

“Año del Bicentenario, de la consolidación de nuestra Independencia, y de la conmemoración de las heroicas batallas de Junín y Ayacucho”

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
(Universidad del Perú, DECANA DE AMÉRICA)
FACULTAD DE SISTEMAS E INFORMÁTICA
ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE



“Informe Final de Proyecto - Control Moderno del Péndulo Invertido”

Trabajo como parte del proyecto final del curso de Automatización y Control de Software

ESTUDIANTES:

- Castillo Carranza, Jose Richard
- Espinoza Fabian, Josue Marcelo
- Hinostroza Quispe, Gianlucas Amed
- Ipanaque Pazo, Jorge Paul
- Peña Manuyama, Dafna Nicole

DOCENTE:

Rosas Cuevas, Yessica

LIMA-PERÚ

2024

Índice

1. Introducción	3
2. Marco teórico	3
3. Modelado del Sistema	6
Ecuaciones del modelo	6
Funciones de transferencia	10
Espacio de estados	12
Código	17
4. Diseño e Implementación del Controlador PID	21
Implementación de la Clase PendulumSystem	21
Implementación de la Clase PendulumApp	23
Preguntas:	25
a. ¿Cómo influyen los parámetros individuales K_p , K_i y K_d en la respuesta del sistema?	25
b. ¿Qué diferencias se observan en el comportamiento del sistema al usar controladores P, PI, PD y PID?	26
5. Optimización del PID con Algoritmos Genéticos	27
6. Optimización con Filtro de Kalman	32
Implementación y Configuración del Filtro de Kalman	32
Cálculo de Predicción y Corrección	33
7. Análisis de Resultados	34
7.1. Evaluación del desempeño	34
Sin control inicial	34
Comparación entre los controladores P, PI, PD y PID	36
7.2. Preguntas	44
a. ¿Qué tan sensible es el sistema con el controlador PID (manual u optimizado) ante variaciones en las condiciones iniciales, como un ángulo mayor del péndulo o una posición inicial desplazada del carro?	44
b. Después de la optimización, ¿qué cambios observas en la curva de error del sistema?	45
c. ¿Qué diferencias importantes se identifican en el comportamiento del sistema con el controlador optimizado respecto a los ajustes manuales?	46
Conclusiones	47
Referencias	48

Control Moderno del Péndulo Invertido

1. Introducción

El péndulo invertido es un problema clásico en la ingeniería de control que involucra mantener una varilla en posición vertical sobre un carro móvil mediante la aplicación de fuerzas horizontales. Este sistema es intrínsecamente inestable, lo que lo convierte en un desafío ideal para el diseño, análisis y evaluación de estrategias de control. Su estudio tiene aplicaciones en robótica, estabilización de vehículos y sistemas de balance, además de servir como un modelo simplificado para analizar problemas complejos en el ámbito de la automatización y el control.

El objetivo de este proyecto es diseñar e implementar un sistema de control moderno para estabilizar el péndulo invertido, abordando tanto técnicas tradicionales como alternativas. En la primera etapa, se deriva la función de transferencia del sistema y se analiza su estabilidad inicial. Posteriormente, se diseña un controlador PID y sus variantes (P, PI, PD), evaluando el impacto de los parámetros K_p , K_i y K_d en el comportamiento del sistema mediante simulaciones. Para mejorar el desempeño, se emplean algoritmos genéticos para optimizar los parámetros del PID. Finalmente, se analiza la sensibilidad del sistema propuesto a diferentes condiciones iniciales, mediante la comparación de gráficos y simulaciones.

2. Marco teórico

El péndulo invertido es un sistema dinámico que se caracteriza por su inestabilidad y la dificultad para mantenerlo en equilibrio, por lo que es un ejemplo clásico en el estudio del control. Kim et al. (2023) desarrollaron un método de planificación de trayectorias basado en un método llamado inversión dinámica, el cual ayuda a que el péndulo se mantenga en equilibrio y se adapte a los cambios del entorno. Este método consiste en intercambiar las entradas y las salidas del sistema para calcular cómo ajustar los controles necesarios para

mantener el péndulo en equilibrio, por lo que permite responder rápidamente a las perturbaciones.

El trabajo de Ali y Mandal (2022) presenta un controlador basado en el Regulador Cuadrático Lineal (LQR) combinado con un filtro de Kalman para estabilizar y rastrear el comportamiento del sistema de péndulo invertido. Este enfoque permite al controlador beneficiarse de estimaciones precisas de las variables de estado, lo que incrementa su efectividad. En este contexto, el filtro de Kalman desempeña un papel esencial al minimizar los efectos del ruido generado por los procesos y los sensores en las salidas del sistema. Además, los autores realizan una comparación del desempeño del sistema en lazo cerrado, con y sin la implementación del filtro de Kalman. Los resultados obtenidos evidencian que su inclusión mejora significativamente la respuesta del sistema y asegura el cumplimiento de las especificaciones de rendimiento establecidas.

Por otro lado, Nisar et al. (2025) destacan la importancia de la implementación de un controlador PID en sistemas que requieren regulaciones precisas y eficientes. En su estudio, aplicaron este enfoque dentro de un modelo de monitoreo de niveles de insulina y glucosa, con el propósito de controlar las fluctuaciones observadas en estas variables. El uso del controlador PID demostró ser crucial en escenarios donde los datos presentan variaciones constantes, garantizando una alta precisión y minimizando el margen de error asociado con las herramientas de medición. En este sentido, los controladores PID se posicionan como una solución confiable para optimizar el análisis y manejo de magnitudes variables en entornos dinámicos.

Teniendo en cuenta también Askari et al. (2010), desarrollaron un modelo de control predictivo (MPC) para un sistema de péndulo invertido, demostrando la aplicabilidad de la metodología de control a un sistema inestable. El modelo se aplicó a un módulo de péndulo rotatorio, utilizando un enfoque de programación cuadrática para resolver el problema de

optimización en tiempo real. La investigación se centró en estudiar el efecto de las perturbaciones de entrada y demostró la capacidad del controlador para mantener el péndulo en posición vertical, rechazar perturbaciones manuales y operar dentro de restricciones físicas del sistema, como los límites de corriente del motor y el ángulo del brazo.

3. Modelado del Sistema

En esta sección se representa matemáticamente el comportamiento del péndulo invertido a través de las ecuaciones físicas que describen los movimientos. Estas ecuaciones serán transformadas al dominio de Laplace para analizar de mejor manera la estabilidad del sistema del péndulo invertido.

Ecuaciones del modelo

En esta sección se derivan las ecuaciones dinámicas que describen el comportamiento del sistema de péndulo invertido. Posteriormente, estas ecuaciones se transforman mediante la transformada de Laplace para obtener la función de transferencia, que relaciona la entrada del sistema (fuerza aplicada al carro) con sus salidas (ángulo del péndulo y posición del carro). En este caso, la entrada del sistema sería la fuerza aplicada al carro y las salidas del sistema serían la posición del carro y el ángulo del péndulo partiendo desde la posición vertical. Por lo tanto, se tienen las siguientes ecuaciones:

$$Ml\ddot{\theta} = (M + m)g\theta - u \quad \dots (1)$$

$$M\ddot{x} = u - mg\theta \quad \dots (2)$$

Donde:

M : Masa del carro

m : Masa del péndulo

l : Longitud de la varilla del péndulo

θ : Ángulo de inclinación del péndulo con respecto a la posición vertical

$\ddot{\theta}$: Aceleración angular del péndulo (Segunda derivada del ángulo θ con respecto al tiempo)

x : Posición del carro

\ddot{x} : Aceleración lineal del carro (Segunda derivada de la posición x con respecto al tiempo)

De las ecuaciones mostradas, la ecuación (1) representa la dinámica angular del péndulo, es decir, describe cómo la fuerza u afecta al movimiento angular del péndulo. La ecuación (2) describe la dinámica lineal del carro, es decir, describe cómo la fuerza u afecta a la posición lineal del carro.

Ahora, se aplicará la transformada de Laplace para hallar las funciones de transferencia.

Transformada de Laplace de la ecuación (1):

$$Ml\ddot{\theta} = (M + m)g\theta - u$$

Para pasar esta ecuación al dominio de Laplace, se aplicará la transformada de Laplace a cada término, entonces:

$$L\{Ml\ddot{\theta}\} = L\{(M + m)g\theta\} - L\{u\}$$

Transformada de $Ml\ddot{\theta}$:

En este caso, Ml se puede tomar como una constante, por lo que multiplica a $\ddot{\theta}$, es decir, multiplica a la segunda derivada de θ con respecto al tiempo. Para aplicar una transformada de Laplace de una derivada de orden n se tiene lo siguiente:

$$L\left\{\frac{d^n f(t)}{dt^n}\right\} = s^n F(s) - s^{n-1}f(0) - \dots - \frac{d^{n-1}f(0)}{dt^{n-1}}$$

Entonces, para la segunda derivada, $n = 2$, por lo que nos queda lo siguiente:

$$L\{\ddot{\theta}(t)\} = s^2\theta(s) - s\dot{\theta}(0) - \theta(0)$$

De lo anterior, se tiene que $\dot{\theta}(0) = 0$ y $\theta(0) = 0$, ya que se asumen condiciones iniciales nulas, es decir, que en $t = 0$, el ángulo inicial del péndulo es de 0 radianes y la velocidad angular inicial también es de 0 rad/s, ya que no tiene movimiento inicial. Por lo tanto, se tiene lo siguiente:

$$L\{\ddot{\theta}(t)\} = s^2\theta(s)$$

Entonces, como Ml es una constante, se conserva cuando se aplica la transformada, por lo que esta queda de esta manera:

$$L\{Ml\ddot{\theta}\} = Ml \cdot L\{\ddot{\theta}(t)\}$$

$$L\{Ml\ddot{\theta}\} = Ml \cdot s^2\theta(s) \quad \dots (3)$$

Transformada de $(M + m)g\theta$:

$(M + m)g$ es una constante que multiplica a θ , que sería la función dependiente del tiempo, es decir, $\theta(t)$. Para el caso de las transformadas de Laplace para una función con respecto al tiempo, se tiene algo más sencillo:

$$L\{f(t)\} = F(s)$$

Por lo tanto, para $\theta(t)$, se tiene lo siguiente:

$$L\{\theta(t)\} = \theta(s)$$

Entonces:

$$L\{(M + m)g\theta(t)\} = (M + m)g \cdot L\{\theta(t)\}$$

$$L\{(M + m)g\theta(t)\} = (M + m)g \cdot \theta(s) \quad \dots (4)$$

Transformada de u :

Este término es dependiente del tiempo, ya que es la entrada del sistema en el dominio del tiempo, por lo tanto:

$$L\{u(t)\} = U(s) \quad \dots (5)$$

Finalmente, se tiene la transformada para la ecuación (1):

$$L\{Ml\ddot{\theta}\} = L\{(M + m)g\theta\} - L\{u\}$$

Reemplazando (3), (4) y (5):

$$Mls^2\theta(s) = (M + m)g\theta(s) - U(s)$$

Transformada de Laplace de la ecuación (2):

$$M\ddot{x} = u - mg\theta$$

Se realiza un procedimiento similar al anterior, por lo tanto:

$$L\{M\ddot{x}\} = L\{u\} - L\{mg\theta\}$$

Transformada de $M\ddot{x}$:

Para pasar esta ecuación al dominio de Laplace, se realiza algo similar a lo que se hizo con el término $Ml\ddot{\theta}$ de la ecuación (1), ya que se tiene una segunda derivada de x , por lo tanto:

$$L\{\ddot{x}(t)\} = s^2 X(s) - s\dot{x}(0) - x(0)$$

Nuevamente se asumen condiciones iniciales nulas, entonces queda de la siguiente manera:

$$L\{\ddot{x}(t)\} = s^2 X(s)$$

La transformada para el término $M\ddot{x}$ queda así:

$$L\{M\ddot{x}\} = M \cdot L\{\ddot{x}(t)\} = M \cdot s^2 X(s)$$

$$L\{M\ddot{x}\} = M \cdot s^2 X(s) \quad \dots (6)$$

Transformada de u :

$$L\{u(t)\} = U(s) \quad \dots (7)$$

Transformada de $mg\theta$:

Se tiene la constante mg y $\theta(t)$ que sería la función dependiente del tiempo, por lo tanto, se tiene la transformada:

$$L\{mg\theta(t)\} = mg \cdot L\{\theta(t)\}$$

$$L\{mg\theta(t)\} = mg \cdot \Theta(s) \quad \dots (8)$$

Finalmente, se tiene la transformada para la ecuación (2):

$$L\{M\ddot{x}\} = L\{u\} - L\{mg\theta\}$$

Reemplazando (6), (7) y (8):

$$Ms^2 X(s) = U(s) - mg\Theta(s)$$

En resumen, se tienen las ecuaciones en el dominio de Laplace, para las ecuaciones (1) y (2), respectivamente:

$$Mls^2\Theta(s) = (M + m)g\Theta(s) - U(s) \quad \dots (9)$$

$$Ms^2X(s) = U(s) - mg\Theta(s) \quad \dots (10)$$

Funciones de transferencia

Con las ecuaciones en el dominio de Laplace obtenidas anteriormente, se pueden hallar las funciones de transferencia.

Para la ecuación (9):

Para la ecuación (9), se debe relacionar la salida $\theta(t)$ con la entrada $u(t)$, por lo tanto, se tiene lo siguiente:

$$Mls^2\Theta(s) = (M + m)g\Theta(s) - U(s)$$

$$\Theta(s) \cdot (Mls^2 - (M + m)g) = -U(s)$$

$$\Theta(s) = \frac{-U(s)}{Mls^2 - (M+m)g}$$

$$\frac{\Theta(s)}{U(s)} = \frac{-1}{Mls^2 - (M+m)g} \quad \dots (11)$$

Entonces, la función de transferencia para la ecuación (9) es $\frac{\Theta(s)}{U(s)} = \frac{-1}{(Mls^2 - (M+m)g)}$,

esto permite analizar cómo la salida del sistema (en este caso, el ángulo del péndulo $\Theta(s)$) responde a una entrada (en este caso, la fuerza $U(s)$ aplicada al carro).

