Implementación de un PID en Raspberry Pi PICO de Doble Núcleo

Carlos Emmanuel Ornelas-López[a], Teodoro Alvarez-Sánchez[b], Jesus Antonio Alvarez-Cedillo[c].*

[a]Instituto Politécnico Nacional, cornelas@ipn.mx*, México.*
**[b]Instituto Politécnico Nacional, talvarezs@ipn.mx, México.*
[c]Instituto Politécnico Nacional, jaalvarez@ipn.mx, México.

Resumen

Esta investigación se centra en el diseño e implementación de un sistema de control de velocidad para un motor eléctrico de corriente directa (CD) utilizando un controlador PID en lazo cerrado. La necesidad de un control preciso de la velocidad es fundamental en diversas aplicaciones industriales y robóticas.

Para llevar a cabo este proyecto, se utilizó un microcontrolador de doble núcleo que proporciona la capacidad de procesar la señal del ADC y otro núcleo se encarga del PWM (Modulación por Ancho de Pulso) necesaria para ejecutar el algoritmo PID. Para controlar la potencia suministrada al motor (con encoder rotativo). Además, se empleó un puente H adecuado para manejar los voltajes requeridos por el motor, garantizando un funcionamiento eficiente y seguro.

Los resultados obtenidos demostraron que el sistema diseñado es capaz de mantener la velocidad del motor dentro de los parámetros deseados, incluso ante variaciones en la carga, mostrando una respuesta rápida y estable ante cambios en la referencia de velocidad.

En conclusión, la implementación del sistema de control PID ha demostrado ser viable para regular la velocidad del motor eléctrico CD, mejorando su rendimiento y ofreciendo una solución escalable y adaptable a diferentes aplicaciones, lo que abre nuevas oportunidades para futuras investigaciones y desarrollos en el campo del control automático de motores.

Palabras clave— Encoder, microcontrolador, núcleo, PID, puente H.

Abstract

This research focuses on designing and implementing a speed control system for a direct current (DC) electric motor utilizing a closed-loop proportional-integral-derivative (PID) controller. Precise speed control is crucial in various industrial and robotic applications.

To carry out this project, a dual-core microcontroller was used, which provides the ability to process the ADC signal, while another core is responsible for the PWM (Pulse Width Modulation) necessary to run the PID algorithm. To control the power supplied to the motor. In addition, a suitable H-bridge was employed to handle the voltages required by the motor, ensuring efficient and safe operation.

The results showed that the designed system can maintain the motor speed within the desired parameters, even in the face of variations in load, showing a fast and stable response to changes in the speed reference.

In conclusion, the implementation of the PID control system has proven to be viable for regulating the speed of the DC electric motor, enhancing its performance, and providing a scalable and adaptable solution for various applications, which opens up new opportunities for future research and development in the field of automatic motor control.

Keywords*— Encoder, microcontroller, Core, PID, H bridge*

1. INTRODUCCIÓN

El control de sistemas es un área multidisciplinaria que integra el diseño de máquinas, la electrónica y los sistemas embebidos. En este proyecto, se implementa un sistema de control utilizando un microcontrolador Raspberry Pi Pico, aprovechando sus dos núcleos para optimizar el rendimiento disponible [1].

La Raspberry Pi Pico incorpora el RP2040, que cuenta con un procesador Arm Cortex-M0+, que es familiar de los STM32, RISC de 32 bits. Estos destacan por su potencia, versatilidad y herramientas avanzadas de desarrollo, lo que los hace ideales para aplicaciones industriales y complejas. Sin embargo, su uso requiere mayor experiencia técnica y tiene una curva de aprendizaje más pronunciada. La elección entre distintas plataformas depende tanto de las necesidades del proyecto como del nivel de conocimiento del usuario.

Arduino, por otro lado, es una opción ideal para iniciadores debido a su facilidad de uso, su amplia comunidad y su compatibilidad con sensores y módulos. No obstante, su capacidad de procesamiento es limitada, lo que lo hace menos adecuado para aplicaciones avanzadas.

