UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

Escuela profesional de Ingeniería de Software

Curso*: Automatización y Control de Software*

Docente*:*

Grupo 6

INTEGRANTES:

Aldair Jhostin Solis Flores 20200293

Balandra Camacho, Ivan 20200248

Calixto Goñe, Fabrizio Alonso 20200252

Wong Gómez, Carlos Augusto 14160018

Montes Perez, Josue Justi 20200311

2024-2

CONTROL PÉNDULO INVERTIDO

Considere el sistema del péndulo invertido de la Figura 1, el cual representa un diagrama de cuerpo libre del sistema. Asumimos que la varilla del péndulo no tiene masa y que la bisagra a la que está fijado el péndulo no tiene fricción. La masa del péndulo está concentrada en el centro de gravedad del péndulo que está ubicado en el centro de la bola del péndulo. La masa del carro se representa como *M* y la masa del péndulo se representa como *m*. La ley de control(t) se aplica en la fuerza necesaria para actuar a lo largo de la dirección x del carro. La longitud de la varilla se representa como *l*. El ángulo con el que el péndulo está inclinado se representa como θ .

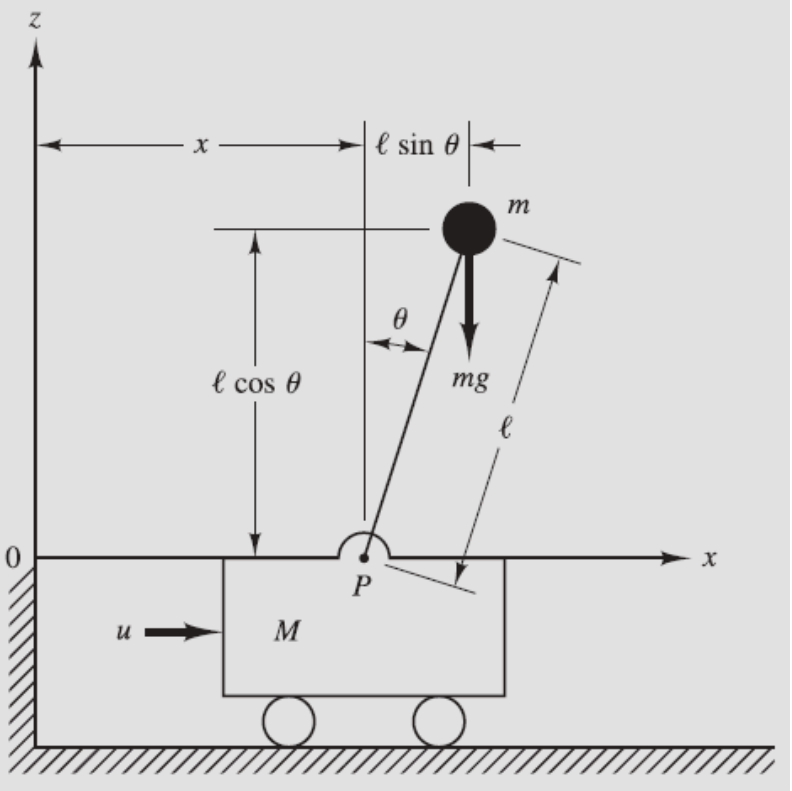


Figura 1. Diagrama del péndulo.

El modelo matemático del sistema es el siguiente:

$$(M+m)\ddot{x}+ml\ddot{\theta}=u \qquad (1)$$

$$ml^2\ddot{\theta}+ml\ddot{x}=mgl\theta \qquad (2)$$

Las ecuaciones (1) y (2) pueden ser modificadas a:

$$Ml\ddot{\theta}=(M+m)g\theta-u \qquad (3)$$

$$M\ddot{x} = u - mg\theta \quad (4)$$

1. ***Deriva la función de transferencia (FT) del sistema a partir de las ecuaciones (3) y (4). Calcula las raíces del sistema para determinar su estabilidad inicial sin control.***

   Estas ecuaciones 3 y 4 constituyen el modelo matemático del sistema (péndulo y carro respectivamente). Para poder hallar la relación entre la salida y la entrada del sistema debemos trabajar con la transformada de Laplace que nos permite simplificar el análisis matemático y nos ayudará en el diseño de los sistemas de control.

   a) Para hallar la función de transferencia para el péndulo, a la ecuación:

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

   Vamos a aplicar la **transformada de Laplace**. Recordemos las propiedades de esta herramienta:

- La transformada de Laplace de $\ddot{\theta}$ es $s^2\Theta(s) - s\theta(0) - \dot{\theta}(0)$
- La transformada de Laplace de θ es $\Theta(s)$.
- La transformada de Laplace de u es $U(s)$.

   Entonces aplicando Laplace:

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

   Sustituyendo:

$$Ml\left(s^2\Theta(s) - s\theta(0) - \dot{\theta}(0)\right) = (M+m)g\Theta(s) - U(s)$$

   Expandiendo los términos:

$$Mls^2\Theta(s) - Mls\theta(0) - Ml\dot{\theta}(0) = (M+m)g\Theta(s) - U(s)$$

   Reorganizando:

$$Mls^2\Theta(s) - (M+m)g\Theta(s) - Mls\theta(0) - Ml\dot{\theta}(0) = -U(s)$$

En este punto necesitamos asumir que las condiciones iniciales son cero ($\dot{\theta}(0)=0$ y $\theta(0)=0$), ya que esto es común en sistemas lineales cuando derivamos funciones de transferencia.

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

Factorizando:

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

La función de transferencia para el péndulo se calcula a partir de (a): $\dfrac{\Theta(s)}{-U(s)}$

Función de transferencia del péndulo: $$\frac{\Theta(s)}{-U(s)}=\frac{1}{\left(Mls^2-(M+m)g\right)} \quad \ldots \quad (5)$$

b) Para hallar la función de transferencia del carro:

Recodemos que:

- La transformada de Laplace de $\ddot{x}$ es $s^2\mathrm{X}(s)-sx(0)-\dot{x}(0)$.
- La transformada de Laplace de θ es $\Theta(s)$.
- La transformada de Laplace de u es $U(s)$.

La transformada de Laplace de la ecuación (4) es:

$$L\left(M\ddot{x}\right)=L(u)-L(mg\theta)$$

$$M\left(s^2\mathrm{X}(s)-sx(0)-\dot{x}(0)\right)=U(s)-mg\Theta(s)$$

Para expresar la ecuación como una **función de transferencia**, debemos considerar que las condiciones iniciales son cero ($x(0)=0$ y $\dot{x}(0)=0$). Esto simplifica la ecuación considerablemente.

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

Sustituyendo $\Theta(s)$ de la ecuación (5):

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

Simplificando:

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

Factorizando $U(s)$:

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

Finalmente, la función de transferencia del carro que relaciona $U(s)$ con $\mathrm{X}(s)$ es:

$$G_x(s) = \frac{\mathrm{X}(s)}{U(s)} = \frac{1}{Ms^2\left(1 + \frac{mg}{Mls^2 - (M+m)g}\right)} \quad \ldots \quad (6)$$

Reduciendo:

$$G_x(s) = \frac{\mathrm{X}(s)}{U(s)} = \frac{Mls^2 - (M+m)g}{M^2ls^4 - M^2gs^2} \quad \ldots \quad (7)$$

c) Cálculo de las raíces del sistema, considerando la FT péndulo:

Cambiando la forma del denominador de

$$Mls^2 - (M+m)g = Ml\left(s^2 - \frac{(M+m)g}{Ml}\right) = Ml\left(s - \sqrt{\frac{(M+m)g}{Ml}}\right)\left(s + \sqrt{\frac{(M+m)g}{Ml}}\right)$$

Entonces reemplazando:

$$\frac{\Theta(s)}{-U(s)} = \frac{1}{Ml\left(s - \sqrt{\frac{(M+m)g}{Ml}}\right)\left(s + \sqrt{\frac{(M+m)g}{Ml}}\right)}$$

Del denominador de la función de transferencia, las raíces del sistema son:

$$s = \sqrt{\frac{(M+m)g}{Ml}} \qquad \text{y} \qquad s = -\sqrt{\frac{(M+m)g}{Ml}}$$

La estabilidad de un sistema en lazo abierto o sin control depende de la ubicación de las raíces o polos de su función de transferencia en el plano complejo. Si una función de transferencia tiene una raíz negativa y otra positiva, esta situación nos indica que el sistema es ***inestable en lazo abierto***.