Ahora, se procede a hallar los polos del sistema a partir de esta función de transferencia, por lo que se debe igualar el denominador a cero, ya que estos polos son los valores que hacen que la función de transferencia tienda a infinito.

$$Mls^2 - (M + m)g = 0$$

$$Mls^2 = (M + m)g$$

$$s^2 = \frac{(M+m)g}{Ml}$$

$$s = \pm \sqrt{\frac{(M+m)g}{Ml}}$$

Por lo tanto, se tienen los siguientes polos del sistema:

$$s_1 = \sqrt{\frac{(M+m)g}{Ml}}, s_2 = -\sqrt{\frac{(M+m)g}{Ml}}$$

Para la ecuación (10):

$$Ms^2 X(s) = U(s) - mg\Theta(s)$$

$$Ms^2 \frac{X(s)}{U(s)} = \frac{U(s)}{U(s)} - mg \frac{\Theta(s)}{U(s)}$$

De la otra función de transferencia ($\frac{\Theta(s)}{U(s)} = \frac{-1}{Mls^2 - (M+m)g}$):

$$Ms^2 \frac{X(s)}{U(s)} = 1 - mg \left(\frac{-1}{Mls^2 - (M+m)g} \right)$$

$$Ms^2 \frac{X(s)}{U(s)} = \frac{(Mls^2 - (M+m)g) + mg}{Mls^2 - (M+m)g}$$

$$Ms^2 \frac{X(s)}{U(s)} = \frac{Mls^2 - Mg}{Mls^2 - (M+m)g}$$

$$s^2 \frac{X(s)}{U(s)} = \frac{ls^2 - g}{Mls^2 - (M+m)g}$$

$$\frac{X(s)}{U(s)} = \frac{ls^2 - g}{s^2(Mls^2 - (M+m)g)} \quad \dots (12)$$

Entonces, la función de transferencia para la ecuación (10) es la siguiente:

$$\frac{X(s)}{U(s)} = \frac{ls^2 - g}{s^2(Mls^2 - (M+m)g)}, \text{ esto permite analizar cómo la salida del sistema (en este caso, la}$$

posición del carro ($X(s)$) responde a una entrada (la fuerza $U(s)$ aplicada al carro).

Ahora, se procede a hallar los polos del sistema a partir de esta función de transferencia:

$$s^2(Mls^2 - (M+m)g) = 0$$

$$s^2 = 0 \Rightarrow s = 0$$

$$Mls^2 - (M+m)g = 0$$

$$Mls^2 = (M + m)g$$

$$s^2 = \frac{(M+m)g}{Ml}$$

$$s = \pm \sqrt{\frac{(M+m)g}{Ml}}$$

Por lo tanto, se tienen los siguientes polos del sistema:

$$s_1 = 0, s_2 = \sqrt{\frac{(M+m)g}{Ml}}, s_3 = -\sqrt{\frac{(M+m)g}{Ml}}$$

De las funciones de transferencia y sus polos hallados en esta sección, se puede afirmar que el sistema es inestable a lazo abierto, esto quiere decir que, sin ningún tipo de control sobre el sistema, el sistema tiende a desviarse de un estado de equilibrio.

Esto se debe al polo positivo que se encontró en ambos casos, ya que este genera un crecimiento exponencial en la respuesta del sistema con el tiempo. En la función de transferencia de la ecuación (12), se tiene un polo cero, por lo que esto podría introducir algo de amortiguamiento, sin embargo, este no es suficiente para estabilizar el sistema. Lo mismo pasa con el polo negativo, no es suficiente para estabilizar el sistema en ambos casos.

Espacio de estados

En esta sección se representará el sistema en espacio de estados para modelar el comportamiento dinámico de este mediante las ecuaciones diferenciales dadas.

Como ya se vio anteriormente, este sistema se compone de dos subsistemas: el sistema del carro y el del péndulo, por lo que este sistema puede ser representado con las siguientes variables de estado:

$$x_1 = \theta$$

$$x_2 = \dot{\theta}$$

$$x_3 = x$$

$$x_4 = \dot{x}$$

Para representar el espacio de estados, se derivan las variables de estado:

$$\dot{x}_1 = \dot{\theta} = x_2$$

$$\dot{x}_2 = \ddot{\theta}$$

$$\dot{x}_3 = \dot{x} = x_4$$

$$\dot{x}_4 = \ddot{x}$$

De lo anterior, se busca representar el sistema con la siguiente forma:

$$\dot{x} = Ax + Bu$$

Donde A es la matriz de estado, x es el vector de estado, B es la matriz de entrada y u es el vector de entrada. Entonces, se utilizan las ecuaciones (1) y (2):

$$Ml\ddot{\theta} = (M + m)gx_1 - u$$

$$M\ddot{x} = u - mg\theta$$

Se reescriben las ecuaciones (1) y (2):

$$Ml\dot{x}_2 = (M + m)gx_1 - u \quad \dots (13)$$

$$M\dot{x}_4 = u - mgx_1 \quad \dots (14)$$

Se despeja \dot{x}_2 y \dot{x}_4 :

$$\dot{x}_2 = \frac{(M+m)g}{Ml}x_1 - \frac{1}{Ml}u$$

$$\dot{x}_4 = -\frac{mg}{M}x_1 + \frac{1}{M}u$$

Además, se sabe que x y θ son las salidas del sistema, por lo que también se puede representar esto en forma matricial y siguiendo la forma que ya se mencionó:

$$Y = Cx + Du$$

A continuación, se define el vector de estado x y la salida del sistema Y :

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$y = \begin{bmatrix} x \\ \theta \end{bmatrix}$$

Ahora, podemos agrupar los términos de x para darle forma a la ecuación de \dot{x} y hallar la matriz de estado A :

$$\dot{x} = Ax + Bu$$

Además, se tiene lo siguiente:

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix}$$

$$\dot{x} = Ax + Bu$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = A \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + Bu$$

De lo que se halló anteriormente, se puede reemplazar:

$$\begin{bmatrix} \dot{x}_2 \\ \frac{(M+m)g}{Ml}x_1 - \frac{1}{Ml}u \\ \dot{x}_4 \\ -\frac{mg}{M}x_1 + \frac{1}{M}u \end{bmatrix} = A \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + Bu$$

$$\begin{bmatrix} \dot{x}_2 \\ \frac{(M+m)g}{Ml}x_1 - \frac{1}{Ml}u \\ \dot{x}_4 \\ -\frac{mg}{M}x_1 + \frac{1}{M}u \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{(M+m)g}{Ml} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{mg}{M} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{Ml} \\ 0 \\ \frac{1}{M} \end{bmatrix} u$$

Se ha obtenido la forma que se quería conseguir, por lo que se pueden obtener los valores y A y B :

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{(M+m)g}{Ml} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{mg}{M} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{Ml} \\ 0 \\ \frac{1}{M} \end{bmatrix} u$$

$$\dot{x} = Ax + Bu$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{(M+m)g}{Ml} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{mg}{M} & 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ -\frac{1}{Ml} \\ 0 \\ \frac{1}{M} \end{bmatrix}$$

Para el caso de la matriz de salida se tiene lo siguiente:

$$Y = Cx + Du$$

$$\begin{bmatrix} \theta \\ x \end{bmatrix} = C \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + Du$$

$$\begin{bmatrix} x_1 \\ x_3 \end{bmatrix} = C \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$Y = Cx + Du$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, D = 0$$

De lo anterior, se tienen las ecuaciones vectoriales que describen el espacio de estados del péndulo invertido:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{(M+m)g}{Ml} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{mg}{M} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{Ml} \\ 0 \\ \frac{1}{M} \end{bmatrix} u$$

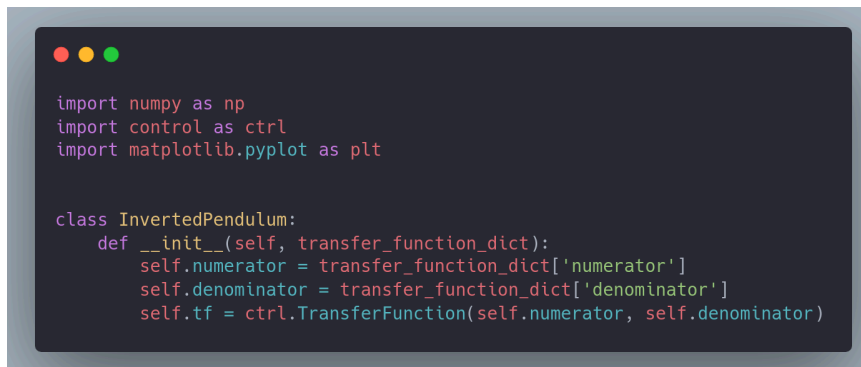
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

Código

En esta sección se detalla cómo se implementó el sistema del péndulo invertido en Python, con las diferentes clases que requiere para su funcionamiento y su relación con las funciones de transferencia halladas anteriormente.

Figura 1

Constructor de la clase InvertedPendulum

A screenshot of a code editor with a dark background and light-colored text. The code defines the InvertedPendulum class and its __init__ method. The imports are at the top, followed by the class definition and the __init__ method. The method takes a transfer_function_dict as an argument and assigns its 'numerator' and 'denominator' values to self.numerator and self.denominator respectively. It then creates a TransferFunction object from the control library using these values and assigns it to self.tf.

```
import numpy as np
import control as ctrl
import matplotlib.pyplot as plt

class InvertedPendulum:
    def __init__(self, transfer_function_dict):
        self.numerator = transfer_function_dict['numerator']
        self.denominator = transfer_function_dict['denominator']
        self.tf = ctrl.TransferFunction(self.numerator, self.denominator)
```

En la Figura 1 se muestra el constructor de la clase InvertedPendulum, la cual representa el sistema del péndulo invertido. En primer lugar, se realizan las importaciones necesarias (numpy, control y matplotlib.pyplot) para el funcionamiento de esta clase. Luego, se tiene la clase y su método `__init__`, el cual recibe un diccionario que representa la función de transferencia, este diccionario debe tener los atributos “numerator” y “denominator”, los cuales representan el numerador y el denominador de la función de transferencia, respectivamente. Además, estos atributos se pasan en forma de lista, ya que son los coeficientes). Luego, se asigna la función de transferencia y se utiliza la librería control (ctrl), específicamente la clase TransferFunction, el cual genera un modelo matemático del sistema en forma de la función de transferencia.

Figura 2

Método `get_step_response` de la clase InvertedPendulum



```
# Simulamos la respuesta del sistema sin control (lazo abierto)
def get_step_response(self, t_end=5, steps=500):
    # Definimos el tiempo de simulación con np.linspace
    time = np.linspace(0, t_end, steps)
    # Obtenemos la respuesta al escalón utilizando step_response
    time, response = ctrl.step_response(self.tf, T=time)

    return time, response
```

En la Figura 2 se muestra el método `get_step_response`, el cual simula y obtiene la respuesta al escalón del sistema a lazo abierto (sin control). Este tipo de respuesta se da para una entrada de escalón unitario, es decir, una señal que permanece en cero hasta un tiempo inicial (generalmente es $t = 0$), luego de este instante de tiempo, cambia a un valor constante de 1, lo que permite simular una entrada repentina en el sistema.

Este método toma como parámetros el tiempo total de simulación (`t_end`) y el número de puntos de evaluación (`steps`), los cuales son 5 y 500 respectivamente si no se pasan otros valores. El método `linspace` de la librería `numpy` sirve para generar un arreglo de *steps* valores distribuidos uniformemente desde 0 hasta t_{end} . Luego, el método `step_response` de la librería `control` sirve para calcular cómo responde el sistema a una entrada de tipo escalón unitario. Aquí se le pasa la función de transferencia y los instantes de tiempo. Finalmente, el método retorna estos valores: los instantes de tiempo de la simulación (`time`) y la respuesta del sistema en esos instantes (`response`).

Figura 3

Método plot_response de la clase InvertedPendulum

```
# Graficamos la respuesta del sistema
@staticmethod
def plot_response(time, response, y_label, title="Respuesta del sistema"):
    plt.figure()
    plt.plot(time, response)
    plt.title(title)
    plt.xlabel("Tiempo (s)")
    plt.ylabel(y_label)
    plt.grid()
    plt.show()
```

En la Figura 3 se muestra el método estático `plot_response`, el cual genera una gráfica que muestra la evolución de una respuesta del sistema en función del tiempo, lo cual es útil para analizar su comportamiento dinámico. Recibe como parámetros el arreglo de tiempos (`time`), el arreglo de respuestas (`response`), una etiqueta para el eje y (`y_label`), y un título opcional para la gráfica (`title`, que por defecto es "Respuesta del sistema"). Para realizar la gráfica, se utiliza la biblioteca Matplotlib y le añade etiquetas, un título y una cuadrícula.

Figura 4

Constructor de la clase PendulumAnimation

```
import plotly.graph_objects as go
import numpy as np

class PendulumAnimation:
    def __init__(self, time_steps, response, rod_length, cart_motion=None):
        self.time_steps = time_steps # Array de tiempos
        self.response = response # Array de respuestas
        self.rod_length = rod_length # Longitud de la varilla del péndulo
        self.cart_motion = cart_motion # Array opcional que define el desplazamiento del carrito en cada paso
```

En la Figura 4 se muestra el método `__init__` de la clase `PendulumAnimation`, el cual inicializa los parámetros necesarios para crear una animación que representa el comportamiento de un péndulo invertido montado sobre un carrito. Este constructor recibe cuatro argumentos: `time_steps`, un arreglo que contiene los instantes de tiempo de la

simulación; response, un arreglo con los valores de la respuesta del sistema (puede ser el ángulo del péndulo en cada instante); rod_length, que representa la longitud de la varilla del péndulo; y cart_motion, que es un argumento opcional que puede incluir un arreglo con la posición del carrito a lo largo del tiempo.