Para la implementación de control PID, existen diferentes enfoques: el método clásico, versiones optimizadas con filtrado, modelos autoajustables y estrategias basadas en modelos (MPC). Cada uno varía en complejidad y en los recursos requeridos.

En cuanto a la selección de plataformas, Arduino es ideal para proyectos educativos y aplicaciones simples, aunque con limitaciones en potencia de procesamiento. Raspberry Pi Pico ofrece un equilibrio entre rendimiento y facilidad de uso, lo que la hace versátil para el desarrollo de prototipos. Por su parte, STM32 destaca en aplicaciones avanzadas gracias a su alto rendimiento y capacidad de personalización, aunque su curva de aprendizaje es más exigente. La elección dependerá de los requerimientos específicos del proyecto y del nivel de experiencia del usuario.

2. DISEÑO

En este proyecto se desea controlar la velocidad de un motor eléctrico de CD con una ley de control PID en lazo cerrado. Se selecciona un motor con encoder rotativo, un puente H controlado por PWM como etapa de potencia y un potenciómetro como entrada de referencia; se encuentran los siguientes requisitos:

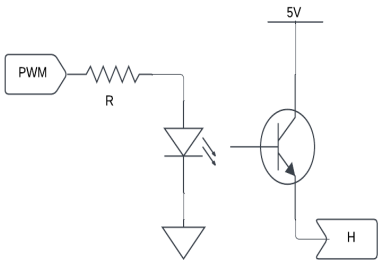
- Microcontrolador con módulos de convertidor analógico-digital y PWM.
- Motor de CD con encoder rotativo.
- Puente H acondicionado a los voltajes de trabajo.

2.2 Diseño eléctrico

El microcontrolador Raspberry pico utilizado, cuenta con dos núcleos de procesamiento, convertidor analógico-digital y módulo de PWM. También se propone un motor de 12V GM 25-370 con encoder con efecto Hall y un puente H L298N, el cuál cumple las características de voltaje.

Cabe mencionar que para evitar el ruido eléctrico que puede ocasionar la rápida conmutación del motor por los pulsos PWM y las características inductivas del motor de CD, se decide colocar una etapa de opto-acoplamiento entre el puente H y el microcontrolador. El circuito implementado con un optoacoplador PC817 se ve en la figura 1, donde R debe ser alrededor de 3.3V/20mA, por lo que se selecciona una resistencia de 220 Ohm. Ya que la entrada de control del puente H puede funcionar incluso con 3.3V; se toma la configuración de emisor común a la salida del optoacoplador [2].

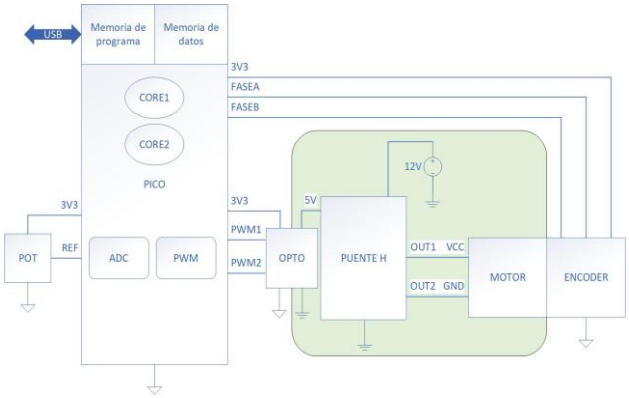
Figura 1. Circuito opto-acoplador.



Fuente: elaboración propia.

Por lo tanto, se diseña el diagrama de bloques de la figura 2, donde se resalta la zona verde de alto voltaje que es opto acoplada por el circuito de la figura 1 y que no comparte referencia a tierra con el circuito digital. La salida de 5V del puente H es una fuente de voltaje que viene de un regulador integrado en el dispositivo seleccionado [3].