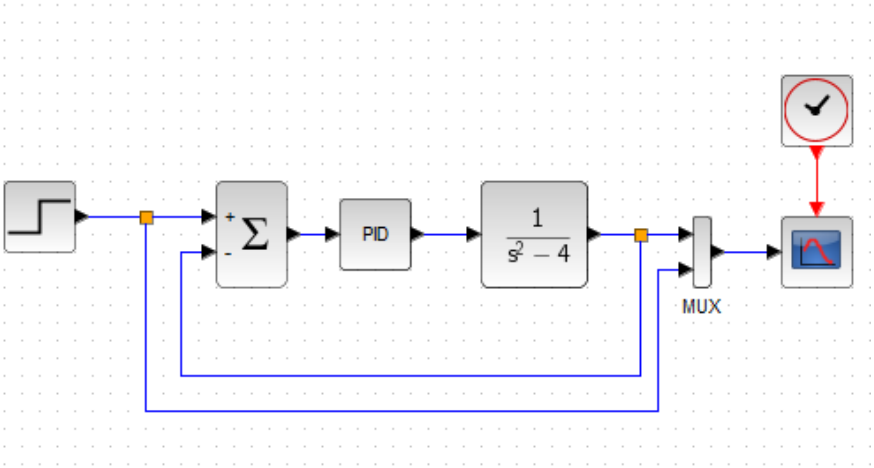
2. ***Diseña un controlador PID que permita mantener el péndulo en posición vertical. Simula el comportamiento del sistema con cada tipo de controlador (P, PI, PD y PID). ¿Cómo influyen los parámetros individuales Kp, Ki y Kd en la respuesta del sistema? ¿Qué diferencias observas en el comportamiento del sistema al usar controladores P, PI, PD y PID?***

Aplicamos un control PID cuya función de transferencia tiene esta forma:

$$k_p + \frac{k_i}{s} + k_d s$$

Donde los parámetros del controlador PID Kp, Ki y Kd determinan el peso que cada componente tiene en la respuesta final. Por lo tanto, lograr un buen desempeño del controlador requiere un ajuste adecuado de estos tres parámetros.

Aplicamos el control PID a la planta en un lazo cerrado aplicando la función step a su entrada para ver su comportamiento como se muestra en la siguiente figura:



El siguiente código en Python simula el comportamiento de un controlador PID utilizando la función de transferencia del péndulo. Se le asigna valores a M, m, l y g. Valores de Kp, Ki, Kd de manera manual para los diferentes controladores P, PI, PD, PID.

```
import numpy as np

import matplotlib.pyplot as plt

import control as ctrl
```

```
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg


# Parámetros del sistema
M = 2  # Masa del carro (kg)
m = 0.5  # Masa del péndulo (kg)
l = 1  # Longitud del péndulo (m)
g = 9.81  # Aceleración de la gravedad (m/s²)


# Sistema G(s)
numerador_G = [1]
denominador_G = [(M * l), 0, -(M + m) * g]
G = ctrl.TransferFunction(numerador_G, denominador_G)


# Función para actualizar el gráfico
def update_plot():
    # Obtener valores de las barras deslizantes
    Kp = slider_kp.get()
    Ki = slider_ki.get()
    Kd = slider_kd.get()


    # Definir los controladores
    C_P = ctrl.TransferFunction([Kp], [1])
    C_PI = ctrl.TransferFunction([Kp, Ki], [1, 0])
    C_PD = ctrl.TransferFunction([Kd, Kp], [1])
    C_PID = ctrl.TransferFunction([Kd, Kp, Ki], [1, 0])


    # Lazo cerrado
```

```
    T_P = ctrl.feedback(G * C_P)
    T_PI = ctrl.feedback(G * C_PI)
    T_PD = ctrl.feedback(G * C_PD)
    T_PID = ctrl.feedback(G * C_PID)

    # Tiempo de simulación
    t = np.linspace(0, 5, 1000)

    # Respuestas al escalón
    time_P, yout_P = ctrl.step_response(T_P, t)
    time_PI, yout_PI = ctrl.step_response(T_PI, t)
    time_PD, yout_PD = ctrl.step_response(T_PD, t)
    time_PID, yout_PID = ctrl.step_response(T_PID, t)

    # Limpiar el gráfico actual
    ax.clear()

    # Graficar las respuestas
    ax.plot(time_P, yout_P, label="Controlador P")
    ax.plot(time_PI, yout_PI, label="Controlador PI")
    ax.plot(time_PD, yout_PD, label="Controlador PD")
    ax.plot(time_PID, yout_PID, label="Controlador PID")

    # Personalizar gráfico
    ax.axhline(y=1, color='r', linestyle='--', label="Referencia (setpoint)")
    ax.set_title("Respuestas al Escalón de Controladores P, PI, PD y PID")
    ax.set_xlabel("Tiempo (s)")
    ax.set_ylabel("Salida")
```

```
    ax.legend()
    ax.grid(True)
    ax.set_ylim(-5, 5)

    # Dibujar el gráfico actualizado
    canvas.draw()

# Crear la ventana principal
root = tk.Tk()
root.title("Simulador de Controladores PID")

# Crear un marco para los sliders
frame_controls = tk.Frame(root)
frame_controls.pack(side=tk.LEFT, padx=10, pady=10)

# Crear sliders para Kp, Ki, Kd
slider_kp = tk.Scale(frame_controls, from_=0, to_=500, resolution=1,
orient=tk.HORIZONTAL, label="Kp")
slider_kp.set(120)
slider_kp.pack()

slider_ki = tk.Scale(frame_controls, from_=0, to_=500, resolution=1,
orient=tk.HORIZONTAL, label="Ki")
slider_ki.set(160)
slider_ki.pack()

slider_kd = tk.Scale(frame_controls, from_=0, to_=500, resolution=1,
orient=tk.HORIZONTAL, label="Kd")
slider_kd.set(120)
```

```
slider_kd.pack()


# Botón para actualizar el gráfico
btn_update = tk.Button(frame_controls, text="Actualizar Gráfico", command=update_plot)
btn_update.pack(pady=10)


# Crear el gráfico
fig, ax = plt.subplots(figsize=(8, 6))
canvas = FigureCanvasTkAgg(fig, master=root)
canvas_widget = canvas.get_tk_widget()
canvas_widget.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)


# Mostrar la ventana principal
update_plot()  # Mostrar el gráfico inicial
root.mainloop()
```

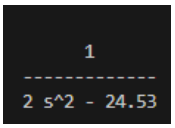


Fig. 1. Impresión por el código python de la función de transferencia.

Resultado de la ejecución del código:

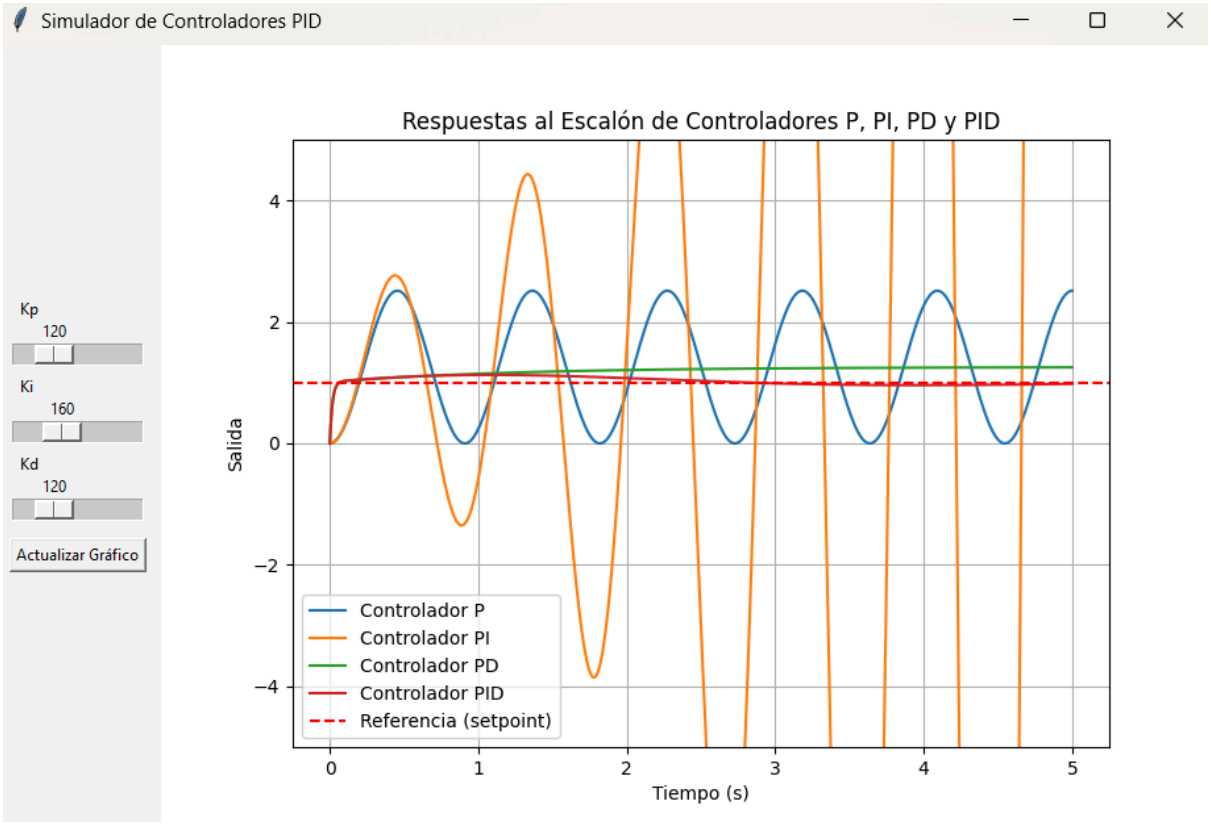


Fig. 2. Gráfica de la respuesta del sistema a la función STEP con los controladores P, PI, PD y PID.

Esta gráfica muestra las respuestas de diferentes tipos de controladores (P, PI, PD y PID) frente a una referencia de escalón (setpoint) aplicada a una planta. A continuación, analizamos los comportamientos observados:

a) Controlador P (línea azul)

- **Características:**
  - Oscila constantemente sin estabilizarse.
  - Presenta un sobre impulso significativo.
  - No elimina el error en estado estacionario.
- **Conclusión:** El controlador proporcional puro no es suficiente para estabilizar este sistema

b) Controlador PI (línea naranja)

- **Características:**
  - La respuesta muestra oscilaciones severas y amplificación inestable.
  - Parece que la acumulación del término integral genera una inestabilidad creciente.

- **Conclusión:** En este caso, el controlador PI mal ajustado produce inestabilidad, probablemente debido a un valor elevado de *Ki*.

c) Controlador PD (línea verde)

- **Características:**
    - La respuesta es bien amortiguada y se acerca rápidamente a la referencia (setpoint).
    - Presenta un sobre impulso reducido y una buena estabilidad.
- **Conclusión:** El término derivativo *Kd* ayuda a amortiguar las oscilaciones, logrando una respuesta más estable y precisa.

d) Controlador PID (línea roja)

- **Características:**
    - La respuesta se estabiliza alrededor de la referencia sin oscilaciones importantes.
    - Elimina el error en estado estacionario.
    - Tiene un sobre impulso menor en comparación con el controlador PI y una mejor estabilidad que el P puro.
- **Conclusión:** Este controlador combina las ventajas de *Kp*, *Ki* y *Kd*, logrando una respuesta balanceada entre rapidez, precisión y estabilidad.

3. **Analiza cómo cambia el desempeño del sistema al integrar los algoritmos genéticos para optimizar los parámetros Kp, Ki, Kd, del controlador PID. Compara el desempeño del PID optimizado frente a uno ajustado manualmente, destacando sus ventajas y limitaciones.**

   Código Python de algoritmo genético donde introduciremos como función de transferencia, la perteneciente a la planta, para que de esta forma nos dé como resultado los valores de Kp, Ki, Kd ajustados que nos permitan tener un mejor desempeño del controlador. Además, en una gráfica nos mostrará el PID ajustado con el algoritmo genético, otra gráfica con el PID ajustado manualmente de la pregunta 2, y el nivel de referencia para hacer una comparación.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lti, step
from deap import base, creator, tools, algorithms

# --- Definir la planta como una función de transferencia ---
from scipy.signal import TransferFunction

# Definir las ganancias segun los valores de la pregunta 2, valores ajustados
manualmente
Kp_m = 120  # Ganancia proporcional
Ki_m = 160 # Ganancia integral
Kd_m = 120 # Ganancia derivativa

# Parámetros del sistema
M = 2  # Masa del carro (kg)
m = 0.5  # Masa del péndulo (kg)
l = 1  # Longitud del péndulo (m)
g = 9.81 # Aceleración de la gravedad (m/s²)

#definir numerador y denominador de la planta
num = [1]
den = [M*l, 0, -(M+m)*g]
plant = TransferFunction(num, den)

# --- Simulación del sistema cerrado con un PID ---
def simulate_pid(kp, ki, kd, plant, time):
    """Simula la respuesta de la planta controlada por un PID."""
    # PID Transfer Function: G_pid(s) = kp + ki/s + kd*s
    num_pid = [kd, kp, ki]  # Numerador del PID
    den_pid = [1, 0]  # Denominador del PID
    pid = TransferFunction(num_pid, den_pid)

    # Sistema cerrado: G_cl(s) = (G_pid(s) * Plant(s)) / (1 + G_pid(s) * Plant(s))
```

```
    system_closed = lti(np.convolve(pid.num, plant.num),
np.polyadd(np.convolve(pid.num, plant.num), np.convolve(pid.den, plant.den)))

    # Simulación del escalón
    t, y = step(system_closed, T=time)
    return t, y

# --- Definir la función objetivo ---
def fitness(individual):
    """Función objetivo para el algoritmo genético."""
    kp, ki, kd = individual
    time = np.linspace(0, 10, 1000)  # Tiempo de simulación
    t, y = simulate_pid(kp, ki, kd, plant, time)

    # Métrica: Minimizar el error cuadrático medio (MSE) y el overshoot
    mse = np.mean((y - 1)**2)  # Error respecto al escalón unitario
    overshoot = max(y) - 1 if max(y) > 1 else 0  # Penalización por sobreimpulso

    # Penalización si el sistema es inestable (explosión de la salida)
    if np.isnan(y).any() or max(y) > 10:
        return 1e6,  # Valor muy alto para descartar

    return mse + overshoot,  # Fitness a minimizar

# --- Configuración del algoritmo genético ---
# Crear un espacio de soluciones para Kp, Ki y Kd
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()
toolbox.register("attr_float", np.random.uniform, 0, 10)  # Rango de búsqueda para
Kp, Ki, Kd
toolbox.register("individual", tools.initRepeat, creator.Individual,
toolbox.attr_float, n=3)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("evaluate", fitness)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)

# --- Ejecución del algoritmo genético ---
def main():
    np.random.seed(42)  # Para reproducibilidad
    pop = toolbox.population(n=150)  # Tamaño de la población
    hof = tools.HallOfFame(1)  # Mejor individuo encontrado
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("min", np.min)
    stats.register("avg", np.mean)
```

```
    pop, log = algorithms.eaSimple(pop, toolbox, cxpb=1, mutpb=0.2, ngen=50,
stats=stats, halloffame=hof, verbose=True)