Figura 5

Método create_animation de la clase PendulumAnimation

```
def create_animation(self):
    frames = [] # Lista para almacenar los frames de la animación
    x_data = 0 # Posición inicial del carrito

    for i in range(len(self.time_steps)):
        # Calculamos el ángulo y ajusta para la posición vertical invertida
        theta = self.response[i]
        adjusted_theta = np.pi - theta

        # Si no se proporciona un cart_motion, calcula dinámicamente el movimiento
        if self.cart_motion is None:
            x_data += 0.05 * np.sin(theta)
        else:
            x_data = self.cart_motion[i]

        # Coordenadas del péndulo
        pendulum_x = [x_data, x_data + self.rod_length * np.sin(adjusted_theta)]
        pendulum_y = [0, -self.rod_length * np.cos(adjusted_theta)]

        # Creamos un frame para la animación
        frames.append(go.Frame(data=[
            # Dibujamos el carrito
            go.Scatter(x=[x_data - 0.1, x_data + 0.1], y=[0, 0], mode='lines', line=dict(width=10)),
            # Dibujamos el péndulo
            go.Scatter(x=pendulum_x, y=pendulum_y, mode='lines+markers', line=dict(width=4),
            marker=dict(size=8))
        ]))

    # Creamos la figura de la animación
    fig = go.Figure(
        data=[
            go.Scatter(x=[0, 0], y=[0, 0], mode='lines', line=dict(width=10)),
            go.Scatter(x=[0, 0], y=[0, 0], mode='lines+markers', line=dict(width=4), marker=dict(size=8))
        ],
        layout=go.Layout(
            xaxis=dict(range=[-2, 2], autorange=False),
            yaxis=dict(range=[-1.5, 1.5], autorange=False),
            updatemenus=[dict(type="buttons", showactive=False,
                buttons=[dict(label="Play", method="animate",
                    args=[None, {"frame": {"duration": 20}, "fromcurrent":
                    True}])]),
            frames=frames
        )
    )

    fig.show()
```

En la Figura 5 se muestra el método create_animation, el cual genera una animación visual del comportamiento dinámico de un péndulo invertido utilizando Plotly. Para cada instante de tiempo en time_steps, calcula las coordenadas del carrito y del péndulo basándose en la respuesta angular del sistema (response) y la longitud de la varilla (rod_length). Si no se

proporciona un desplazamiento explícito del carrito (`cart_motion`), este se calcula dinámicamente según la inclinación del péndulo. A partir de estas coordenadas, se crea un conjunto de frames que representan la posición del carrito y el péndulo en cada instante. Luego, configura un objeto Figure que incluye los ejes, rangos y controles de la animación, y se muestra mediante `fig.show()`.

En el contexto de este proyecto, el objetivo principal es estabilizar el péndulo en su posición vertical, por lo que se utilizará un controlador PID en la siguiente sección. Para esto, se utilizará la primera función de transferencia del sistema, ya que esta describe la dinámica de la posición angular del péndulo en respuesta a una entrada aplicada al carrito. En cambio, la segunda función de transferencia describe la dinámica del sistema desde el punto de vista de la posición del carrito y la fuerza aplicada, por lo que el análisis de este aspecto no es muy relevante para el objetivo de este proyecto. Por esta razón, la implementación del controlador PID y su optimización se realizará teniendo en cuenta la primera función de transferencia.

4. Diseño e Implementación del Controlador PID

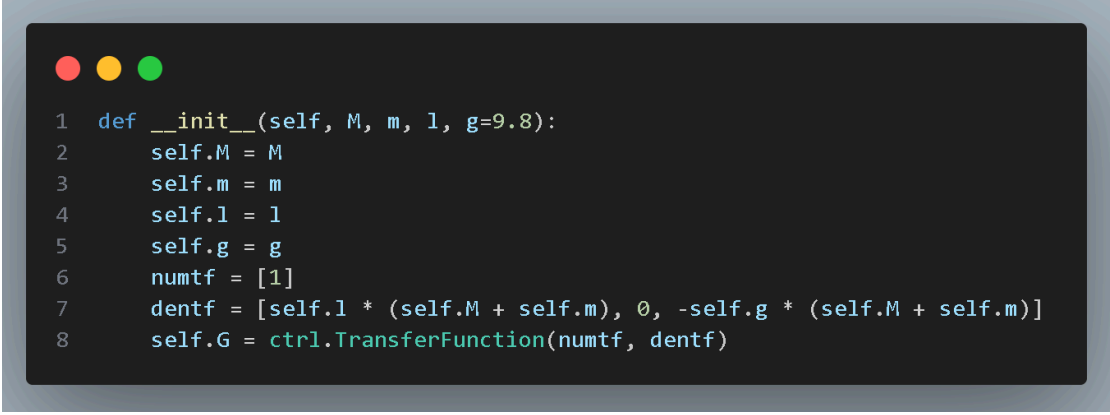
En esta sección se describe el diseño e implementación del controlador PID aplicado al sistema de péndulo invertido. Se detalla cómo se integran los métodos y clases desarrollados, así como los impactos de los parámetros de control (K_p , K_i , K_d) en la respuesta dinámica del sistema. Además, se analizan los comportamientos individuales de los controladores P, PI, PD y PID mediante simulaciones, con énfasis en el análisis de estabilización, sobreimpulso y respuesta temporal.

Implementación de la Clase PendulumSystem

La clase `PendulumSystem` modela el péndulo invertido y permite simular su comportamiento bajo diferentes estrategias de control.

Figura 7

Inicialización de la clase PendulumSystem

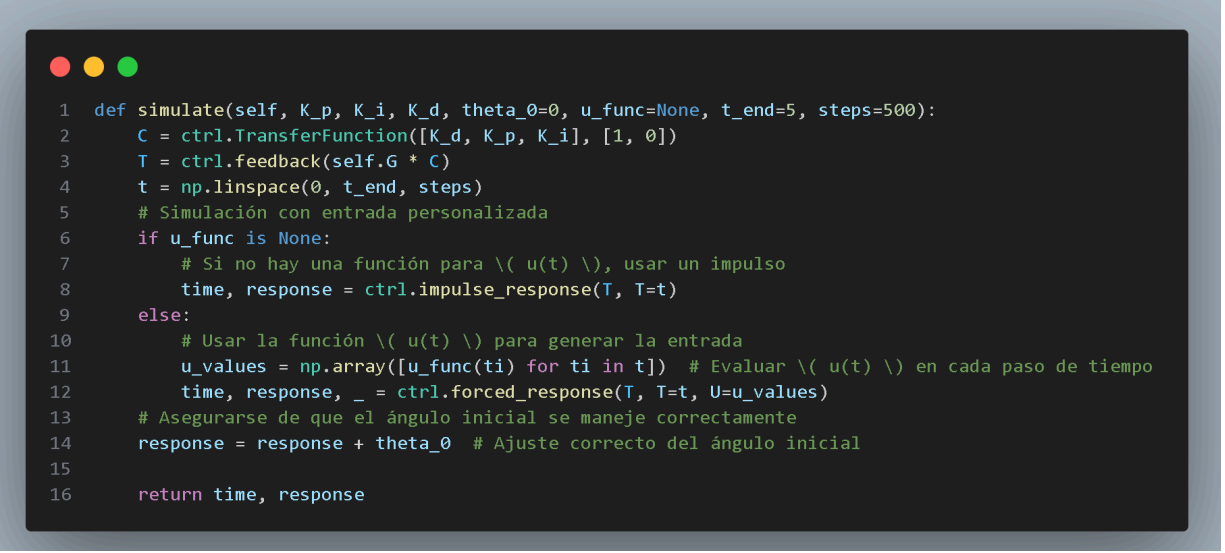


```
1 def __init__(self, M, m, l, g=9.8):
2     self.M = M
3     self.m = m
4     self.l = l
5     self.g = g
6     numtf = [1]
7     dentf = [self.l * (self.M + self.m), 0, -self.g * (self.M + self.m)]
8     self.G = ctrl.TransferFunction(numtf, dentf)
```

El método inicializa los parámetros fundamentales del sistema (masa del carrito, masa del péndulo, longitud) y construye una función de transferencia matemática (ver Figura 7). Esto simplifica el análisis dinámico al transformar el modelo mecánico en un sistema controlable mediante ecuaciones diferenciales.

Figura 8

Método para simular la respuesta del sistema



```
1 def simulate(self, K_p, K_i, K_d, theta_0=0, u_func=None, t_end=5, steps=500):
2     C = ctrl.TransferFunction([K_d, K_p, K_i], [1, 0])
3     T = ctrl.feedback(self.G * C)
4     t = np.linspace(0, t_end, steps)
5     # Simulación con entrada personalizada
6     if u_func is None:
7         # Si no hay una función para \(\ u(t) \), usar un impulso
8         time, response = ctrl.impulse_response(T, T=t)
9     else:
10        # Usar la función \(\ u(t) \) para generar la entrada
11        u_values = np.array([u_func(ti) for ti in t]) # Evaluar \(\ u(t) \) en cada paso de tiempo
12        time, response, _ = ctrl.forced_response(T, T=t, U=u_values)
13        # Asegurarse de que el ángulo inicial se maneje correctamente
14        response = response + theta_0 # Ajuste correcto del ángulo inicial
15
16    return time, response
```

El método simulate (ver Figura 8) evalúa la respuesta dinámica del péndulo bajo diferentes estrategias de control PID y genera simulaciones mediante dos mecanismos: respuesta de impulso (predeterminada) o respuesta forzada con función de entrada

personalizada. El método calcula la evolución temporal del ángulo del péndulo, incorporando ganancias proporcional, integral y derivativa, junto con condiciones iniciales específicas.

Implementación de la Clase PendulumApp

Esta clase configura el entorno interactivo de experimentación, integrando elementos de Tkinter para control y Matplotlib para visualización.

Figura 9

Inicialización de la clase PendulumApp

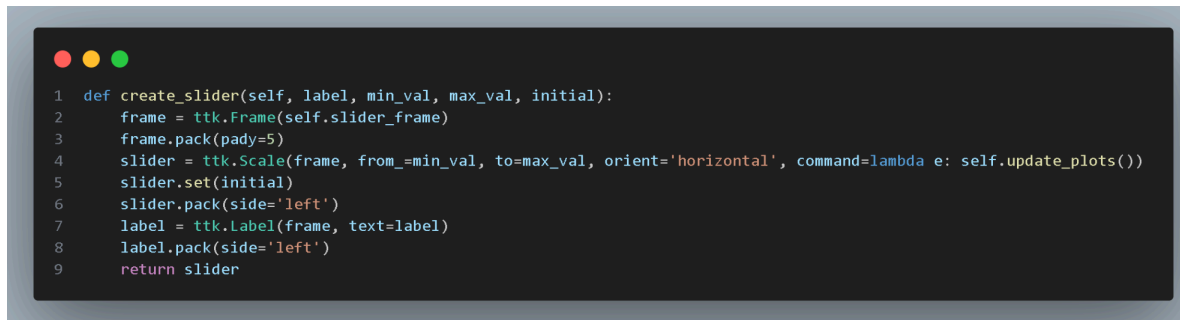
```
1 def __init__(self, root):
2     self.root = root
3     self.root.title("Controlador de Péndulo Invertido")
4     # Crear un marco para los sliders
5     self.slider_frame = ttk.Frame(root)
6     self.slider_frame.pack(side='left', padx=10, pady=10)
7     # Inicializar parámetros del sistema del péndulo
8     self.M = 1.0 # Masa del carrito
9     self.m = 0.1 # Masa del péndulo
10    self.l = 1.0 # Longitud del péndulo
11    self.theta_0 = 0.5 # Ángulo inicial
12    # Crear sliders para M, m y l
13    self.M_slider = self.create_slider("M (Masa del Carrito)", 0.1, 5.0, self.M)
14    self.m_slider = self.create_slider("m (Masa del Péndulo)", 0.01, 2.0, self.m)
15    self.l_slider = self.create_slider("l (Longitud del Péndulo)", 0.1, 2.0, self.l)
16    # Crear sliders para Kp, Ki y Kd
17    self.Kp_slider = self.create_slider("Kp", 0, 100, 20)
18    self.Ki_slider = self.create_slider("Ki", 0, 10, 0.7)
19    self.Kd_slider = self.create_slider("Kd", 0, 20, 5)
20    # Botón para actualizar las gráficas
21    self.update_button = ttk.Button(self.slider_frame, text="Actualizar Gráficas", command=self.update_plots)
22    self.update_button.pack(pady=10)
23    # Crear un marco para las gráficas
24    self.plot_frame = ttk.Frame(root)
25    self.plot_frame.pack(side='right', padx=10, pady=10)
26    # Crear las figuras para las gráficas
27    self.fig, self.axs = plt.subplots(2, 2, figsize=(10, 8))
28    self.canvas = FigureCanvasTkAgg(self.fig, master=self.plot_frame)
29    self.canvas.get_tk_widget().pack()
30    # Graficar inicialmente
31    self.update_plots()
```

El método `__init__` de la clase `PendulumApp` (ver Figura 9) se encarga de inicializar la interfaz gráfica y configurar el entorno para la simulación del sistema de péndulo invertido. Este método establece la estructura principal de la aplicación, definiendo un marco para los controles deslizantes y botones que permiten al usuario interactuar con los parámetros del sistema y del controlador PID. Adicionalmente, inicializa los valores predeterminados de los parámetros físicos del péndulo, como la masa del carrito (M), la masa del péndulo (m), la

longitud del péndulo (l), y el ángulo inicial (θ). Asimismo, se configura un área dedicada a las gráficas, donde se visualizarán las respuestas del sistema bajo diferentes configuraciones de control.

