Laboratorio: Control de un Péndulo Invertido con PID en Python

Luis Felipe Rubio Morelo
Universidad Militar Nueva Granada
Laboratorio inteligencia artificial

29 de agosto de 2025

1. Introducción

El péndulo invertido es uno de los problemas clásicos en control automático debido a su inestabilidad inherente. Se utiliza como sistema de referencia en la enseñanza y validación de técnicas de control, pues representa un modelo simplificado de problemas reales como el control de cohetes, robots bípedos y sistemas de transporte.

En este laboratorio se implementó una simulación en Python de un péndulo invertido montado sobre un carro, controlado por un controlador PID (Proporcional-Integral-Derivativo). El objetivo principal es mantener el péndulo en posición vertical ($\theta = 0$) mediante la acción del carro, que se desplaza horizontalmente para compensar las perturbaciones iniciales.

2. Objetivos

- Implementar un modelo físico-matemático del sistema carro–péndulo.
- Diseñar un controlador PID para estabilizar el ángulo del péndulo.
- Visualizar la dinámica del sistema en una simulación gráfica interactiva.
- Analizar el comportamiento del controlador frente a condiciones iniciales.

3. Marco Teórico

3.1. Dinámica del péndulo invertido

El sistema está compuesto por:

- Carro de masa M.
- Péndulo de masa m y longitud L.
- Ángulo θ respecto a la vertical.

Las ecuaciones de movimiento que gobiernan la dinámica son:

$$\ddot{x} = \frac{F - b\dot{x} + m\sin(\theta)(L\dot{\theta}^2 + g\cos(\theta))}{M + m\sin^2(\theta)} \tag{1}$$

$$\ddot{\theta} = \frac{-F\cos(\theta) - mL\dot{\theta}^2\cos(\theta)\sin(\theta) - (M+m)g\sin(\theta) + b\dot{x}\cos(\theta)}{L(M + m\sin^2(\theta))} \tag{2}$$

donde:

- x: posición del carro.
- F: fuerza de control aplicada.
- b: coeficiente de fricción.

3.2. Control PID

El controlador PID calcula la fuerza aplicada sobre el carro en función del error del ángulo:

$$F(t) = -\left[K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}\right] \tag{3}$$

donde:

- $e(t) = \theta(t)$ (error angular).
- K_p: ganancia proporcional.
- K_i: ganancia integral.
- K_d: ganancia derivativa.

3.3. Control Difuso

A diferencia del controlador PID, el control difuso no requiere un modelo matemático exacto del sistema. En su lugar, utiliza reglas heurísticas basadas en lógica difusa para decidir la fuerza F aplicada al carro según el ángulo θ y su velocidad angular $\dot{\theta}$.

Definición de variables

- **Entrada 1:** Ángulo θ con etiquetas `neg`, `zero`, `pos`.
- **Entrada 2:** Velocidad angular $\dot{\theta}$ con etiquetas `neg`, `zero`, `pos`.
- **Salida:** Fuerza F aplicada sobre el carro con etiquetas `left`, `zero`, `right`.

Cada variable se representó con funciones de membresía triangulares, por ejemplo:

$$\mu_{\text{neg}}(\theta) = \text{trimf}(\theta; -0{,}5, -0{,}25, 0), \quad \mu_{\text{zero}}(\theta) = \text{trimf}(\theta; -0{,}1, 0, 0{,}1), \quad \mu_{\text{pos}}(\theta) = \text{trimf}(\theta; 0, 0{,}25, 0{,}5)$$

Base de reglas difusas

El conocimiento de control se modeló con un conjunto de reglas tipo “SI–ENTONCES”:

- SI θ es `neg` Y $\dot{\theta}$ es `zero` $\to F$ es `left`.
- SI θ es `pos` Y $\dot{\theta}$ es `zero` $\to F$ es `right`.
- SI θ es `zero` Y $\dot{\theta}$ es `pos` $\to F$ es `left`.
- SI θ es `zero` Y $\dot{\theta}$ es `neg` $\to F$ es `right`.