    # Mejor conjunto de parámetros encontrados
    best = hof[0]
    print(f"Mejor individuo (Kp, Ki, Kd): {best}")

    # Simulación final con los parámetros encontrados
    time = np.linspace(0, 10, 1000)
    t, y = simulate_pid(best[0], best[1], best[2], plant, time)
    t_m, y_m = simulate_pid(Kp_m, Ki_m, Kd_m, plant, time)
    # Graficar resultado
    plt.figure()
    plt.plot(t, y, label="Gráfica PID ajustado con algoritmos genéticos")
    plt.plot(t_m, y_m, label="Gráfica PID ajustado con ajuste manual")
    plt.axhline(1, color='r', linestyle='--', label="Referencia (Setpoint)")
    plt.title("Respuesta del sistema con PID optimizado")
    plt.xlabel("Tiempo (s)")
    plt.ylabel("Salida")
    plt.legend()
    plt.grid()
    plt.show()

if __name__ == "__main__":
    main()
```

```
47      150     0.0467098       0.0644214
48      150     0.043167        0.0606338
49      150     0.0370664       0.055611
50      150     0.0336589       0.0497022
Mejor individuo (Kp, Ki, Kd): [523.4020625749363, 195.60421500474638, 654.2407482321987]
```

Fig. 3. Resultado del algoritmo genético.

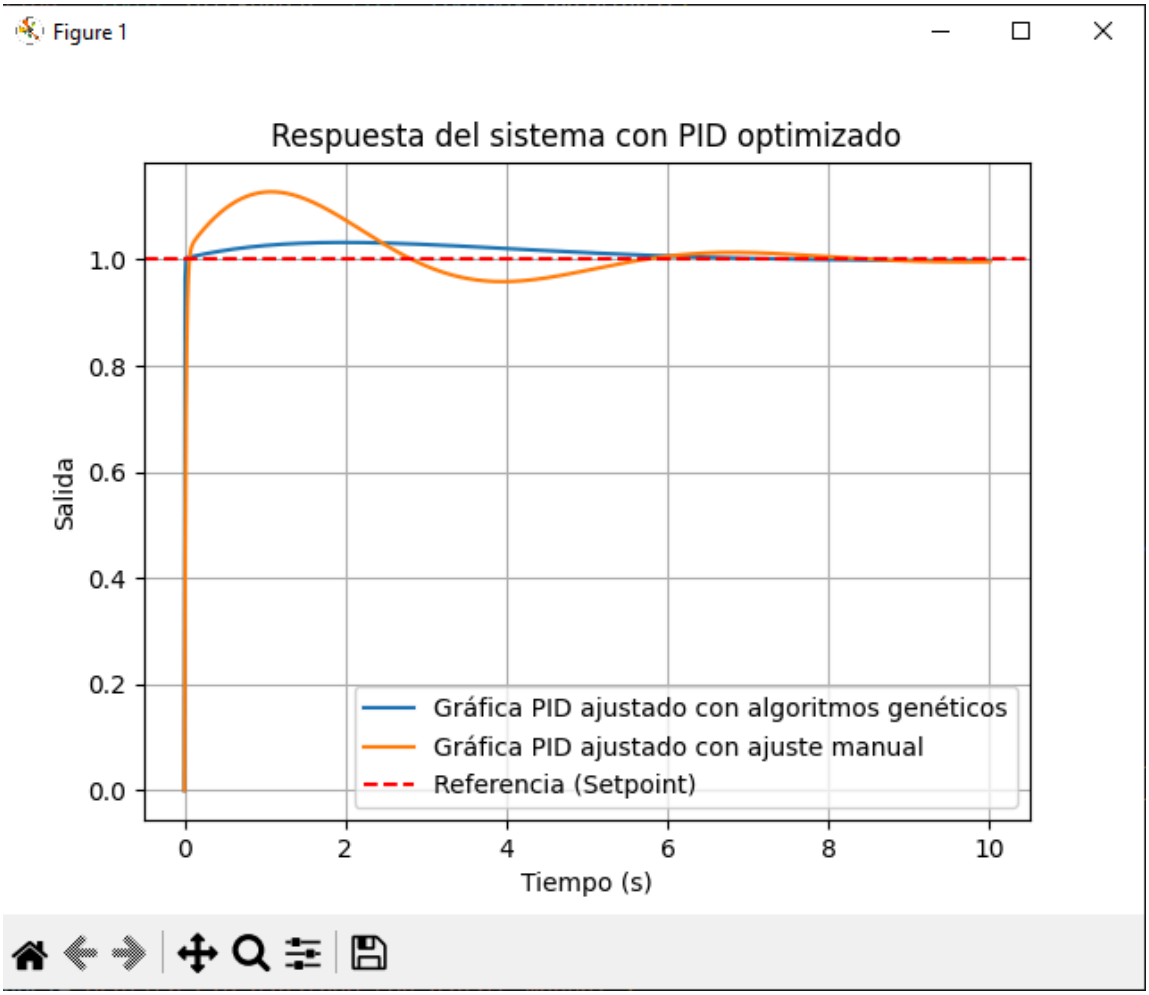


Fig. 4. PID ajustado manualmente de la pregunta 2 y con algoritmos genéticos.

Un PID ajustado manualmente se refiere a un contrlador PID en el que los parámetros *Kp* (proporcional), *Ki* (integral), y *Kd* (derivativo) son seleccionados y ajustados manualmente por un ingeniero o técnico con experiencia en planta.

En cambio, el PID optimizado mediante algoritmos genéticos puede superar al PID ajustado manualmente en términos de error acumulado y estabilidad del sistema, ya que el manual para su ajuste se basa en un proceso iterativo, donde se requiere experiencia que lo hace un proceso más largo y no adecuado para procesos más complejos.

En las limitaciones que encontramos para los algoritmos genéticos tenemos que requiere un alto costo computacional ya que requiere muchas simulaciones y puede tener un impacto en procesos controlados en tiempo real. Además, si el modelo no está bien representado los resultados pueden ser ineficaces.

Opciones para mejorar el comportamiento del PID ajustado con el algoritmo genético:

- El algoritmo genético puede requerir más exploración para encontrar soluciones óptimas. Aumentar el tamaño de la población y el número de generaciones ayuda al algoritmo a encontrar mejores soluciones.
- Aumenta ligeramente la probabilidad de cruce para combinar más características de individuos exitosos.
- Reduce la probabilidad de mutación si el algoritmo converge de forma inestable.
- Introduce restricciones dinámicas para forzar al sistema a cumplir con condiciones específicas, como un límite máximo de sobre impulso permitido:

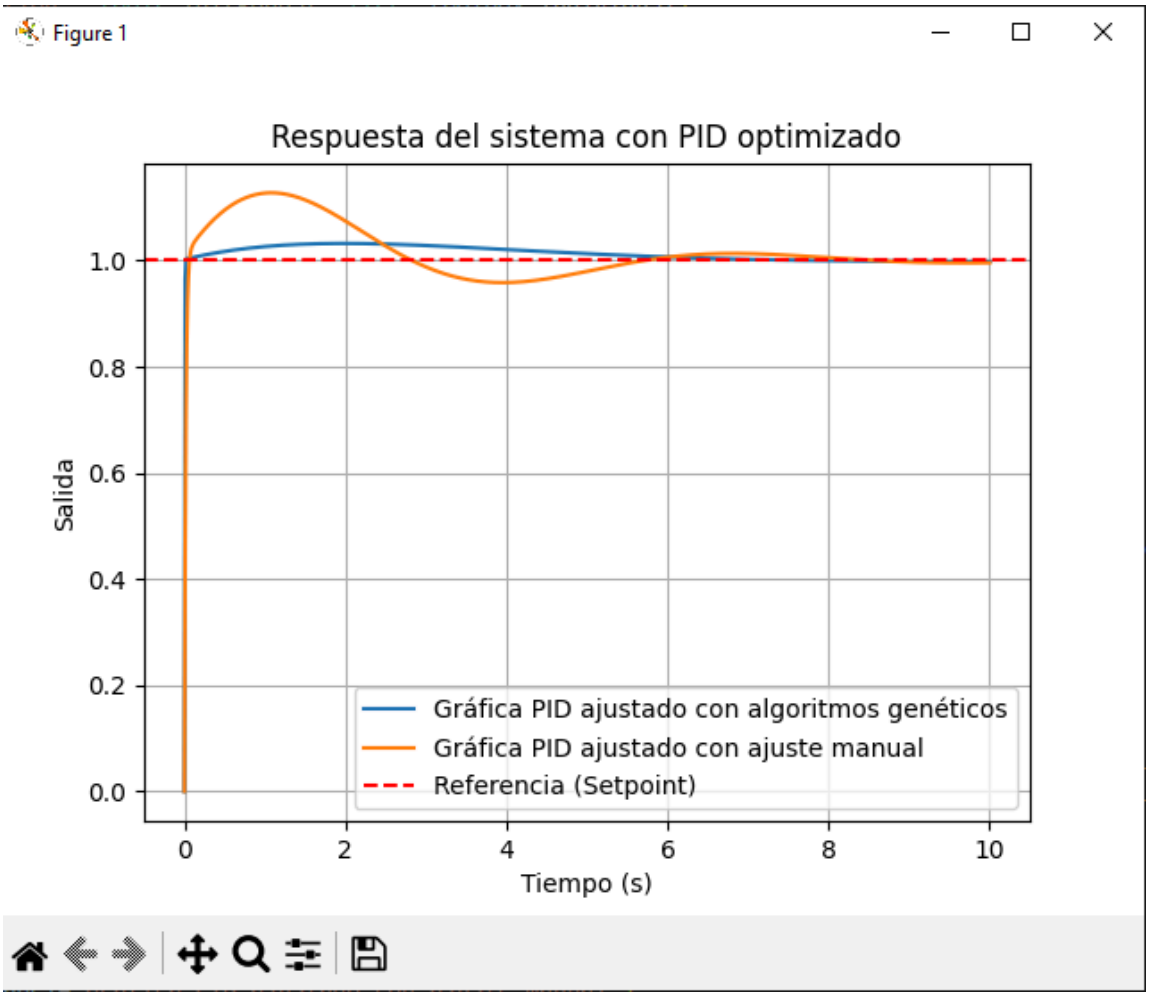


Fig. 5. PID ajustado con algoritmos genéticos con un mayor valor de probabilidad de cruce (2) y un menor valor de probabilidad de mutación (0.1).

4. Genera gráficos que permitan visualizar las dos leyes de control diseñados (PID ajustado manualmente y PID optimizado), evaluando el comportamiento del sistema en términos de tiempo de estabilización, sobre impulso y robustez ante perturbaciones.

Generamos el siguiente código en Python para comparar el desempeño del PID con ajuste manual y con un ajusta automático de la pregunta 3.

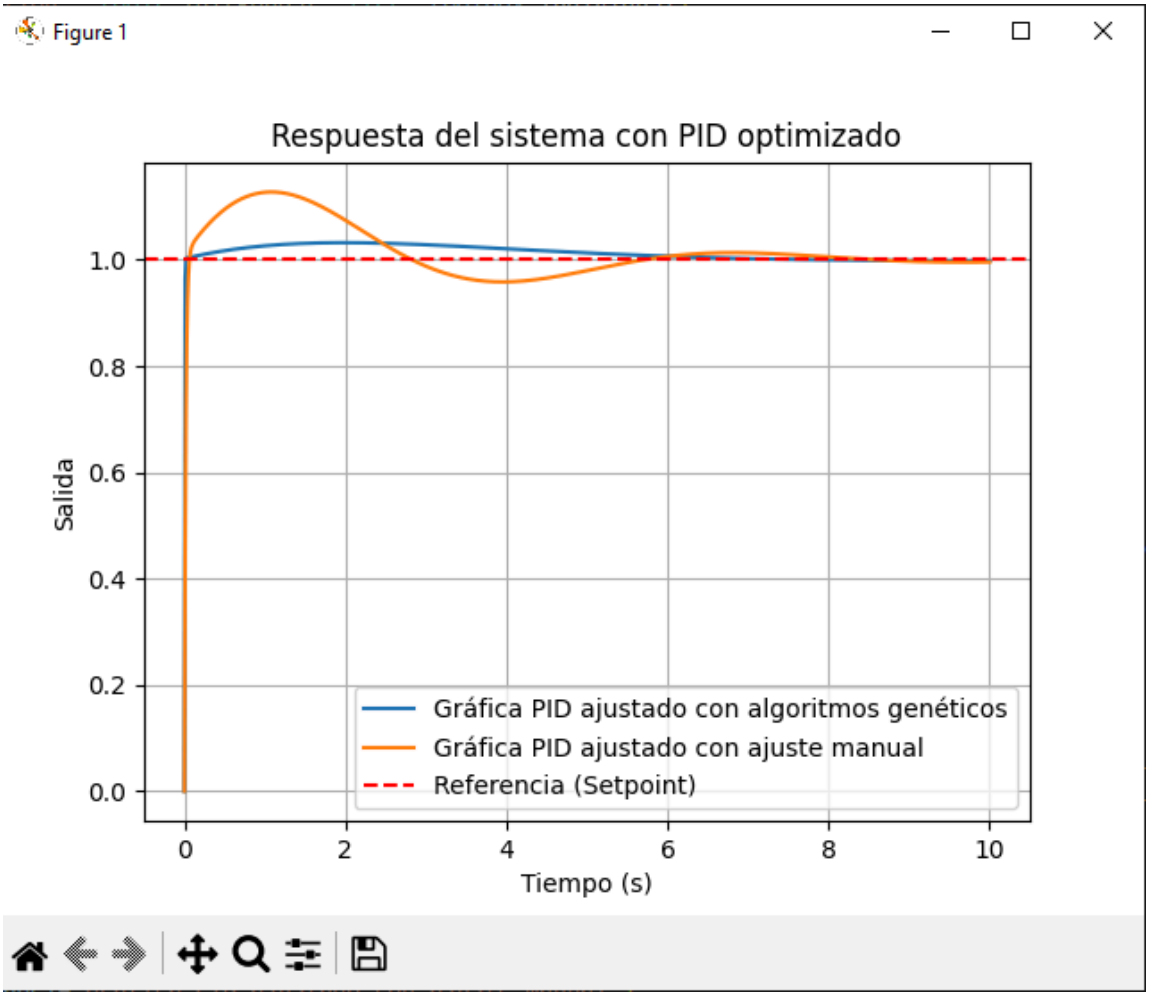


Fig. 6. Comparación de control ajustado manualmente y por medio de algoritmos genéticos.