Figura 10

Método para la creación de sliders

A screenshot of a code editor with a dark background and light-colored text. The code is a Python method named create_slider. It takes five arguments: self, label, min_val, max_val, and initial. The method creates a Tkinter frame, adds a slider widget with a horizontal orientation and a command to call self.update_plots(), sets the initial value, packs the slider to the left, creates a label, packs the label to the left, and returns the slider.

```
1 def create_slider(self, label, min_val, max_val, initial):
2     frame = ttk.Frame(self.slider_frame)
3     frame.pack(pady=5)
4     slider = ttk.Scale(frame, from_=min_val, to=max_val, orient='horizontal', command=lambda e: self.update_plots())
5     slider.set(initial)
6     slider.pack(side='left')
7     label = ttk.Label(frame, text=label)
8     label.pack(side='left')
9     return slider
```

El método `create_slider` (ver Figura 10) es responsable de crear controles deslizantes (sliders) asociados a parámetros específicos del sistema o a las ganancias del controlador PID. Cada slider se configura con un rango de valores permitido, un valor inicial, y una etiqueta descriptiva para facilitar su identificación. Estos sliders están vinculados dinámicamente al método encargado de actualizar las gráficas, permitiendo que cualquier cambio en su valor se refleje de inmediato en la simulación. Este enfoque asegura una interacción fluida entre el usuario y el sistema, al tiempo que estandariza la creación de sliders para mantener la consistencia visual y funcional de la aplicación.

Por su parte, el método `update_plots` (ver Figura 11) es el núcleo de la funcionalidad dinámica de la aplicación. Este método recupera los valores actuales de los sliders y los utiliza para configurar una nueva instancia del sistema mediante la clase `PendulumSystem`. A continuación, simula el comportamiento del sistema bajo cuatro configuraciones de control: P, PI, PD y PID. Para cada configuración, calcula la respuesta temporal utilizando el método `simulate` y representa gráficamente los resultados en subgráficas independientes.

Figura 11

Método `update_plots`

```
1 def update_plots(self):
2     try:
3         # Obtener valores de los sliders
4         M = self.M_slider.get()
5         m = self.m_slider.get()
6         l = self.l_slider.get()
7         Kp = self.Kp_slider.get()
8         Ki = self.Ki_slider.get()
9         Kd = self.Kd_slider.get()
10        # Actualizar el sistema del péndulo con los nuevos parámetros
11        self.pendulum_system = PendulumSystem(M, m, l)
12        # Limpiar las gráficas
13        for ax in self.axes.flatten():
14            ax.clear()
15        # Simulaciones para cada tipo de controlador
16        controllers = {
17            'P': (Kp, 0, 0),
18            'PI': (Kp, Ki, 0),
19            'PD': (Kp, 0, Kd),
20            'PID': (Kp, Ki, Kd)
21        }
22        for i, (ctrl_type, gains) in enumerate(controllers.items()):
23            Kp, Ki, Kd = gains
24            time, response = self.pendulum_system.simulate(Kp, Ki, Kd, theta_0=self.theta_0)
25            ax = self.axes[i // 2, i % 2]
26            ax.plot(time, response, label=ctrl_type)
27            ax.set_title(f'Respuesta del Controlador {ctrl_type}')
28            ax.set_xlabel('Tiempo (s)')
29            ax.set_ylabel('Ángulo  $\theta$  (rad)')
30            ax.grid()
31            ax.legend()
32        self.canvas.draw()
33    except AttributeError as e:
34        print(f"-")
```

Preguntas:

- a. ¿Cómo influyen los parámetros individuales K_p , K_i y K_d en la respuesta del sistema?

Los parámetros K_p , K_i y K_d tienen roles específicos en la respuesta del sistema. La ganancia proporcional determina la magnitud de la reacción del sistema ante un error. Un valor bajo de K_p resulta en una estabilización lenta, mientras que un valor elevado puede inducir oscilaciones significativas alrededor del punto de equilibrio debido a una respuesta excesivamente agresiva.

Por su parte, la ganancia integral (K_i) actúa acumulando el error a lo largo del tiempo, lo que permite eliminar el error en estado estacionario. Un K_i bajo hace que el péndulo no

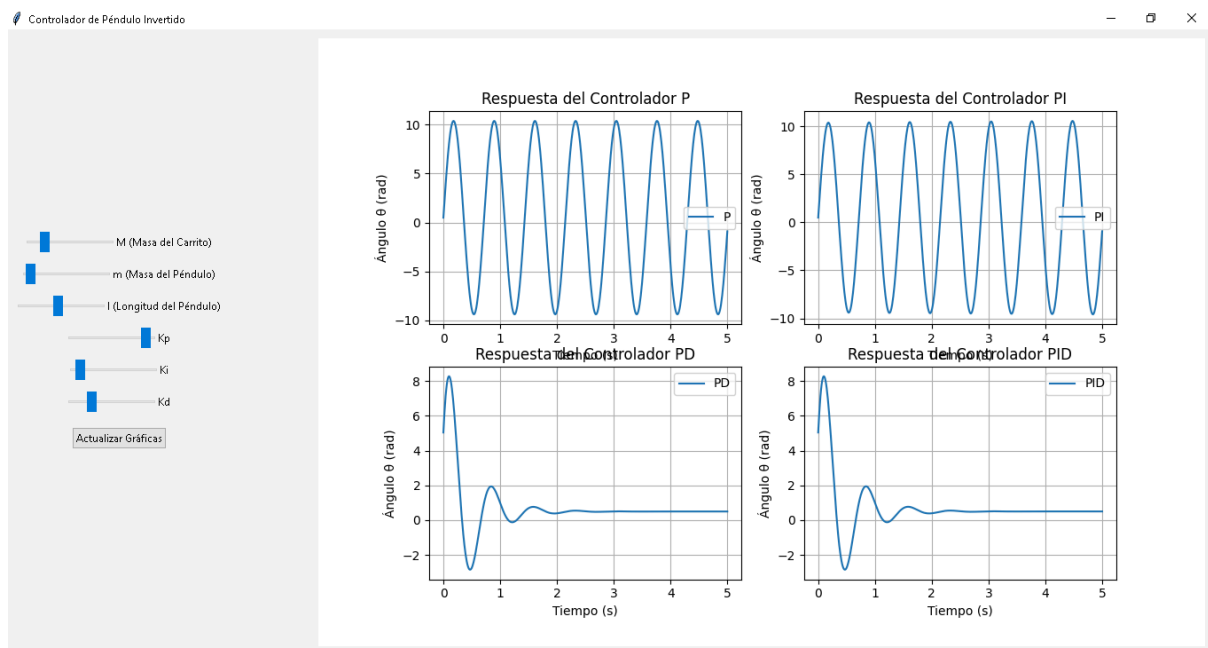
llegue exactamente a la posición vertical; es decir, produce un error persistente en estado estacionario, mientras que los valores altos provocan oscilaciones alrededor del equilibrio debido a la acumulación excesiva de error anterior.

Finalmente, la ganancia derivativa mide la velocidad de cambio del error y agrega amortiguamiento al sistema. Si K_d es bajo, puede hacer que el péndulo oscile mucho antes de estabilizarse, dificultando el control de oscilaciones iniciales, pero si el valor es demasiado alto, el sistema se vuelve excesivamente lento ante perturbaciones.

Los gráficos de los controladores (Figura 12) muestran cómo varía el comportamiento del sistema al aplicar cada controlador, evidenciando las ventajas y limitaciones en diferentes contextos de operación.

Figura 12

Gráficos de los controladores



b. ¿Qué diferencias se observan en el comportamiento del sistema al usar controladores P, PI, PD y PID?

Los diferentes controladores afectan la estabilidad, precisión y dinámica del sistema de péndulo invertido:

Controlador P (Proporcional)

La acción de control es proporcional al error, proporcionando una respuesta rápida, pero deja un error persistente en estado estacionario y es sensible a perturbaciones, causando inestabilidad en sistemas más complejos.

Controlador PI (Proporcional-Integral)

Es la evolución del controlador proporcional con un término integral integrado para compensar el error en estado estacionario. Este término integral permite una eliminación más efectiva de la desviación persistente, mejorando la precisión del sistema. No obstante, introduce el riesgo de oscilaciones y potencial reducción del margen de estabilidad.

Controlador PD (Proporcional-Derivativo)

Como una estrategia de control, este controlador adiciona un término derivativo para mejorar la dinámica del sistema. El componente derivativo proporciona un efecto de amortiguamiento crítico, reduciendo significativamente las oscilaciones y optimizando la respuesta transitoria. Su principal limitación radica en la sensibilidad potencial al ruido de la señal.

Controlador PID (Proporcional-Integral-Derivativo)

Se considera el paradigma de control más versátil y robusto para sistemas de péndulo invertido. La sinergia entre los términos proporcional, integral y derivativo permite una compensación simultánea de múltiples aspectos dinámicos:

- Término Proporcional: Mejora la respuesta transitoria
- Término Integral: Elimina errores en estado estacionario
- Término Derivativo: Proporciona amortiguamiento y estabilidad

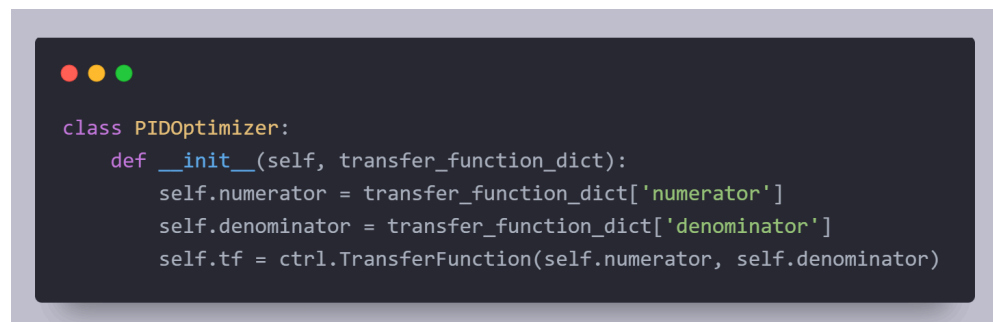
5. Optimización del PID con Algoritmos Genéticos

El algoritmo genético utilizado para la optimización del PID fue el *Differential Evolution* que nos lo facilita la librería “scipy.optimize”; se ha optado por este algoritmo

debido a que es capaz de probar múltiples valores para K_p , K_i y K_d y es resistente a datos ruidosos, lo que lo hace una opción puesto que en nuestro sistema tenemos distintas variables como la masa del carro, longitud del péndulo, gravedad, ángulo, entre otros. La clase que va a utilizar este código toma como parámetro la función de transferencia que se define anteriormente con un numerador y un denominador.

Figura 13

Inicialización de la clase PIDOptimizer

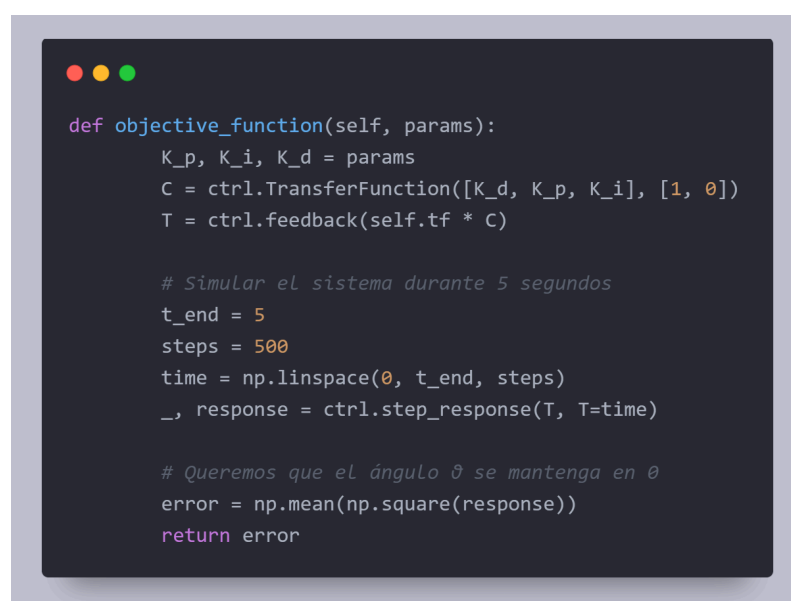


```
class PIDOptimizer:
    def __init__(self, transfer_function_dict):
        self.numerator = transfer_function_dict['numerator']
        self.denominator = transfer_function_dict['denominator']
        self.tf = ctrl.TransferFunction(self.numerator, self.denominator)
```

En este caso, el error se calcula mediante el error cuadrático medio como se muestra en el código de la Figura 13 en la variable *error*. Esto es entre la salida del sistema y el valor deseado del ángulo que es 0 dado que se desea mantener el ángulo en equilibrio.