Figura 2. Diagrama eléctrico.



Fuente: elaboración propia.

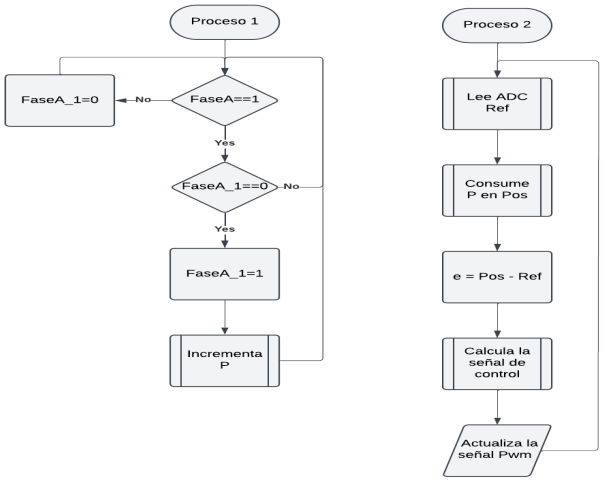
2.3 Diseño lógico.

Haciendo uso de los dos núcleos del microcontrolador utilizado, se propone un proceso en el que se calcula constantemente la cantidad de pasos leídos del encoder en el motor, mientras que un segundo proceso lee la referencia del potenciómetro y estima la señal de control que se enviará al motor. Se describen a continuación las variables utilizadas en el programa:

- FaseA: Fase A de entrada del encoder.
- FaseA_1: retraso de FaseA.
- P: Pasos dados por el encoder.
- Ref: Referencia analógica en el potenciómetro.
- Pos: Posición o estado actual del motor (velocidad actual).
- e: Señal de error.

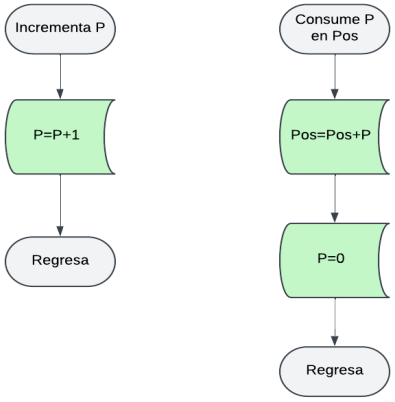
Por lo tanto, se diseñan los procesos de la figura 3. Donde "Lee ADC Ref" obtiene el valor analógico del potenciómetro y lo almacena en la variable de referencia. "Calcula la señal de control" y "Actualiza la señal de Pwm" obtienen el ancho de pulso para aumentar o disminuir la velocidad del motor.

Figura 3. Diagrama de flujo de los procesos a implementar.



Fuente: elaboración propia

Figura 4. Subprocesos que actualizan y consumen los pasos dados por el encoder.



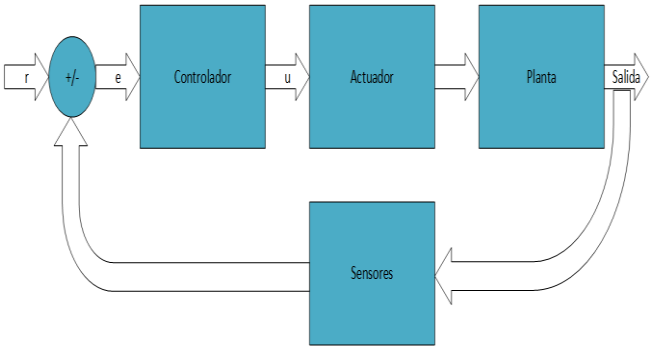
Fuente: elaboración propia.

Por lo tanto, el subproceso "Incrementa P" ocurre cada que la fase tiene un flanco de subida y el subproceso "Consume P en Pos" agrega los pasos obtenidos por el proceso 1 y los agrega a la variable "Pos", que posteriormente será utilizada en el cálculo de la señal de control. Estos últimos subprocesos se muestran en la figura 4 y modifican a la localidad de memoria P [3].