En la gráfica se muestran dos resultados de controladores PID: uno ajustado con **algoritmos genéticos** (línea azul) y otro ajustado de forma **manual** (línea naranja). Estas son las comparaciones clave entre ambos:

✔ **Respuesta inicial (Rise Time)**

- PID ajustado con algoritmos genéticos (azul): Tiene un tiempo de respuesta más rápido, ya que alcanza el setpoint (referencia) más pronto que el PID manual.

- PID ajustado manualmente (naranja): Es ligeramente más lento para alcanzar el setpoint.

✔ **Sobre impulso (Overshoot)**

- PID ajustado con algoritmos genéticos: Muestra un sobreimpulso menor o prácticamente inexistente, lo que indica un comportamiento más controlado.
- PID ajustado manualmente: Presenta un sobreimpulso significativo al inicio (se eleva por encima del setpoint) antes de estabilizarse.

✔ **Estabilización (Settling Time)**

- PID ajustado con algoritmos genéticos: La salida se estabiliza más rápidamente alrededor del setpoint.
- PID ajustado manualmente: Tarda más en estabilizarse debido a las oscilaciones posteriores al sobreimpulso inicial.

✔ **Oscilaciones**

- PID ajustado con algoritmos genéticos: Tiene menos oscilaciones alrededor del setpoint, lo que resulta en una respuesta más suave.
- PID ajustado manualmente: Hay oscilaciones evidentes alrededor del setpoint antes de estabilizarse.

✔ **Precisión**

- PID ajustado con algoritmos genéticos: La salida converge con mayor precisión al valor del setpoint, con un error más pequeño en estado estable.
- PID ajustado manualmente: Aunque se estabiliza, parece tener pequeñas desviaciones antes de converger.

✔ **Robustez ante perturbaciones**

- PID ajustado con algoritmos genéticos: Se comportará mejor frente a perturbaciones, reaccionando de forma más controlada y recuperándose más rápido con menor error transitorio.
- PID ajustado manualmente: Será más sensible a perturbaciones, presentando oscilaciones más grandes y una recuperación más lenta, lo que puede comprometer el rendimiento del sistema.

En el caso de perturbaciones simularemos una perturbación en tiempo 5 seg. De la planta con un control pid ajustado manualmente, dándonos el siguiente resultado:

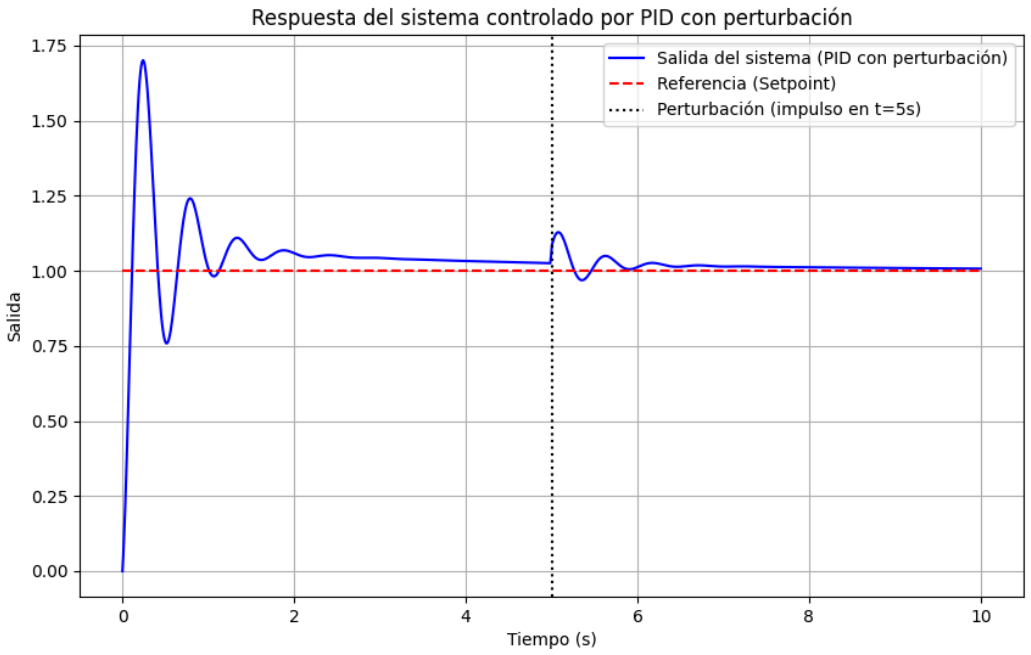


Fig. 7. PID con perturbación a t=5s, control ajustado manualmente.

Y el comportamiento de un PID ajustado ante una perturbación en t=5s.

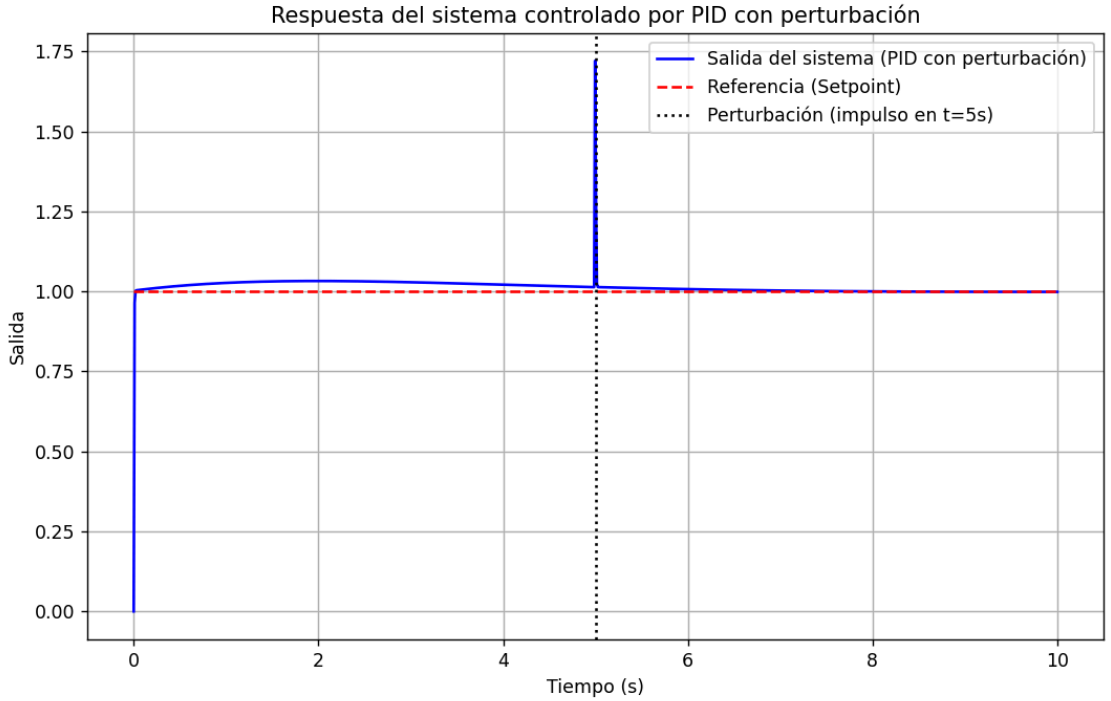


Fig. 8. Perturbación en t=5s con PID ajustado con algoritmo genético.

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?

- Un ángulo inicial mayor del péndulo $\theta(0) = 0.2rad$
- Una posición inicial desplazada del carro $(x(0)=0.1m)$

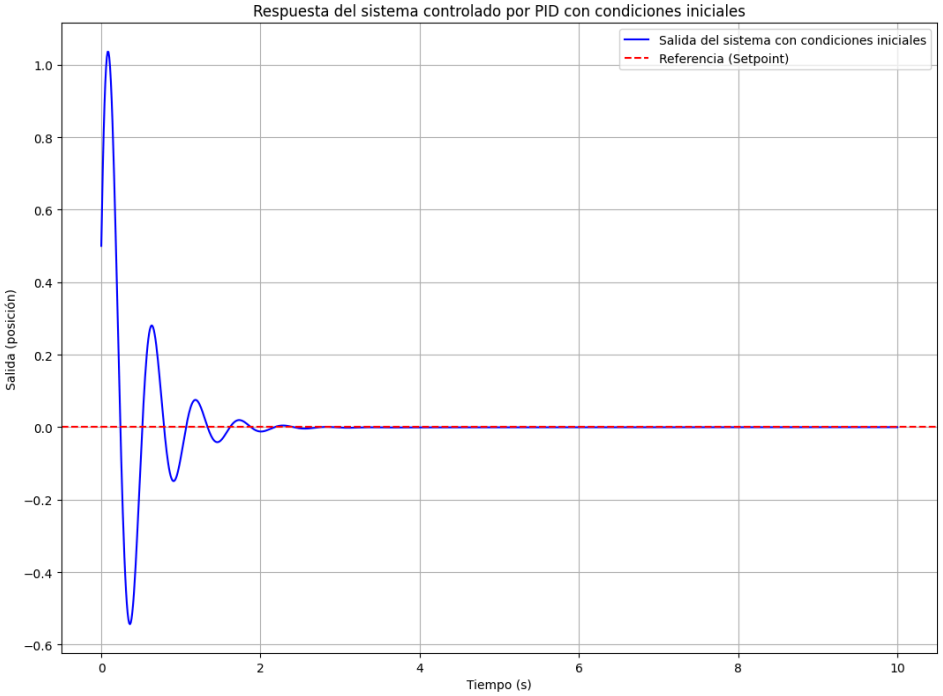


Fig. 9. Con ángulo de inicio y desplazamiento de carro en PID ajustado manualmente.

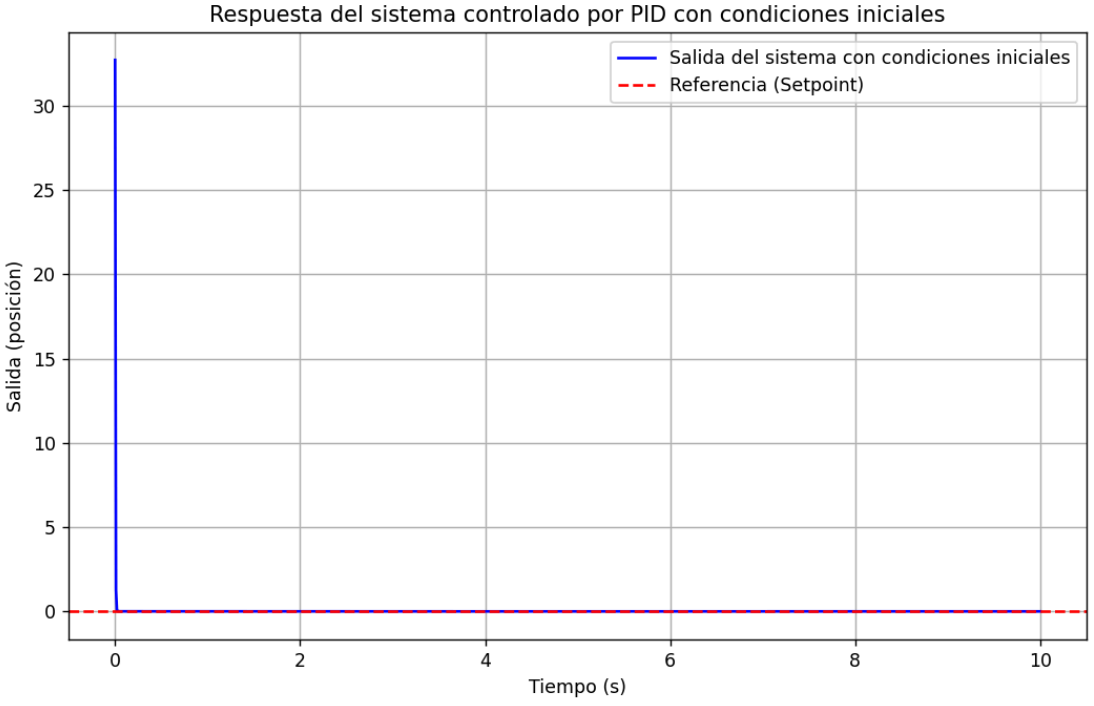


Fig. 10. Con ángulo de inicio y desplazamiento de carro en PID ajustado.

✔ *Comportamiento del PID ajustado manualmente:*

- **Ángulo mayor del péndulo:**

  - **Respuesta típica:**

    Este controlador, al tener parámetros no optimizados, puede resultar en mayores oscilaciones y un sobre impulso más pronunciado. Si el ángulo inicial es muy grande, el controlador podría llegar a un límite donde no logra estabilizar el sistema debido a la insuficiente acción de control.

  - **Riesgo:** Si las oscilaciones no están bien amortiguadas, el sistema puede entrar en un estado inestable, especialmente si la planta tiene no linealidades significativas.

- **Posición inicial desplazada del carro:**

  - **Respuesta típica:**