Figura 14

Método objective_function de la clase PIDOptimizer



```
def objective_function(self, params):
    K_p, K_i, K_d = params
    C = ctrl.TransferFunction([K_d, K_p, K_i], [1, 0])
    T = ctrl.feedback(self.tf * C)

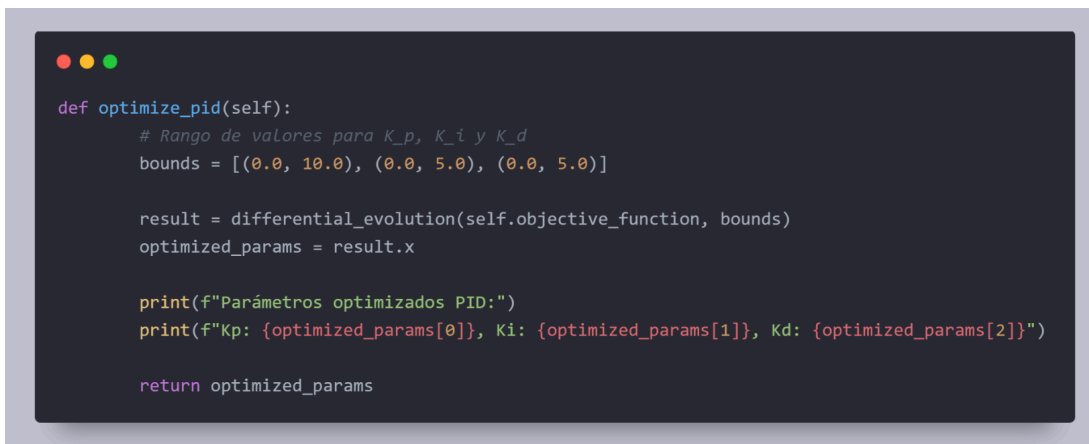
    # Simular el sistema durante 5 segundos
    t_end = 5
    steps = 500
    time = np.linspace(0, t_end, steps)
    _, response = ctrl.step_response(T, T=time)

    # Queremos que el ángulo  $\theta$  se mantenga en 0
    error = np.mean(np.square(response))
    return error
```

En la Figura 14 usamos K_p , K_i y K_d como parámetros así como la librería *ctrl* que nos ayuda a obtener la función de transferencia del PID y luego mediante la función *feedback* lo que se busca es ajustar continuamente el error. Con la variable T que representa al sistema simulado (comportamiento de cuando el ángulo intenta nivelarse o volver a 0) procedemos a calcular *response* que representa cómo el ángulo reacciona a través del tiempo determinado por t_{end} y *steps* que vendrían a significar 5 segundos.

Figura 15

Método `optimize_pid` de la clase `PIDOptimizer`



```
def optimize_pid(self):
    # Rango de valores para K_p, K_i y K_d
    bounds = [(0.0, 10.0), (0.0, 5.0), (0.0, 5.0)]

    result = differential_evolution(self.objective_function, bounds)
    optimized_params = result.x

    print(f"Parámetros optimizados PID:")
    print(f"Kp: {optimized_params[0]}, Ki: {optimized_params[1]}, Kd: {optimized_params[2]}")

    return optimized_params
```

En la Figura 15 se muestra el código donde definimos los rangos de valores de K_p , K_i y K_d para luego aplicar el algoritmo de evolución diferencial proporcionado por la librería *scipy* que en este caso le pasamos el error representado por la función *objective_function* mencionada anteriormente y *bounds* que son los valores límites; esto es porque el algoritmo de evolución diferencial busca valores óptimos que en este caso son K_p , K_i y K_d partiendo de una población inicial que son valores aleatorios dentro del rango dado, usa el error para evaluarlas y la “evolución” donde las posibles soluciones se combinan para dar otras nuevas soluciones como resultado, al final se seleccionan las mejores y se repite durante determinados ciclos o generaciones. Por último retorna los parámetros optimizados.

Figura 16

Método `simulate_with_optimized_pid` de la clase `PIDOptimizer`

```
def simulate_with_optimized_pid(self, K_p, K_i, K_d):  
    C = ctrl.TransferFunction([K_d, K_p, K_i], [1, 0])  
    T = ctrl.feedback(self.tf * C)  
  
    t_end = 5  
    steps = 500  
    time = np.linspace(0, t_end, steps)  
    _, response = ctrl.step_response(T, T=time)  
  
    return K_p, K_i, K_d
```

La anterior figura representa un método que en resumidas cuentas es para obtener los valores K_p , K_i y K_d para posteriormente utilizarlos en la ejecución principal del proyecto así como las otras simulaciones que se tienen en otras clases dentro del código.

Para llevar a cabo la correcta optimización se añadió un nuevo método a la clase `InvertedPendulum` declarada inicialmente. El cambio se puede observar en la Figura 17.

Figura 17

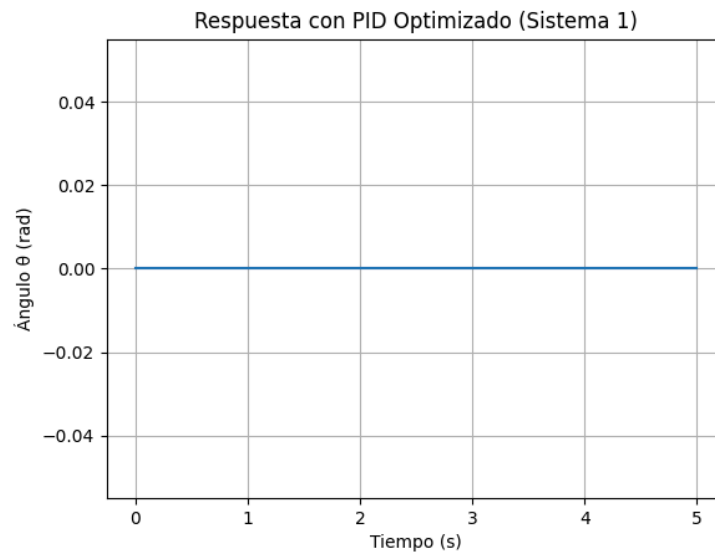
Método `simulate_with_pid` de la clase `InvertedPendulum`

```
def simulate_with_pid(self, K_p, K_i, K_d, t_end=5, steps=500):  
    """Simula la respuesta del sistema con un controlador PID (sistema cerrado)."""  
    # Controlador PID  
    C = ctrl.TransferFunction([K_d, K_p, K_i], [1, 0])  
    # Sistema en lazo cerrado  
    T = ctrl.feedback(self.tf * C)  
    time = np.linspace(0, t_end, steps)  
    time, response = ctrl.step_response(T, T=time)  
    return time, response
```

A continuación, se mostrará el gráfico del PID optimizado y cómo se muestra dentro de la simulación gráfica del carrito con el péndulo.

Figura 18

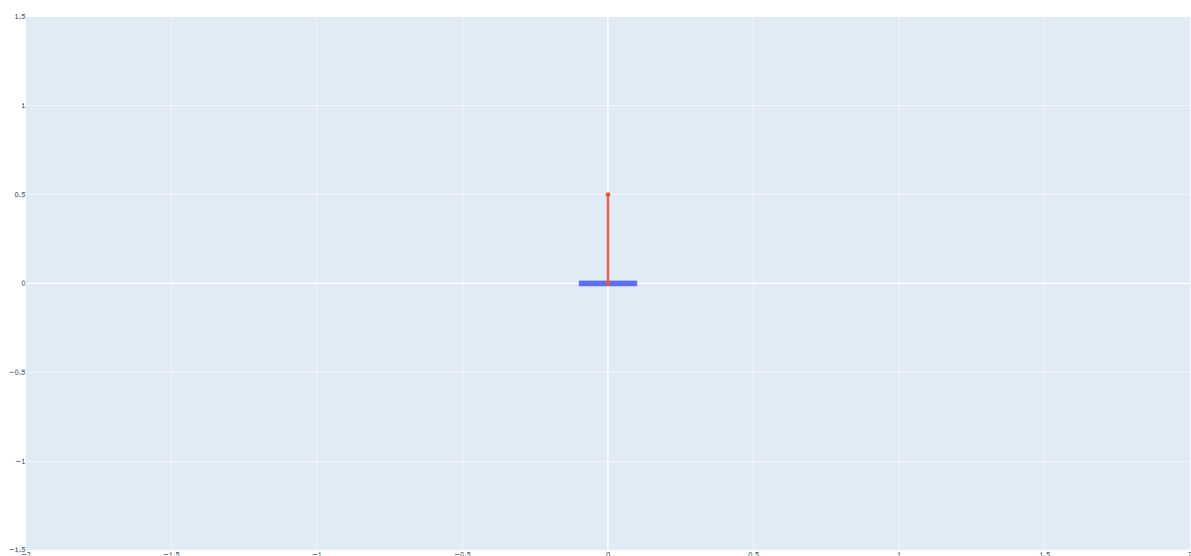
Gráfica de PID Optimizado



Dentro de la Figura 18, el gráfico muestra que el ángulo del péndulo se mantiene en 0, es decir, no se mueve o tambalea de una manera minúscula generando un ángulo despreciable para el sistema. Esto es debido a que los parámetros K_p , K_i y K_d han sido ajustados para minimizar el error y actuar rápidamente ante las perturbaciones del sistema para mantener en posición vertical al péndulo.

Figura 19

Simulación gráfica del PID Optimizado



En la animación del carrito con el péndulo, el péndulo permanece en posición vertical porque el controlador PID actúa constantemente para contrarrestar cualquier desviación del equilibrio. En comparación con un PID ajustado manualmente que puede contener valores K_p , K_i y K_d a criterio del usuario, el rendimiento del sistema para equilibrar el péndulo depende de los intentos que realice para ajustarlo lo que puede llevar a oscilaciones del péndulo en un momento determinado, un tiempo de estabilización prolongado y la probabilidad de que el péndulo no pueda alcanzar por completo el ángulo 0 mientras que con un PID optimizado mediante el algoritmo mostrado estas posibles complicaciones se mitigan o se eliminan dando como resultado un sistema más eficiente y equilibrado.

6. Optimización con Filtro de Kalman

El Filtro de Kalman es una herramienta para estimar estados no directamente medibles en sistemas dinámicos, como la velocidad angular o la posición precisa de un péndulo invertido. Este enfoque combina mediciones ruidosas del sistema con un modelo matemático de su dinámica para proporcionar estimaciones óptimas de los estados del sistema. En el caso del control del péndulo invertido, el filtro de Kalman se integra con un controlador PID para reducir el ruido en las mediciones y mejorar la estabilidad del sistema.

Al trabajar junto con un controlador PID, el filtro proporciona estados filtrados que para evitar que el ruido en las mediciones afecte las decisiones del controlador. Esto busca mejorar el desempeño del PID al reducir el sobreimpulso y el tiempo de estabilización del sistema.

Implementación y Configuración del Filtro de Kalman

El filtro de Kalman se implementa mediante las siguientes matrices:

- **Matriz de transición de estado A:** Define cómo evolucionan los estados del sistema basándose en el modelo del sistema:

$$A = \begin{bmatrix} 1 & \Delta \\ 0 & 1 \end{bmatrix}$$

Donde Δt es el intervalo de tiempo entre mediciones. Esta matriz relaciona la posición angular (θ) y la velocidad angular ($\dot{\theta}$).

- **Matriz de entrada B:** Representa cómo la entrada de control u_k afecta los estados.

$$B = \begin{bmatrix} 0 \\ \Delta t \end{bmatrix}$$

- **Matriz de observación C:** Relaciona las mediciones observadas con los estados internos del sistema ($y_k = Cx_k$). Esta matriz es usada para calcular la ganancia de Kalman K , que ajusta las estimaciones del estado basándose en la diferencia entre la medición observada y la predicción.
- **Matriz de covarianza del ruido del proceso Q:** Modela la incertidumbre en el modelo del sistema. Valores pequeños indican confianza en el modelo.

$$Q = \begin{bmatrix} 10^{-4} & 0 \\ 0 & 10^{-2} \end{bmatrix}$$

- **Matriz de covarianza del ruido de la medición R:** Representa la incertidumbre en las mediciones.

$$R = [0.1]$$

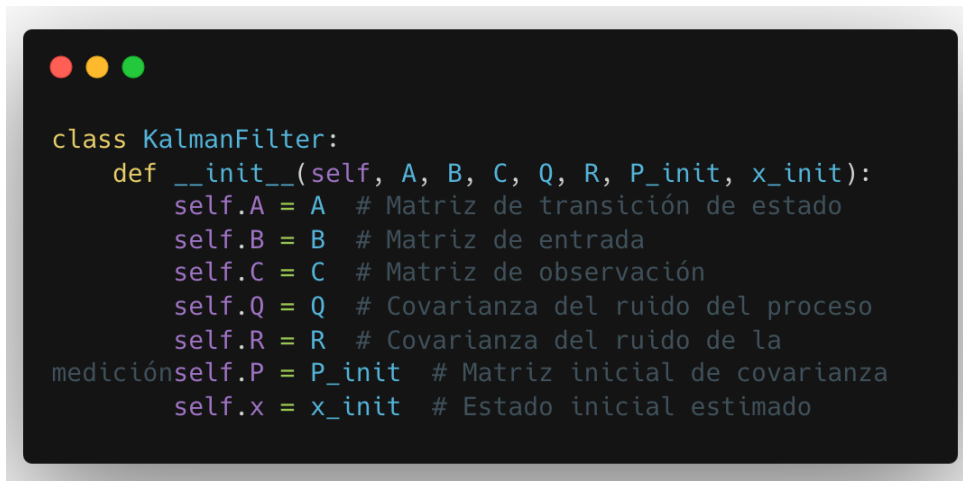
- **Matriz inicial de covarianza P_{init} :** Representa la incertidumbre inicial en los estados estimados. Es una matriz identidad.

$$P_{init} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Antes de iniciar las predicciones y correcciones, el filtro se configura con los parámetros clave que definen el modelo del sistema y las características del ruido (ver Figura 20).

Figura 20

Inicialización del Filtro de Kalman



```
class KalmanFilter:
    def __init__(self, A, B, C, Q, R, P_init, x_init):
        self.A = A # Matriz de transición de estado
        self.B = B # Matriz de entrada
        self.C = C # Matriz de observación
        self.Q = Q # Covarianza del ruido del proceso
        self.R = R # Covarianza del ruido de la
mediciónself.P = P_init # Matriz inicial de covarianza
        self.x = x_init # Estado inicial estimado
```


Cálculo de Predicción y Corrección

- **Predicción.** En esta etapa, el filtro predice el próximo estado del sistema y actualiza la incertidumbre asociada.
 - $x_{k+1,k} = Ax_k + Bu_k$: Predice el siguiente estado.
 - $P_{k+1,k} = AP_kA^T + Q$: Actualiza la incertidumbre del estado considerando Q.