Estas reglas reflejan la intuición de que, si el péndulo cae hacia un lado, el carro debe moverse en esa dirección para compensar.

Inferencia y defuzzificación

El motor de inferencia combina las reglas activadas para un valor específico de entrada y calcula una función difusa de salida. Finalmente, se aplica el método del **centroide** para obtener un valor crisp de la fuerza F:

$$F = \frac{\int_{\Omega} \mu_F(f)\, f\, df}{\int_{\Omega} \mu_F(f)\, df}$$

4. Metodología

1. Se definieron los parámetros físicos del sistema (masas, gravedad, longitud del péndulo, fricción).
2. Se resolvieron las ecuaciones de movimiento usando integración numérica con paso $dt = 0{,}02\, s$.
3. Se implementó el controlador PID con saturación de la fuerza $|F| \leq 5\, N$.
4. Se utilizó `Pygame` para simular gráficamente el movimiento del carro y el péndulo, con una cámara que sigue al carro.

5. Resultados

- La simulación muestra cómo el carro se desplaza horizontalmente para compensar el ángulo del péndulo.
- El controlador PID logra estabilizar el péndulo dentro de ciertos límites, evitando que este caiga rápidamente.
- Se observó que con ciertas ganancias ($K_p = 20, K_i = 0{,}5, K_d = 15$) el sistema presenta oscilaciones, lo que indica que aún requiere ajuste fino del PID.
- El comportamiento demuestra la interacción directa entre el error angular y el movimiento del carro, evidenciando la retroalimentación del sistema de control.

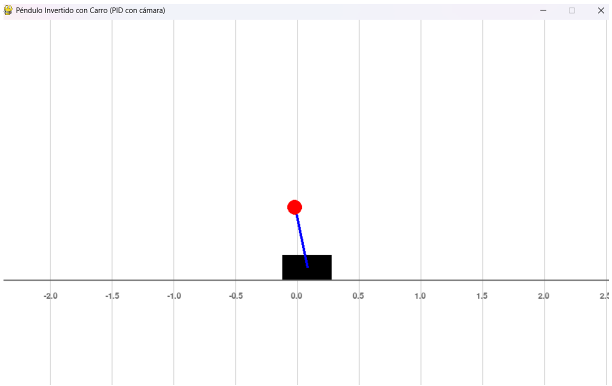

Figura 1: Control Péndulo

6. Conclusiones

- El péndulo invertido es un sistema altamente inestable que requiere control en lazo cerrado.
- El controlador PID es capaz de estabilizar el péndulo, pero la selección de parámetros es crucial.
- La simulación en Python con `Pygame` permite visualizar en tiempo real la dinámica del sistema, lo cual facilita el análisis.
- Se recomienda aplicar métodos de sintonización más precisos (Ziegler–Nichols o LQR) para mejorar la estabilidad.

7. Recomendaciones

- Probar condiciones iniciales más exigentes (ángulos mayores).
- Incluir ruido o perturbaciones externas para simular condiciones reales.
- Comparar el desempeño del PID con controladores más avanzados (LQR, MPC).
- Extender el modelo a más grados de libertad (doble péndulo invertido).

A. Código de Simulación en Python

A continuación, se presenta el código utilizado para la simulación del sistema carro–péndulo invertido con un controlador PID. El mismo fue implementado en `Python` utilizando la librería `Pygame` para la visualización gráfica.