    Un desplazamiento inicial del carro hará que el controlador trate de compensar, pero las oscilaciones observadas en la gráfica original indican que el ajuste manual podría sobre compensar, aumentando el tiempo de estabilización.

  - **Riesgo:** La recuperación al setpoint será más lenta y el error transitorio más elevado.

✔ ***Comportamiento del PID optimizado (algoritmos genéticos):***

- **Ángulo mayor del péndulo:**

  - **Respuesta típica:**

    Este controlador, al estar optimizado, tenderá a reaccionar de forma más suave y controlada a ángulos iniciales más grandes. Su acción derivativa contribuirá a amortiguar las oscilaciones, y la acción proporcional y la integral minimizarán el error.

    Sin embargo, si el ángulo inicial supera un rango lineal razonable, la planta puede comportarse de forma no lineal, limitando la eficacia del PID.

- **Posición inicial desplazada del carro:**

  - **Respuesta típica:**

    La respuesta será más eficiente en términos de tiempo de estabilización y menor sobre impulso. El ajuste optimizado del controlador permite una

mejor adaptación al desplazamiento inicial, rechazando perturbaciones con mayor rapidez.

b) Después de la optimización, ¿Qué cambios observas en la curva de error del sistema?

✔ ***Disminución del error transitorio:***

- ***Antes de la optimización, el error transitorio es más alto debido a un sobre impulso significativo y a oscilaciones prolongadas antes de estabilizarse.***
- ***Después de la optimización, el error transitorio disminuye considerablemente, ya que el sistema responde de manera más rápida y controlada al setpoint. Esto se traduce en un tiempo de respuesta más corto y oscilaciones más pequeñas.***

✔ ***Eliminación o reducción del error estacionario:***

- ***Con el PID ajustado manualmente, puede haber un error estacionario si el término integral (Ki) no es suficientemente alto o está mal ajustado.***
- ***Con la optimización, el error estacionario se elimina o reduce a valores insignificantes debido a un balance más preciso entre los términos proporcional, integral y derivativo.***

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

Aspecto	PID Manual	PID Optimizado
Estabilidad inicial	Menor estabilidad, mayores oscilaciones.	Mayor estabilidad, respuesta más suave.
Overshoot	Más elevado.	Reducido o casi nulo.
Tiempo de estabilización	Mayor.	Menor, estabilización rápida.
Rechazo de perturbaciones	Más lento y con mayores oscilaciones.	Rápido y eficiente.
Robustez	Sensible a cambios iniciales.	Más robusto y adaptable.

Aspecto	PID Manual	PID Optimizado
Error estacionario	Posible error persistente.	Error mínimo o eliminado.

Cabe resaltar que, a pesar de estos valores calculados con algoritmos genéticos, es posible hacer correcciones manuales a partir de estos datos para ajustar cierto comportamiento.

En resumen, un controlador optimizado proporciona un desempeño más preciso, estable y rápido, mientras que los ajustes manuales pueden ofrecer un control menos eficiente, con mayor riesgo de inestabilidad y error residual.

- **Simulación de péndulo invertido con GYM y comparar su desempeño con el enfoque clásico PID.**

Actualmente, gym se renombró como gymnasium y ha tenido varias actualizaciones recientes para mantener la compatibilidad con Python y mejorar su funcionalidad.

Aquí tenemos el código en Python para la simulación del péndulo invertido con la función de transferencia del péndulo desarrollada en la pregunta 1.