2.4 Control

Se aplica un control de lazo cerrado como se muestra en la figura 5, donde la planta es el motor eléctrico, el sensor es el encoder, el actuador es la etapa de potencia y el controlador el microcontrolador [4].

Figura 5. Control en lazo cerrado



Fuente: elaboración propia.

Se implementa además la ley de control PID de la ecuación (1), donde las constantes proporcional, derivativa e integral se estiman en las pruebas [5] [6].

$$u = kp * e + kd * \frac{de}{dt} + ki * \int e\, dt \qquad (1)$$

3. INTEGRACION Y RESULTADOS

La implementación del diagrama presentado en las figuras 2 y 6 se llevó a cabo utilizando el entorno de desarrollo Thonny, que permite programar en MicroPython de manera eficiente.

En este proceso, se generó una señal PWM (Modulación por Ancho de Pulso) con una frecuencia de 1000 Hz, la cual es adecuada para cumplir con las características de conmutación de los componentes seleccionados.

La elección de esta frecuencia se basa en la necesidad de optimizar el rendimiento del sistema, asegurando un control preciso sobre los motores de corriente continua. La modulación por ancho de pulso es fundamental para regular la velocidad y el par motor, permitiendo así un funcionamiento más eficiente y responsivo.

El ajuste de los coeficientes en el controlador PID es un factor clave para garantizar la reproducibilidad del experimento, ya que permite obtener un comportamiento estable y predecible del sistema, bajo distintas condiciones. Un enfoque común para este ajuste es el método de prueba y error, que consiste en modificar progresivamente los valores de los coeficientes Kp, Ki y Kd mientras se observa la respuesta del sistema.

Para establecer un procedimiento estructurado, se recomienda:
1. Se inicia con valores base: Comenzar con Kp pequeño y Ki, Kd, en cero.
2. Ajustar Kp primero: Incrementarlo hasta que el sistema alcance una respuesta rápida sin oscilar excesivamente.
3. Incluir Kd sí hay oscilaciones: Aumentarlo para amortiguar la respuesta y reducir sobre impulsos.
4. Ajustar Ki para corregir errores constantes: Incrementarlo gradualmente hasta eliminar el error en estado estable.
5.Probar diferentes escenarios: Validar el comportamiento del sistema ante variaciones en las condiciones de operación.

Este procedimiento facilita la calibración del controlador y mejora la reproducibilidad del experimento al documentar los valores óptimos obtenidos.

Además, el uso de Thonny facilita la depuración y el desarrollo del código, gracias a su interfaz intuitiva y sus herramientas integradas. Esto resulta especialmente útil al trabajar con MicroPython, ya que permite realizar pruebas rápidas y ajustes en tiempo real.

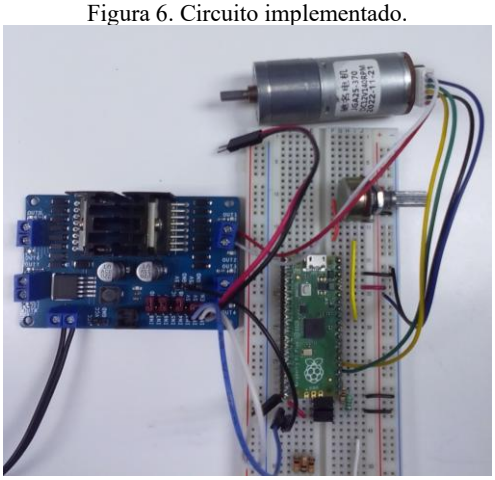
4. CONCLUSIONES Y RECOMENDACIONES

El proyecto de control fue exitosamente implementado mediante uso de un Raspberry pi Pico y la ley de control fue debidamente sintonizada de forma práctica.

