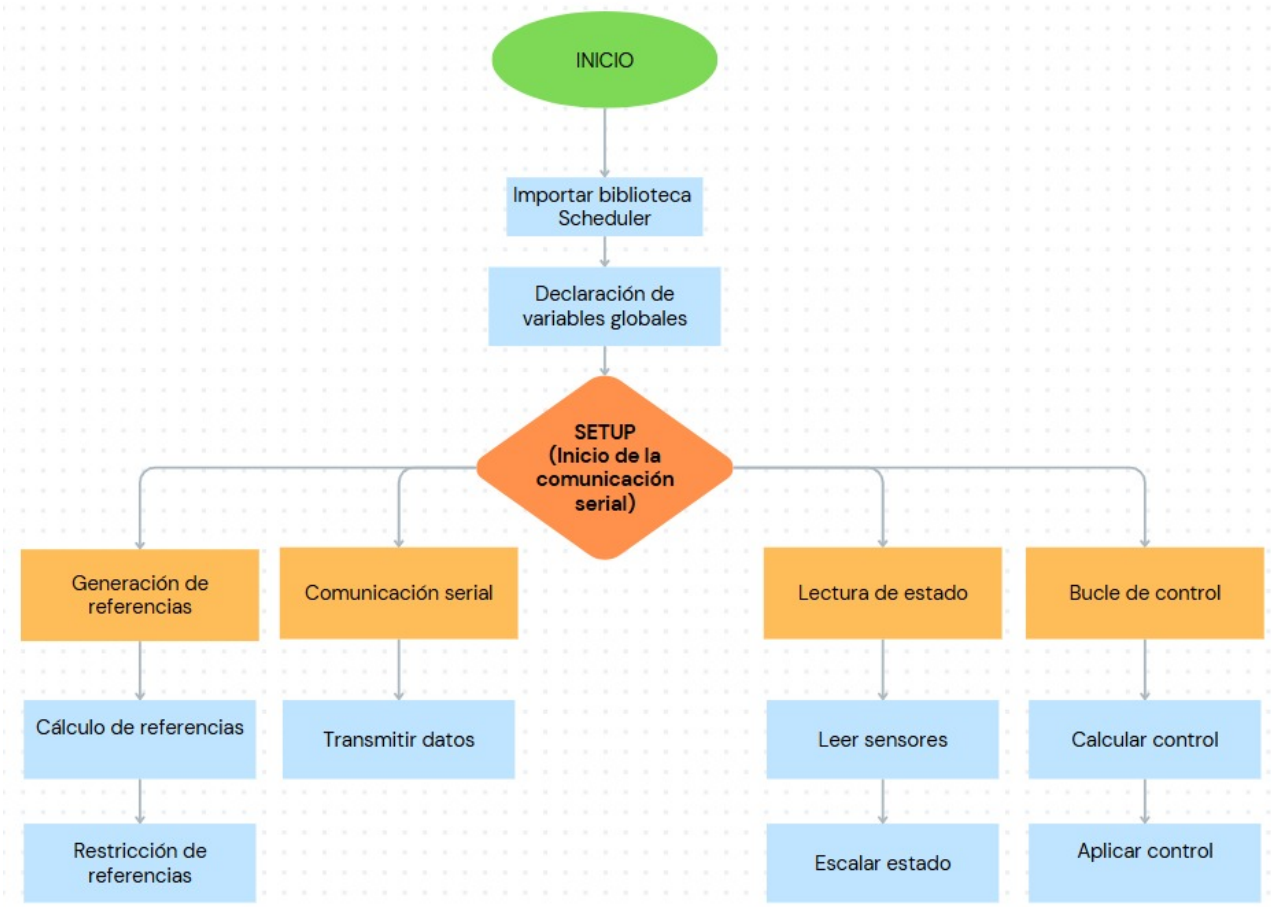

Figura 15: Diagrama de flujo.