```
import gymnasium as gym
import pygame
import numpy as np
from gymnasium import spaces
import math
from scipy import signal

class PenduloInvertidoEnv(gym.Env):
    metadata = {"render_modes": ["human", "rgb_array"], "render_fps": 30}

    # Parámetros del sistema
    M = 2  # Masa del carro (kg)
    m = 0.5  # Masa del péndulo (kg)
    l = 1  # Longitud del péndulo (m)
    g = 9.81 # Aceleración de la gravedad (m/s²)

    def __init__(self, render_mode=None, numerator=None, denominator=None):
        self.gravity = g
        self.masscart = M
        self.masspole = m
        self.total_mass = self.masspole + self.masscart
        self.length = l
        self.polemass_length = self.masspole * self.length
        self.force_mag = 10.0
        self.tau = 0.02  # seconds between state updates

        # Ángulo en el que se considera que el episodio ha fallado
        self.theta_threshold_radians = 12 * 2 * math.pi / 360
```

```
        self.x_threshold = 2.4

        # Definir la función de transferencia
        if numerator is None or denominator is None:
            # Función de transferencia por defecto si no se proporciona
            self.numerator = [1]
            self.denominator = [(M*l), 0, -(M+m)*g]
        else:
            self.numerator = numerator
            self.denominator = denominator

        self.sys = signal.TransferFunction(self.numerator, self.denominator)

        high = np.array(
            [
                self.x_threshold * 2,
                np.finfo(np.float32).max,
                self.theta_threshold_radians * 2,
                np.finfo(np.float32).max,
            ],
            dtype=np.float32,
        )

        self.action_space = spaces.Discrete(2)
        self.observation_space = spaces.Box(-high, high, dtype=np.float32)

        self.render_mode = render_mode
        self.screen = None
        self.clock = None
        self.state = None
        self.steps_beyond_terminated = None

    def step(self, action):
        err_msg = f"{action!r} ({type(action)}) invalid"
        assert self.action_space.contains(action), err_msg
        assert self.state is not None, "Call reset before using step method."
        x, x_dot, theta, theta_dot = self.state
        force = self.force_mag if action == 1 else -self.force_mag

        # Usar la función de transferencia para calcular la aceleración angular
        t = np.linspace(0, self.tau, 2)
        t, y = signal.step(self.sys, T=t)
        thetaacc = y[-1] * force  # Usamos el último valor de y como aceleración

        # Calcular la aceleración del carro
        temp = (force + self.polemass_length * theta_dot ** 2 * math.sin(theta)) / self.total_mass
        xacc = temp - self.polemass_length * thetaacc * math.cos(theta) / self.total_mass