El estado se calcula usando la matriz de transición (A) y la entrada de control (u_k). La incertidumbre actualizada refleja el avance en el tiempo y el ruido del modelo (Q) (ver Figura 21).

Figura 21

Método de Predicción del Filtro

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code for the predict method of a Kalman filter.

```
def predict(self, u):  
    self.x = self.A @ self.x + self.B @ u  
    # Predicción del estado  
    self.P = self.A @ self.P @ self.A.T + self.Q  
    # Actualización de la incertidumbre
```

- **Corrección.** Integra las mediciones observadas (y_k) para ajustar el estado predicho y reducir la incertidumbre.

- Cálculo de la ganancia de Kalman (K): Combina la predicción y la medición observada, ponderando en función de la confianza.

$$K_k = P_k C^T (C P_k C^T + R)^{-1}$$

- Actualización del estado estimado: Ajusta el estado con base en la diferencia entre medición y predicción.

$$x_{k,k} = x_{k,k-1} + K(y - Cx_{k,k-1})$$

- Actualización de la incertidumbre P : Reduce la incertidumbre después de integrar la medición.

$$P_k = (I - KC)P_k$$

Figura 22

Método de Actualización del Filtro

```
def update(self, y):  
    # Ganancia de Kalman  
    K = self.P @ self.C.T @ np.linalg.inv(self.C @ self.P @ self.C.T + self.R)  
    self.x = self.x + K @ (y - self.C @ self.x) # Ajuste del estado estimado  
    self.P = (np.eye(len(self.P)) - K @ self.C) @ self.P # Actualización de la  
    incertidumbre
```

La integración del filtro en el sistema de péndulo genera estimaciones suaves y confiables a partir de mediciones ruidosas (datos generados aleatoriamente para simular un entorno real). La señal de control del PID (u_k) se utiliza como entrada.

El sistema logra reducir significativamente el impacto del ruido en las mediciones (ver Sección de Análisis de Resultados), proporcionando estimaciones suaves y precisas que mejoran la respuesta del controlador PID. Esto permite que el sistema estabilice el péndulo con mayor precisión y robustez, incluso en condiciones con ruido moderado.

7. Análisis de Resultados

En esta sección se analizan y comparan los resultados obtenidos de diferentes configuraciones de control aplicadas a un sistema de péndulo invertido. Se evalúan casos sin control, con controladores P, PI, PD y PID (manuales y optimizados), así como el impacto del filtro de Kalman en combinación con el PID optimizado. Los análisis consideran la estabilidad, precisión y sensibilidad del sistema frente a variaciones en las condiciones iniciales, apoyándose en teoría de control para sustentar las observaciones.

Evaluación del desempeño

Sin control inicial

Utilizando el método de 'get_step_response' se simula la respuesta al escalón del sistema en lazo abierto, es decir, sin ningún controlador implementado (ver Fig 23, 24).

Figura 23

Gráfica de la Respuesta sin PID (posición)

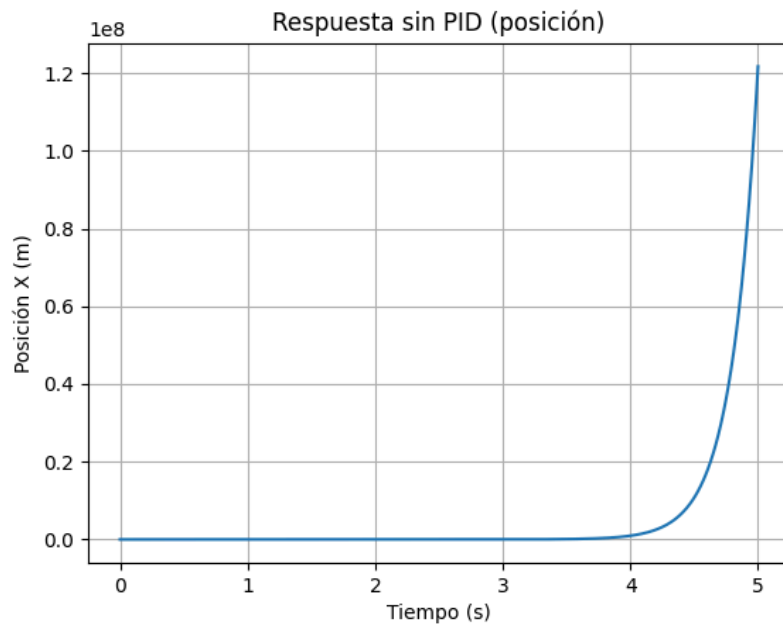
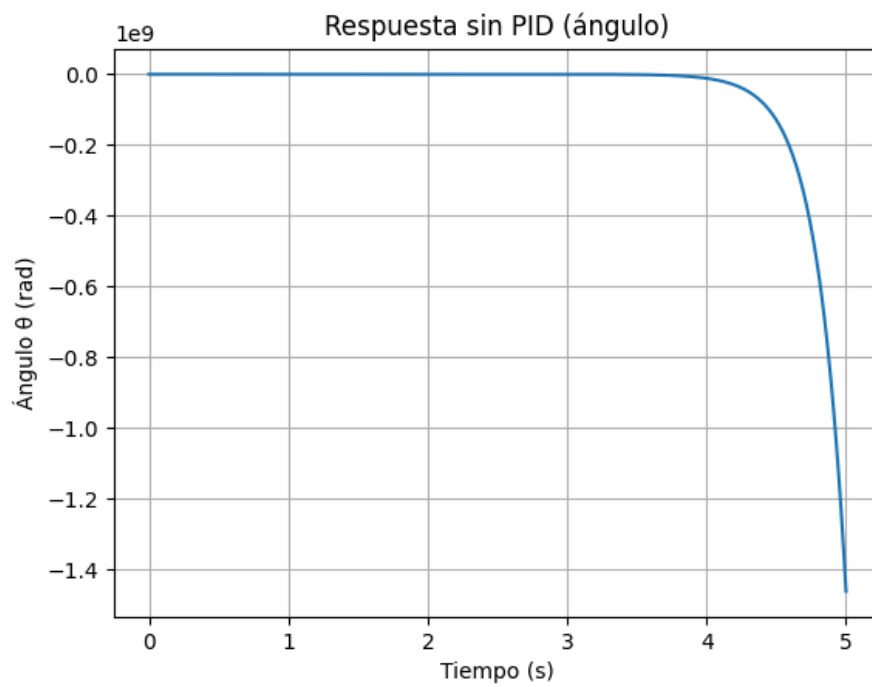


Figura 24

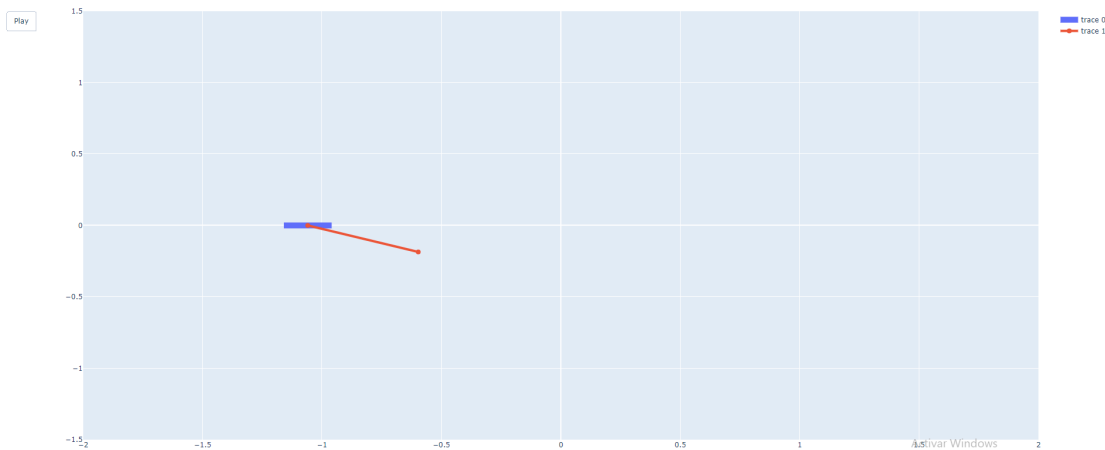
Gráfica de la Respuesta sin PID (ángulo)



El sistema muestra un comportamiento altamente inestable, donde el ángulo θ crece exponencialmente debido a que no hay ninguna estrategia de control implementada para mantener el equilibrio (ver Fig. 25).

Figura 25

Simulación gráfica sin PID



Comparación entre los controladores P, PI, PD y PID

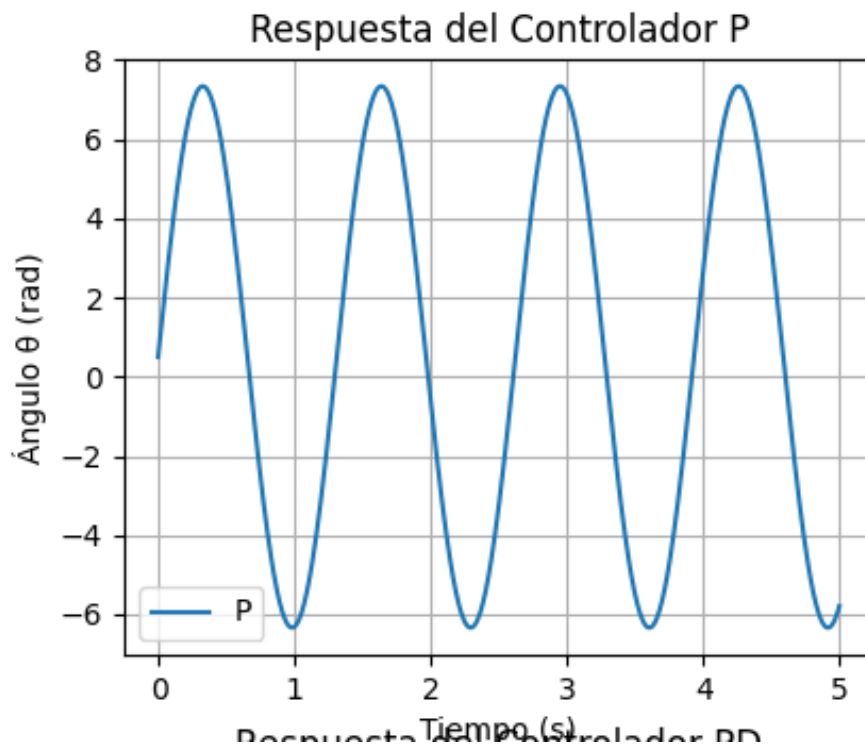
Con el fin de evaluar el desempeño de cada controlador, se ajustaron manualmente los parámetros K_p , K_i y K_d . Los casos serán los siguientes:

- **Controlador P:** $K_p > 0$, $K_i = 0$, $K_d = 0$

En la Figura 26 se observa que el sistema presenta oscilaciones sostenidas con amplitudes constantes, lo que indica una estabilidad marginal. Esto ocurre porque el término proporcional no proporciona amortiguación suficiente para contrarrestar las oscilaciones naturales del sistema.

Figura 26

Gráfica con controlador P

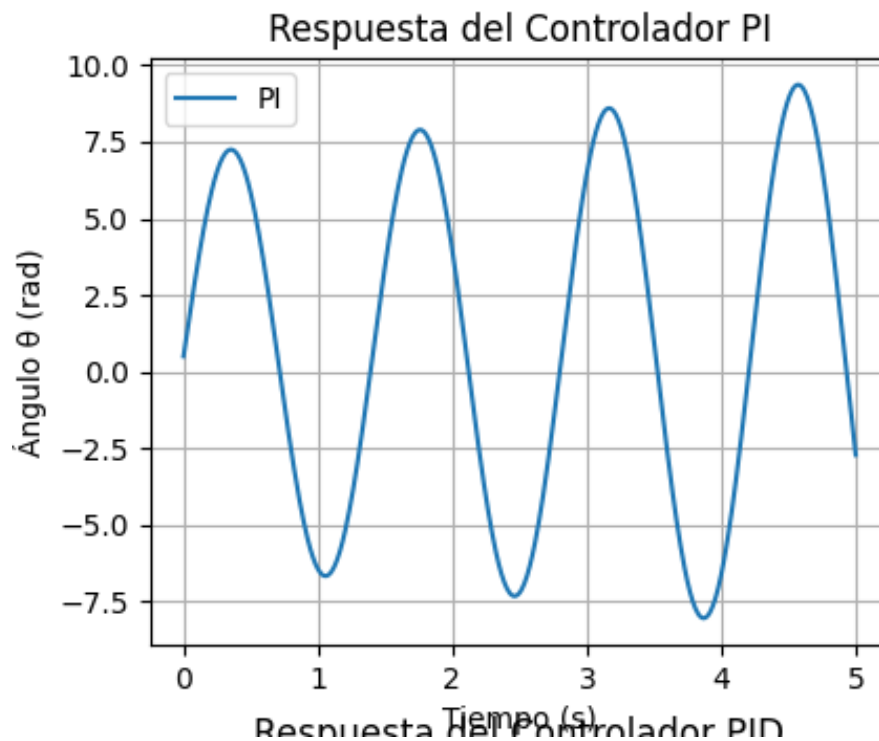


- **Controlador PI:** $K_p > 0$, $K_i > 0$, $K_d = 0$

En la Figura 27, se aprecia un comportamiento oscilatorio similar al del controlador P, con amplitudes que permanecen constantes a lo largo del tiempo. Este fenómeno sugiere que la adición del término integral en el controlador no ha sido suficiente para mitigar las oscilaciones ni para garantizar una respuesta estable. Esto sugiere que la influencia acumulativa del error, aunque ayuda en la corrección a largo plazo, no tiene un impacto significativo en la amortiguación.