Listing 1: Simulación del péndulo invertido con PID

```
import pygame
import numpy as np


g = 9.81
```

```
M = 1.0
m = 2.0
L = 0.5
dt = 0.02
b = 20


Kp = 20.0
Ki = 0.5
Kd = 15.0


x = 0.0
x_dot = 0.0
theta = np.deg2rad(5)
theta_dot = 0.0


integral_error = 0.0
prev_error = 0.0


F_max = 5.0


pygame.init()
WIDTH, HEIGHT = 1000, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("P ndulo Invertido con Carro (PID con c mara)"
clock = pygame.time.Clock()

origin_y = HEIGHT // 2 + 100  # altura del riel
scale = 200  # escala: 1m = 200px

running = True


while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False


    error = theta  # queremos que theta = 0
    integral_error += error * dt
    derivative_error = (error - prev_error) / dt
    prev_error = error
    F = -(Kp * error + Ki * integral_error + Kd * derivative_error)
```

```
F = max(min(F, F_max), -F_max)


sin_theta = np.sin(theta)
cos_theta = np.cos(theta)
denom = M + m * sin_theta**2

x_ddot = (F - b*x_dot + m * sin_theta * (L * theta_dot**2 + g * cos_the
theta_ddot = (-F * cos_theta - m * L * theta_dot**2 * cos_theta * sin_t
              (M + m) * g * sin_theta + b * x_dot * cos_theta) / (L * d


x_dot += x_ddot * dt
x += x_dot * dt
theta_dot += theta_ddot * dt
theta += theta_dot * dt


screen.fill((255, 255, 255))


offset_x = WIDTH//2 - int(x*scale)


for k in range(-20, 21):
    pos_x = int(k*0.5*scale) + offset_x
    if 0 <= pos_x <= WIDTH:
        pygame.draw.line(screen, (200, 200, 200), (pos_x, 0), (pos_x, H
        font = pygame.font.SysFont(None, 20)
        text = font.render(f"{k*0.5:.1f}", True, (100, 100, 100))
        screen.blit(text, (pos_x-10, origin_y+40))


cart_x = int(x*scale) + offset_x
cart_y = origin_y


pygame.draw.rect(screen, (0, 0, 0), (cart_x-40, cart_y-20, 80, 40))


pend_x = cart_x + int(L*scale * np.sin(theta))
pend_y = cart_y - int(L*scale * np.cos(theta))
pygame.draw.line(screen, (0, 0, 255), (cart_x, cart_y), (pend_x, pend_y
pygame.draw.circle(screen, (255, 0, 0), (pend_x, pend_y), 12)
```

```
    pygame.draw.line(screen, (100, 100, 100), (0, cart_y+20), (WIDTH, cart_

    pygame.display.flip()
    clock.tick(1/dt)

pygame.quit()
```

Anexo: Código en Python del Controlador Difuso

A continuación, se presenta el código completo utilizado en la simulación del sistema carro–péndulo con controlador difuso implementado en Python:

Listing 2: Simulación del péndulo invertido con carro usando Control Difuso en Python

```
import pygame
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# ---------------- Par metros f sicos ----------------
g = 9.81       # gravedad (m/s^2)
M = 1.0        # masa del carro (kg)
m = 2.0        # masa del p ndulo (kg)
L = 0.5        # longitud del p ndulo (m)
dt = 0.02      # paso de integraci n (s)
b = 20         # fricci n del carro

# Estado inicial [x, x_dot, theta, theta_dot]
x = 0.0
x_dot = 0.0
theta = np.deg2rad(10)  # inclinaci n inicial
theta_dot = 0.0

# Saturaci n de fuerza (N)
F_max = 10.0

# ---------------- Controlador Difuso ----------------
# Variables ling  sticas
theta_var = ctrl.Antecedent(np.linspace(-1, 1, 200), 'theta')
theta_dot_var = ctrl.Antecedent(np.linspace(-3, 3, 200), 'theta_dot')
force_var = ctrl.Consequent(np.linspace(-F_max, F_max, 200), 'force')

# Funciones de membres a
theta_var['neg'] = fuzz.trimf(theta_var.universe, [-1, -0.5, 0])
theta_var['zero'] = fuzz.trimf(theta_var.universe, [-0.1, 0, 0.1])
theta_var['pos'] = fuzz.trimf(theta_var.universe, [0, 0.5, 1])

theta_dot_var['neg'] = fuzz.trimf(theta_dot_var.universe, [-3, -1.5, 0])
theta_dot_var['zero'] = fuzz.trimf(theta_dot_var.universe, [-0.5, 0, 0.5])
```

```
theta_dot_var['pos'] = fuzz.trimf(theta_dot_var.universe, [0, 1.5, 3])

force_var['left'] = fuzz.trimf(force_var.universe, [-F_max, -F_max/2, 0])
force_var['zero'] = fuzz.trimf(force_var.universe, [-1, 0, 1])
force_var['right'] = fuzz.trimf(force_var.universe, [0, F_max/2, F_max])

# Reglas difusas
rules = [
    ctrl.Rule(theta_var['neg'] & theta_dot_var['zero'], force_var['left']),
    ctrl.Rule(theta_var['pos'] & theta_dot_var['zero'], force_var['right'])
    ctrl.Rule(theta_var['zero'] & theta_dot_var['neg'], force_var['right'])
    ctrl.Rule(theta_var['zero'] & theta_dot_var['pos'], force_var['left']),
    ctrl.Rule(theta_var['neg'] & theta_dot_var['neg'], force_var['left']),
    ctrl.Rule(theta_var['pos'] & theta_dot_var['pos'], force_var['right']),
    ctrl.Rule(theta_var['zero'] & theta_dot_var['zero'], force_var['zero'])
]

# Construir el sistema de control difuso
system = ctrl.ControlSystem(rules)
fuzzy_controller = ctrl.ControlSystemSimulation(system)

# ---------------- Pygame setup ----------------
pygame.init()
WIDTH, HEIGHT = 1000, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("P ndulo Invertido con Carro (Control Difuso)")
clock = pygame.time.Clock()

origin_y = HEIGHT // 2 + 100  # altura del riel
scale = 200  # escala: 1m = 200px

running = True

# ---------------- Simulaci n ----------------
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # --- Controlador Difuso ---
    fuzzy_controller.input['theta'] = theta
    fuzzy_controller.input['theta_dot'] = theta_dot

    try:
        fuzzy_controller.compute()
        F = fuzzy_controller.output['force']
    except:
        F = 0  # si no hay salida, ponemos fuerza neutra
```

```
# --- Din mica del carro-p ndulo ---
sin_theta = np.sin(theta)
cos_theta = np.cos(theta)
denom = M + m * sin_theta**2

x_ddot = (F - b*x_dot + m * sin_theta * (L * theta_dot**2 + g * cos_the
theta_ddot = (-F * cos_theta - m * L * theta_dot**2 * cos_theta * sin_t
              (M + m) * g * sin_theta + b * x_dot * cos_theta) / (L * d

# Integraci n
x_dot += x_ddot * dt
x += x_dot * dt
theta_dot += theta_ddot * dt
theta += theta_dot * dt

# ---------------------- Dibujar ----------------------
screen.fill((255, 255, 255))

# C mara sigue al carro      offset en X
offset_x = WIDTH//2 - int(x*scale)

# Dibujar eje X
for k in range(-20, 21):
    pos_x = int(k*0.5*scale) + offset_x
    if 0 <= pos_x <= WIDTH:
        pygame.draw.line(screen, (200, 200, 200), (pos_x, 0), (pos_x, H
        font = pygame.font.SysFont(None, 20)
        text = font.render(f"{k*0.5:.1f}", True, (100, 100, 100))
        screen.blit(text, (pos_x-10, origin_y+40))

# Posici n del carro en pantalla
cart_x = int(x*scale) + offset_x
cart_y = origin_y

# Carro
pygame.draw.rect(screen, (0, 0, 0), (cart_x-40, cart_y-20, 80, 40))

# P ndulo
pend_x = cart_x + int(L*scale * np.sin(theta))
pend_y = cart_y - int(L*scale * np.cos(theta))
pygame.draw.line(screen, (0, 0, 255), (cart_x, cart_y), (pend_x, pend_y
pygame.draw.circle(screen, (255, 0, 0), (pend_x, pend_y), 12)

# Piso
pygame.draw.line(screen, (100, 100, 100), (0, cart_y+20), (WIDTH, cart_

pygame.display.flip()
```

```
        clock.tick(1/dt)

pygame.quit()
```