Este proyecto integra conocimientos de electrónica, teoría de control y sistemas embebidos. Temas como el control de ruido eléctrico, el acondicionamiento de niveles de voltaje, el uso de lenguajes de programación de alto nivel e incluso la metodología implementada para resolver el problema, son té

fundamentales para resolver problemas futuros y de mayor complejidad y establecer la técnica.

Figura 6. Circuito implementado.



Fuente: elaboración propia.

Este proyecto no solo demuestra la viabilidad técnica sino también ofrece una plataforma para exploraciones y optimizaciones adicionales.

Agradecimientos

Agradecemos las facilidades otorgadas para la realización de este trabajo al Instituto Politécnico Nacional, a través de la Secretaría de Investigación y Posgrado con los proyectos SIP 20241430 y SIP 20241946. A la Unidad Interdisciplinaria de Ingeniería y Ciencias Sociales y Administrativas y al Centro de Investigación y Desarrollo de Tecnología Digital. Asimismo, al Programa de Estímulo al Desempeño de los Investigadores (EDI) y al Programa de Estímulo al Desempeño Docente (EDD).

5. REFERENCIAS

[1] STMicroelectronics, l298 dual full-bridge driver, 2000.
[2] Hot chip, pc817 4 pin dphototransistor photocoupler, Hot chip.
[3] Foneacc co., limited, "Specs DATA sheet for GM25-370," Foneacc motor, 2017. [Online]. Available: https://foneacc-motion.com/Download/Specs-DATA-sheet-for-gm25-370-foneacc-motion.html. [Accessed 2025].
[4] Raspberry Pi Ltd, "RP2024 Datasheet: A microcontroller by Raspberry Pi," Raspberry Pi Ltd, 2024.
[5] J. Gausemeier and S. Moehringer, "VDI 2206 - A new guideline for the design of mechatronic systems," IFAC Publications, pp. 785-790, 2001.
[6] K. Ogata, Discrete-time control systems, 1 ed., Englewood Cliffs, New Jersey: Prentince hall, 1987, p. 994.

Programa en MicroPython

```
from machine import Pin, Timer, PWM
import utime
import _thread

lock = _thread.allocate_lock()
FREQ = 1000
P = 0

adc_read = machine.ADC(26)

FaseA = Pin(18, Pin.IN, Pin.PULL_DOWN)
FaseB = Pin(19, Pin.IN, Pin.PULL_DOWN)
FaseA_1 = False

Motor_pwm1 = PWM(Pin(16))
Motor_pwm1.freq(FREQ)

def   stepCounter():
       global P
      global FaseA_1

    while True:
       # Calcula cuantos pasos ha dado.
       if (FaseA.value()):
          if FaseA_1 == False:
             FaseA_1 = True
             lock.acquire()
             P = P + 1
             lock.release()
          else:
             pass
       else:
          FaseA_1 = False
          utime.sleep(0.001)

def PID():
    global P

    kp = 1.3
    kd = 0.2
    ki = 1.2
    Dt = 1/FREQ
    Pos = 0
    E = 0
    e_1 = 0
    flag = 0

    while True:

        # Lee la referencia
        Ref = adc_read.read_u16()
        if flag == 0:
           flag = 1
        else:
           utime.sleep(0.001)

        # Consume los pasos avanzados y obtener la posicion
        lock.acquire()
        Pos = Pos + P
        P = 0
        lock.release()

        # Calcula el error
        e = Pos - Ref

        # Calcula la parte derivativa de_
        dt = (e-e_1)*FREQ #(e-e_1)/Dt

        # Calcula la parte integral
        E = E * e*Dt

        # Senal de control
        u = kp*e + kd*de_dt + ki*E

        # Saturacion de la senal de control
        if u > 65534:
           u = 65534
        if u < 0:
           u = 0
        # Envia la senal al motor
        Motor_pwm1.duty_u16( int(u) )

        # Recursividad del error
        e_1 = e
        utime.sleep(0.001)

_thread.start_new_thread(stepCounter, ())
     PID()
```