Figura 27

Gráfica con controlador PI

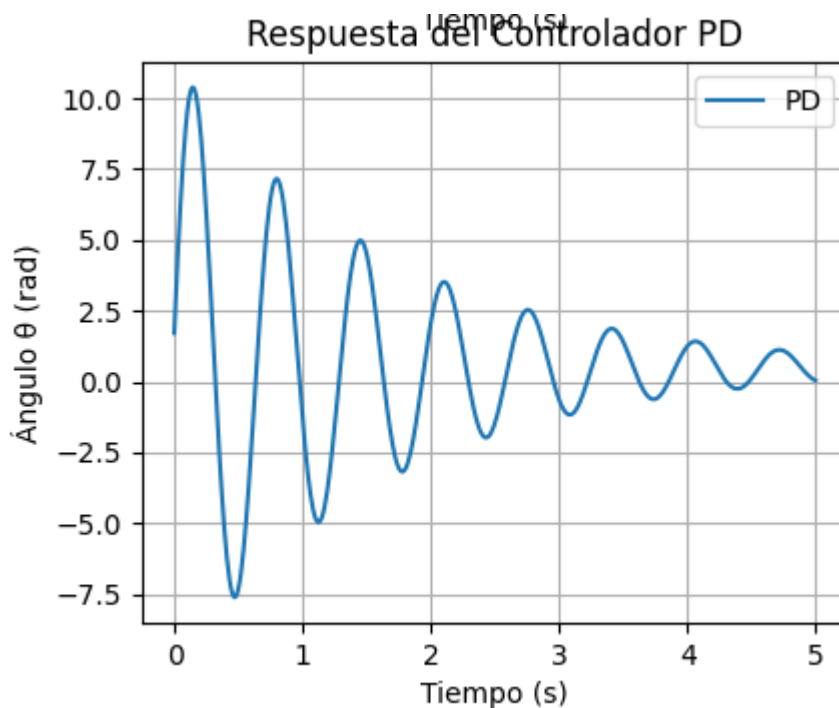


- **Controlador PD:** $K_p > 0$, $K_i = 0$, $K_d > 0$

En la Figura 28 se observa una mejora significativa respecto a los controladores P y PI previamente analizados. Aunque el sistema comienza con oscilaciones iniciales de alta amplitud, estas disminuyen progresivamente hasta casi desaparecer. Esto se debe al término derivativo, que actúa como un amortiguador al anticipar los cambios en el error, mejorando la estabilidad y el tiempo de respuesta.

Figura 28

Gráfica con controlador PD

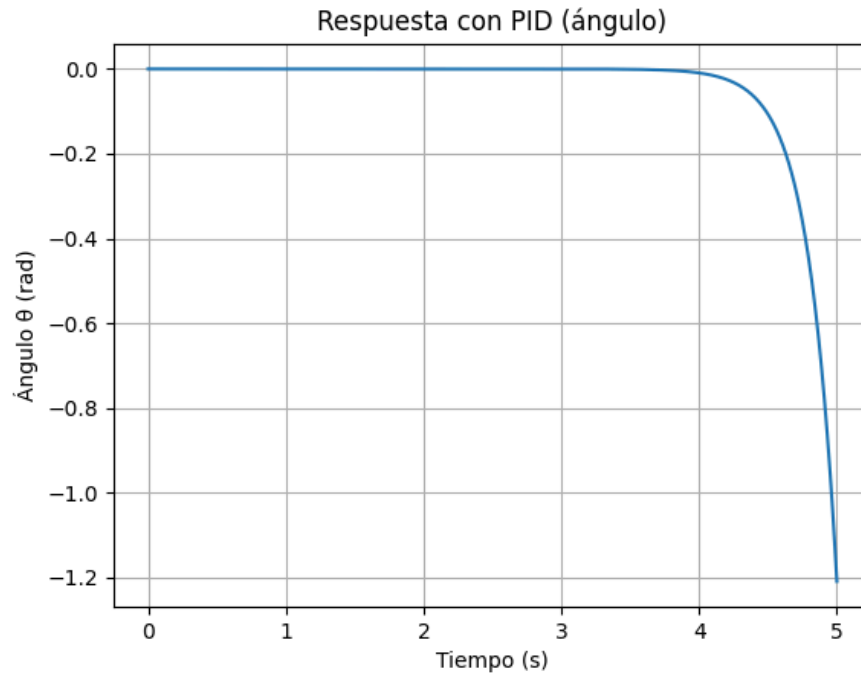


- **Controlador PD:** $K_p > 0$, $K_i > 0$, $K_d > 0$

El comportamiento del sistema bajo el controlador PID (ver Figura 29) muestra un desempeño significativamente diferente respecto a los controladores P, PI y PD. El ángulo θ permanece en 0 durante los primeros 4 segundos, indicando que el sistema mantiene una posición estable sin desviaciones durante este periodo. Sin embargo, una caída abrupta después del segundo 4 indica que el término derivativo puede estar sobreamortiguando el sistema.

Figura 29

Gráfica con controlador PID



- Parámetros resultantes:

Kp: 1×10^{-10}

Ki: 0.4×10^{-12}

Kd: 1.5×10^{-10}

En la Tabla 1 se muestra un resumen de los resultados encontrados para cada controlador aplicado.

Tabla 1

Comportamiento de los Controles Aplicados

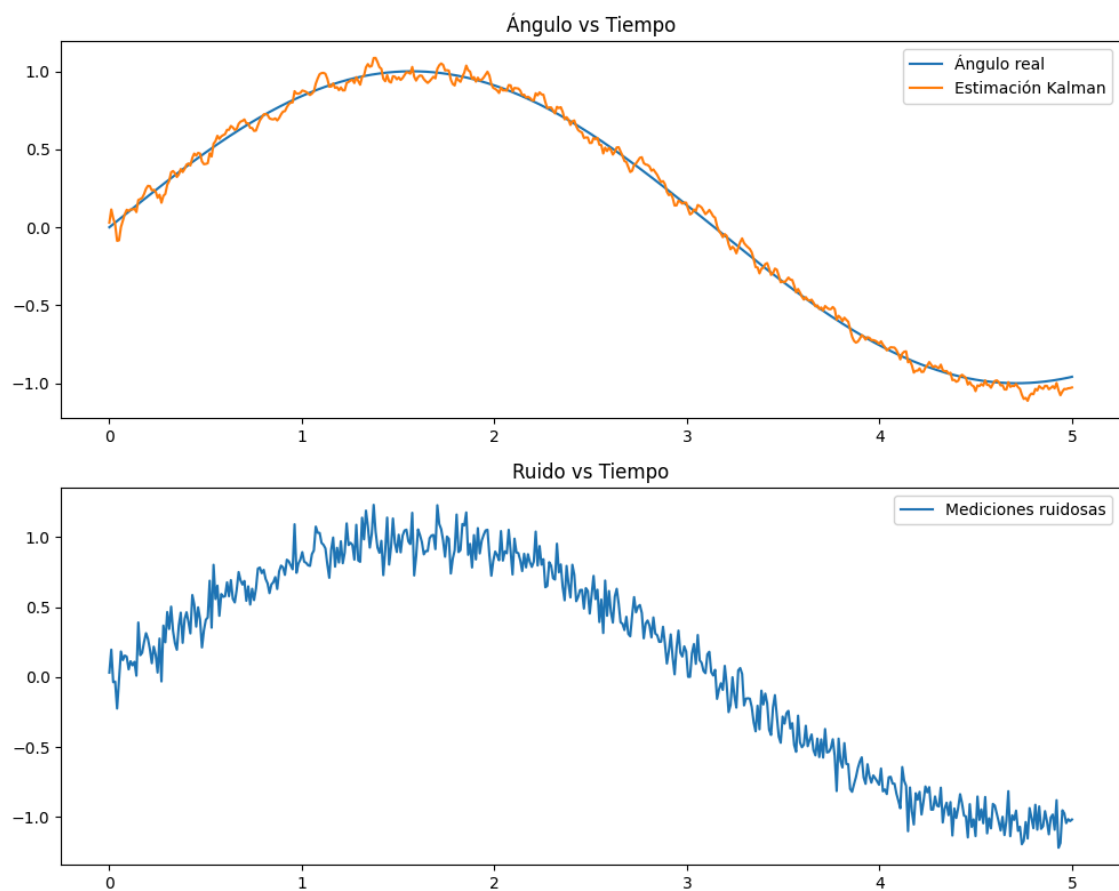
Controlador	Comportamiento
P	Presenta un comportamiento oscilatorio sostenido con amplitudes constantes, indicando que no logra amortiguar las oscilaciones ni estabilizar el sistema.
PI	También presenta un comportamiento oscilatorio con amplitudes

	constantes, similar al controlador P. Aunque incluye un término integral, no mejora significativamente la estabilidad ni reduce las oscilaciones.
PD	Muestra un comportamiento amortiguado con oscilaciones que disminuyen progresivamente hasta casi desaparecer. El sistema se estabiliza más rápidamente, logrando un desempeño más deseable que los controladores P y PI.
PID	Mantiene el ángulo estable en 0 radianes durante un tiempo prolongado (hasta 4 segundos). Sin embargo, experimenta una caída abrupta en el ángulo después del segundo 4, probablemente debido al dominio del término derivativo.

PID optimizado con filtro de Kalman

Figura 32

Gráfica de comparación de Ángulo vs Tiempo

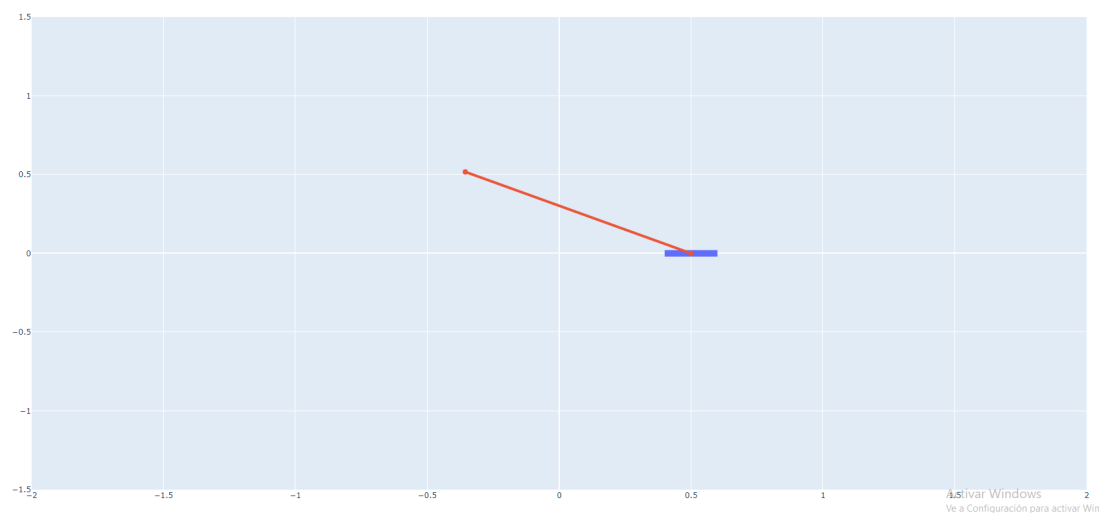


Se observa que la estimación sigue de manera precisa el ángulo real, reduciendo notablemente las variaciones ocasionadas por el ruido en las mediciones. Esto indica que el filtro de Kalman es efectivo para suavizar los datos, proporcionando una señal más limpia y confiable para el controlador PID.

Las mediciones ruidosas muestran un nivel significativo de ruido, especialmente visible en las oscilaciones de alta frecuencia. Sin embargo, al comparar estos datos con las estimaciones filtradas del gráfico superior, se evidencia cómo el filtro de Kalman elimina el ruido, dejando únicamente las variaciones relacionadas con el comportamiento real del sistema.

Figura 33

Simulación gráfica del PID Optimizado con filtro de Karman



La simulación indica que el péndulo invertido no logra estabilizarse completamente y muestra movimientos laterales significativos. Esto sugiere que el controlador PID, incluso optimizado con el filtro de Kalman, no está ajustado de manera adecuada para contrarrestar las desviaciones del péndulo.