        # Actualizar el estado
        x = x + self.tau * x_dot
        x_dot = x_dot + self.tau * xacc
        theta = theta + self.tau * theta_dot
        theta_dot = theta_dot + self.tau * thetaacc

        self.state = (x, x_dot, theta, theta_dot)

        terminated = bool(
            x < -self.x_threshold
            or x > self.x_threshold
            or theta < -self.theta_threshold_radians
            or theta > self.theta_threshold_radians
        )

        if not terminated:
```

```
            reward = 1.0
        elif self.steps_beyond_terminated is None:
            self.steps_beyond_terminated = 0
            reward = 1.0
        else:
            self.steps_beyond_terminated += 1
            reward = 0.0

        if self.render_mode == "human":
            self.render()

        return np.array(self.state, dtype=np.float32), reward, terminated, False, {}

    def reset(self, seed=None, options=None):
        super().reset(seed=seed)
        self.state = self.np_random.uniform(low=-0.05, high=0.05, size=(4,))
        self.steps_beyond_terminated = None

        if self.render_mode == "human":
            self.render()
        return np.array(self.state, dtype=np.float32), {}

    def render(self):
        if self.render_mode is None:
            gym.logger.warn(
                "You are calling render method without specifying any render mode."
            )
            return

        try:
            import pygame
            from pygame import gfxdraw
        except ImportError:
            raise DependencyNotInstalled(
                "pygame is not installed, run `pip install gym[classic_control]`"
            )

        screen_width = 600
        screen_height = 400

        world_width = self.x_threshold * 2
        scale = screen_width / world_width
        polewidth = 10.0
        polelen = scale * (2 * self.length)
        cartwidth = 50.0
        cartheight = 30.0

        if self.state is None:
            return None

        x = self.state

        if self.screen is None:
            pygame.init()
            if self.render_mode == "human":
                pygame.display.init()
                self.screen = pygame.display.set_mode((screen_width, screen_height))
            else:  # mode == "rgb_array"
                self.screen = pygame.Surface((screen_width, screen_height))
        if self.clock is None:
            self.clock = pygame.time.Clock()

        self.surf = pygame.Surface((screen_width, screen_height))
        self.surf.fill((255, 255, 255))
```

```
        l, r, t, b = -cartwidth / 2, cartwidth / 2, cartheight / 2, -cartheight / 2
        axleoffset = cartheight / 4.0
        cartx = x[0] * scale + screen_width / 2.0  # MIDDLE OF CART
        carty = 100  # TOP OF CART
        cart_coords = [(l, b), (l, t), (r, t), (r, b)]
        cart_coords = [(c[0] + cartx, c[1] + carty) for c in cart_coords]
        gfxdraw.aapolygon(self.surf, cart_coords, (0, 0, 0))
        gfxdraw.filled_polygon(self.surf, cart_coords, (0, 0, 0))

        l, r, t, b = (
            -polewidth / 2,
            polewidth / 2,
            polelen - polewidth / 2,
            -polewidth / 2,
)

        pole_coords = []
        for coord in [(l, b), (l, t), (r, t), (r, b)]:
            coord = pygame.math.Vector2(coord).rotate_rad(-x[2])
            coord = (coord[0] + cartx, coord[1] + carty + axleoffset)
            pole_coords.append(coord)
        gfxdraw.aapolygon(self.surf, pole_coords, (202, 152, 101))
        gfxdraw.filled_polygon(self.surf, pole_coords, (202, 152, 101))

        gfxdraw.aacircle(
            self.surf,
            int(cartx),
            int(carty + axleoffset),
            int(polewidth / 2),
            (129, 132, 203),
        )
        gfxdraw.filled_circle(
            self.surf,
            int(cartx),
            int(carty + axleoffset),
            int(polewidth / 2),
            (129, 132, 203),
        )

        gfxdraw.hline(self.surf, 0, screen_width, carty, (0, 0, 0))

        self.surf = pygame.transform.flip(self.surf, False, True)
        self.screen.blit(self.surf, (0, 0))
        if self.render_mode == "human":
            pygame.event.pump()
            self.clock.tick(self.metadata["render_fps"])
            pygame.display.flip()

        elif self.render_mode == "rgb_array":
            return np.transpose(
                np.array(pygame.surfarray.pixels3d(self.screen)), axes=(1, 0, 2)
            )

    def close(self):
        if self.screen is not None:
            import pygame

            pygame.display.quit()
            pygame.quit()
            self.isopen = False

# Uso del entorno
```

```python
# Parámetros del sistema
M = 2  # Masa del carro (kg)
m = 0.5  # Masa del péndulo (kg)
l = 1  # Longitud del péndulo (m)
g = 9.81 # Aceleración de la gravedad (m/s²)

# Definir la función de transferencia
numerator = [1]
denominator = [(M*l), 0, -(M+m)*g]

env = PenduloInvertidoEnv(render_mode="human", numerator=numerator, denominator=denominator)
observation, info = env.reset(seed=42)

for _ in range(1000):
    action = env.action_space.sample()  # Acción aleatoria
    observation, reward, terminated, truncated, info = env.step(action)

    if terminated or truncated:
        observation, info = env.reset()

env.close()
```

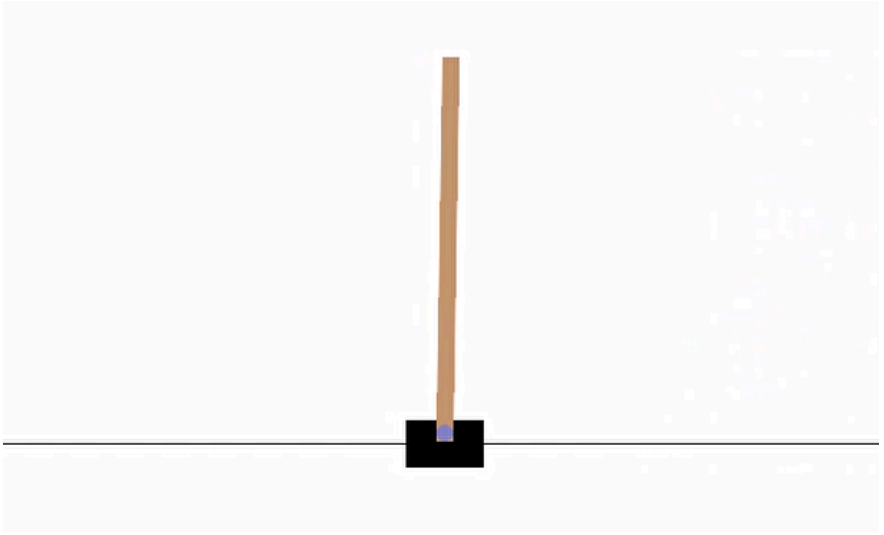


Fig. 11 Resultado de la simulación.

Con el control PID clásico según las gráficas se puede ver una estabilización del péndulo. Con la simulación en Gymnasium y usando la misma función de transferencia, puede ver en la animación que existe un desbalance. Habría que agregar un controlador a adicional a la simulación.

CONCLUSIONES

1. Relevancia de los modelos matemáticos en el diseño de controladores

El desarrollo de funciones de transferencia y la determinación de la estabilidad a través de análisis matemáticos son esenciales para comprender el comportamiento del sistema en lazo abierto. La utilización de herramientas como la transformada de Laplace facilita la simplificación del modelo físico a uno matemático, proporcionando una base sólida para el diseño de controladores. Este enfoque permite evaluar la estabilidad inicial y determinar la necesidad de controladores como el PID para estabilizar sistemas inherentemente inestables, como el péndulo invertido.

2. Comparación de métodos de ajuste manual y optimización automatizada

El ajuste manual de controladores PID, aunque efectivo en ciertas situaciones, se queda corto frente a métodos de optimización automatizada como los algoritmos genéticos. La optimización computacional proporciona parámetros que mejoran el desempeño del controlador en términos de estabilidad, tiempo de respuesta y reducción de sobreimpulsos. Sin embargo, también se identifican limitaciones prácticas, como el costo computacional y la dependencia de la precisión del modelo, lo que subraya la importancia de integrar métodos híbridos que combinen automatización y ajustes manuales según sea necesario.

3. Impacto del diseño del controlador en la robustez del sistema

El análisis del comportamiento del sistema con diferentes controladores (P, PI, PD y PID) destaca cómo cada componente afecta la respuesta del sistema. Los controladores PID optimizados, en particular, ofrecen un balance óptimo entre rapidez, precisión y robustez ante perturbaciones. No obstante, la simulación en entornos como Gymnasium muestra que las dinámicas no modeladas pueden causar desbalances, lo que sugiere la necesidad de explorar estrategias de control complementarias o adaptativas para mejorar la robustez frente a condiciones no previstas.