Mejora con el controlador PID Optimizado

Se optimizó los parámetros del PID, y se simuló en el péndulo:

Figura 34

Gráfica con controlador PID optimizado

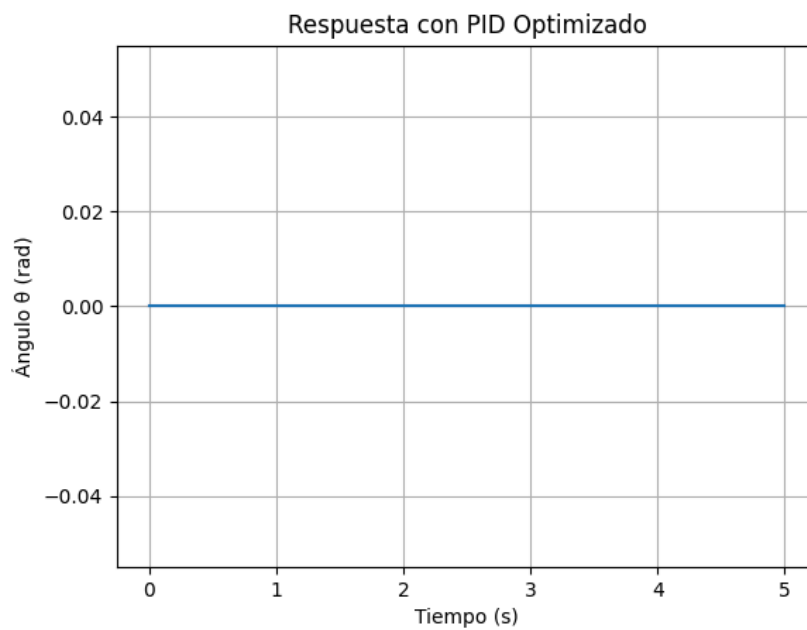
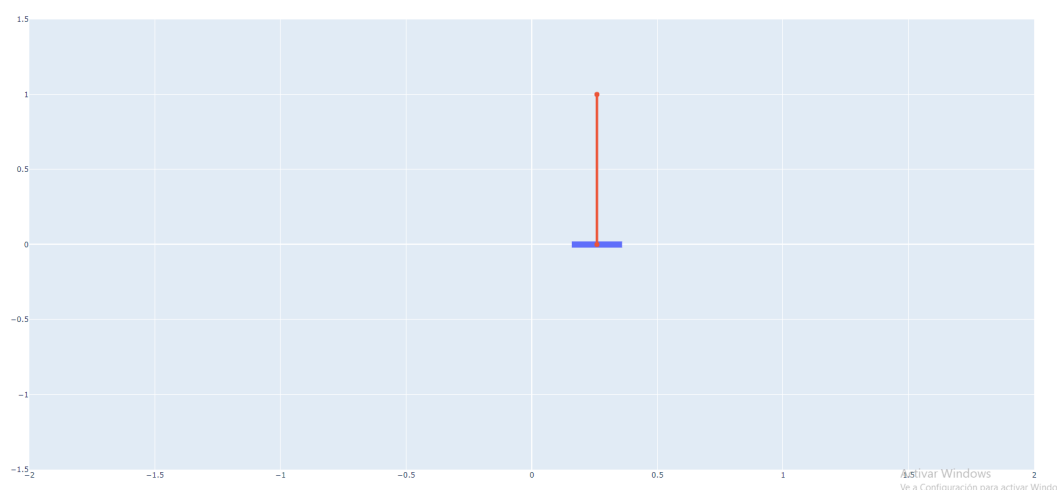


Figura 35

Simulación gráfica del PID Optimizado



A diferencia de los controladores anteriores, el controlador PID optimizado mantiene el ángulo θ en cero en todo momento, indicando que el sistema está perfectamente

estabilizado. No se observan oscilaciones ni errores acumulados, lo que confirma que el sistema ha alcanzado un equilibrio ideal.

- Parámetros resultantes:

$$K_p: 8.88 \times 10^{-16}$$

$$K_i: 0.0$$

$$K_d: 4.88 \times 10^{-16}$$

Estos valores indican que la optimización determinó que los términos proporcional e integral no tienen un impacto significativo en este caso, mientras que el término derivativo aporta un efecto mínimo.

Este resultado refleja la capacidad del controlador PID para garantizar la estabilidad del sistema en presencia de perturbaciones externas. La simulación demuestra que el péndulo no solo se estabiliza en condiciones estáticas, sino que también responde adecuadamente a dinámicas más complejas, como el movimiento del carrito. Este comportamiento destaca la robustez del diseño del controlador, incluso cuando se enfrenta a escenarios no triviales.

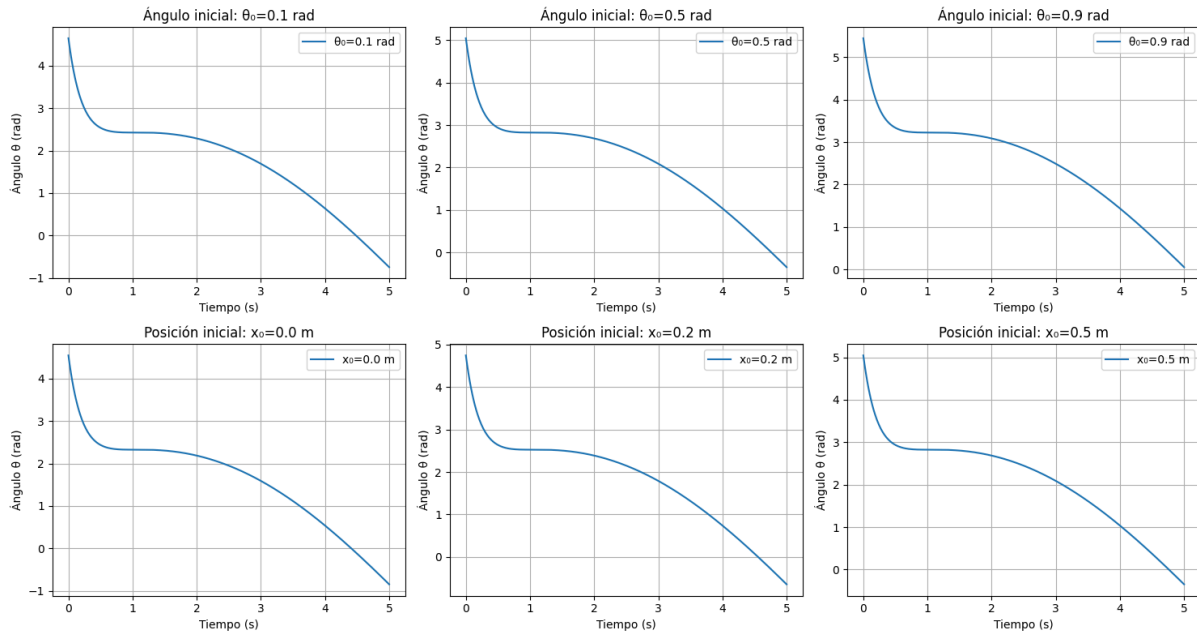
7.1. Preguntas

- a. ¿Qué tan sensible es el sistema con el controlador PID (manual u optimizado) ante variaciones en las condiciones iniciales, como un ángulo mayor del péndulo o una posición inicial desplazada del carro?**

Las gráficas de la Figura 36 muestran la sensibilidad del sistema controlado por el PID ante variaciones en las condiciones iniciales, como el ángulo inicial del péndulo y el desplazamiento inicial del carro. En general, se observa que para ángulos iniciales más pequeños ($\theta_0 = 0.1 \text{ rad}$) o desplazamientos iniciales pequeños ($x_0 = 0.0 \text{ m}$), el sistema converge más rápidamente hacia la posición de equilibrio. Sin embargo, a medida que el ángulo inicial o el desplazamiento inicial aumentan ($\theta_0 = 0.9 \text{ rad}$ o $x_0 = 0.5 \text{ m}$), el sistema requiere más tiempo para estabilizarse, mostrando mayor sensibilidad a estas condiciones.

Figura 36

Gráfica de la sensibilidad al ángulo inicial y el desplazamiento inicial

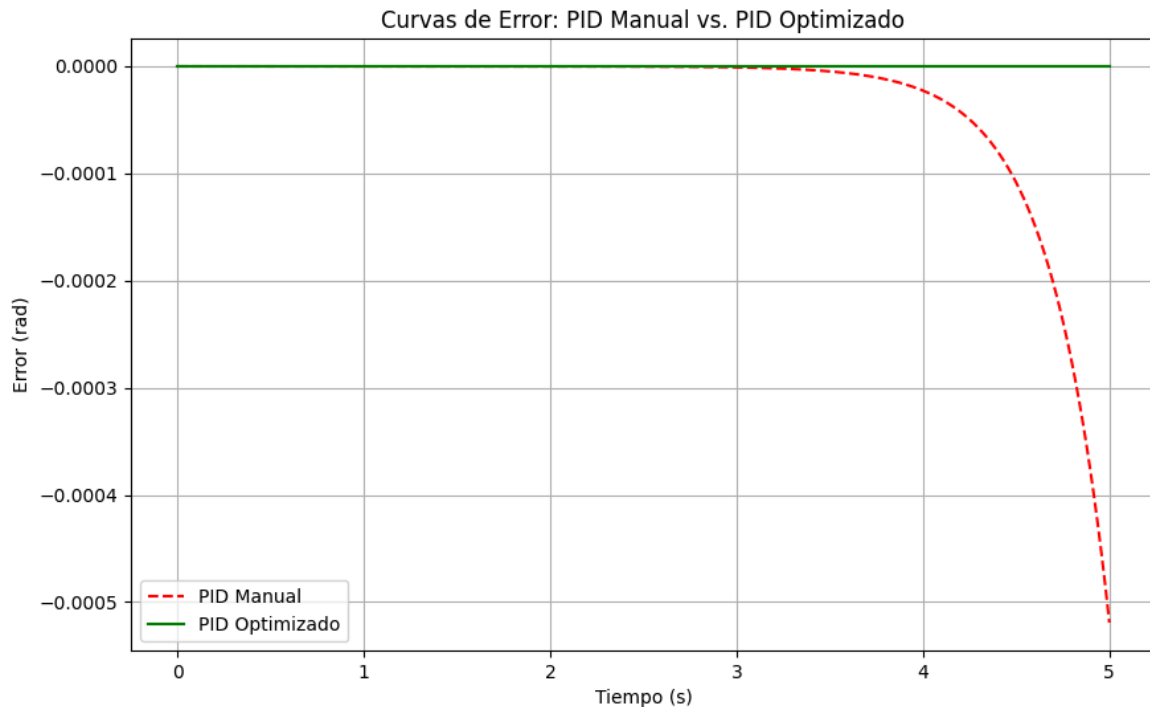


b. Después de la optimización, ¿qué cambios observas en la curva de error del sistema?

Después de la optimización, la curva de error del sistema muestra una mejora significativa (ver Figura 37). Con el PID manual, el error comienza a incrementarse alrededor del tercer segundo, lo que indica una pérdida de control o inestabilidad en el sistema. En cambio, con el PID optimizado, el error se mantiene cerca de cero durante toda la operación, reflejando un control más preciso y estable. Esto sugiere que la optimización del PID ajustó adecuadamente las ganancias para minimizar las oscilaciones y las desviaciones del sistema, logrando un desempeño significativamente mejor frente a perturbaciones o variaciones en las condiciones iniciales.

Figura 37

Gráfica de la curva de error



c. ¿Qué diferencias importantes se identifican en el comportamiento del sistema con el controlador optimizado respecto a los ajustes manuales?

El controlador ajustado manualmente genera una respuesta menos estable, ya que se observan cambios significativos en el estado del sistema con el tiempo. En contraste, el controlador optimizado logra una estabilidad completa, lo que demuestra una mejor sintonización de los parámetros y una eliminación efectiva de errores.

En cuanto a precisión el controlador manual presenta errores de seguimiento significativos, especialmente hacia el final del intervalo analizado. Esto sugiere que los valores ajustados manualmente no son capaces de garantizar el cumplimiento de las especificaciones deseadas. El controlador optimizado es notablemente más preciso, manteniendo el ángulo θ exactamente en el valor deseado durante todo este tiempo.

El uso de un controlador PID optimizado mejora significativamente la estabilidad, precisión y comportamiento dinámico del sistema comparación con un ajuste manual

Conclusiones

La efectividad del control de un sistema de péndulo invertido depende directamente del ajuste preciso de los parámetros K_p , K_i y K_d . Cada parámetro contribuye de forma única a la dinámica del sistema: K_p influye en la rapidez y agresividad de la respuesta, K_i garantiza la eliminación del error en estado estacionario, y K_d proporciona amortiguamiento crítico para estabilizar las oscilaciones. Un balance adecuado entre ellos es esencial para alcanzar una estabilidad óptima.

Aún así, si bien el término derivativo (K_d) es crucial para reducir oscilaciones, su sensibilidad al ruido puede ser problemática en aplicaciones reales. Esto subraya la necesidad de utilizar técnicas complementarias, como filtros, para garantizar un rendimiento confiable en entornos prácticos.

Por parte de los controladores P, PI y PD, estos presentan fortalezas específicas, pero cada uno tiene limitaciones notables cuando se utilizan de forma independiente. Por ejemplo, el controlador P no puede eliminar el error en estado estacionario, el PI tiende a generar oscilaciones si no se ajusta correctamente, y el PD es sensible al ruido de la señal.

Para el sistema estudiado, el controlador PID resultó ser el más robusto y versátil. El equilibrio obtenido de sus componentes individuales permite optimizar la precisión, la estabilidad y la respuesta transitoria del sistema, convirtiéndolo en la solución ideal para el control de un sistema de péndulo invertido sobre un carro.

La sensibilidad del sistema ante variaciones en las condiciones iniciales (como ángulos y desplazamientos grandes) evidenció que, aunque el controlador PID optimizado es eficaz para escenarios nominales, puede requerir ajustes adicionales para entornos más exigentes. Esto sugiere que los sistemas de control deben considerar no solo la estabilidad en condiciones normales, sino también la capacidad de respuesta ante perturbaciones y escenarios extremos.

Referencias

- Ali, M., & Mandal, S. (2022). Preparation of papers for IFAC conferences & symposia: Kalman filter based control of inverted pendulum system. *IFAC-PapersOnLine*, 55(1), 58-63. <https://doi.org/10.1016/j.ifacol.2022.04.010>
- Askari, M., Mohamed, H. A. F., Moghavvemi, M., & Yang, S. S. (2010). Model predictive control of an inverted pendulum. In IEEE Techpos 2009 - 2009 IEEE Conference on Innovative Technologies. IEEE.
https://www.researchgate.net/publication/224113918_Model_predictive_control_of_a_n_inverted_pendulum
- Kim, M., Park, J., Park, D.-I., Park, C., & Cheong, J. (2023). Dynamic inversion-based real-time trajectory planning method for wheeled inverted pendulum using asymptotic expansion technique. *IEEE Access*, 11, 94805-94821.
<https://doi.org/10.1109/ACCESS.2023.3311025>
- Nisar, K. S., Farman, M., Jamil, K., Jamil, S., & Hincal, E. (2025). Fractional-order PID feedback synthesis controller including some external influences on insulin and glucose monitoring. *Alexandria Engineering Journal*, 113, 60–73.
<https://doi.org/10.1016/j.aej.2024.11.017>