Práctrica 2 – Caracterización de un motor DC

1st Oscar Andres Gutierrez Rivadeneira
11993515015
oscar.gutierrezr@udea.edu.co

2nd Imar Nayeli Jimenez Arango
1007424872
imar.jimenez@udea.edu.co

I. Introducción

Este informe documenta la implementación de un sistema de caracterización de motores DC utilizando una Raspberry Pi Pico, tanto en MicroPython como en Arduino. El objetivo principal fue medir la velocidad angular (RPM) mediante un encoder óptico, controlar el motor mediante PWM, y capturar la curva de reacción del sistema.

II. objetivos

- Implementar un sistema de medición de RPM utilizando un encoder óptico conectado a la Raspberry Pi Pico.
- Controlar la velocidad del motor DC mediante señales PWM generadas desde la Raspberry Pi Pico.
- Caracterizar experimentalmente el motor mediante la curva de reacción, relacionando el voltaje aplicado (PWM) con la velocidad (RPM).
- Modelar el motor como un sistema de primer orden con retardo y validar el modelo comparándolo con los datos experimentales.

III. Metodología

A. Configuración del Hardware

El sistema de caracterización se implementó utilizando un motor DC pequeño sin reductor de velocidad, acoplado a una rueda dentada de 20 vueltas por ranura, un encoder óptico detecta los pulsos con los que se realiza la medición del PWM. El motor DC operaba a 12V y era controlado por el driver de motor L298N. Ver figura 1.

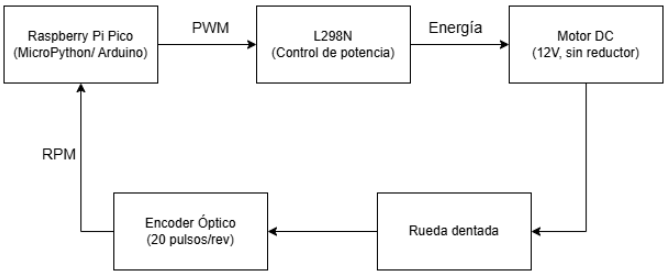

Fig. 1. Diagrama de bloques del sistema de caracterización del motor DC.

B. Implementación del Software

Se desarrollaron dos versiones del sistema: una en Arduino (C/C++) y otra en MicroPython. Ambos programas incluyeron:

- Medición de RPM: Captura de pulsos del encoder y conversión a velocidad angular.
- Generación de PWM: Señal cuadrada ajustable entre 0% y 100% con resolución del 1%.
- Captura de la curva de reacción: Aplicación de escalones de PWM (ascendentes y descendentes) con muestreo a 250 Hz.
- Interfaz de comandos: Soporte para START ¡valor¿ (captura automática) y PWM ¡valor¿ (control manual).

1) Diagrama de flujo Arduino: El código en Arduino utiliza una máquina de estados con tres modos principales: Código

a) IDLE: Espera comandos por serial.
b) Manual PWM: Ajuste manual del ciclo de trabajo (0-100%) con reporte periódico de RPM (2 Hz).
c) Captura automática: Ejecuta una secuencia de escalones PWM (ascendentes/descendentes), midiendo RPM cada 4 ms y almacenando los datos en un buffer. Al finalizar, envía los resultados en formato CSV por el puerto serial.

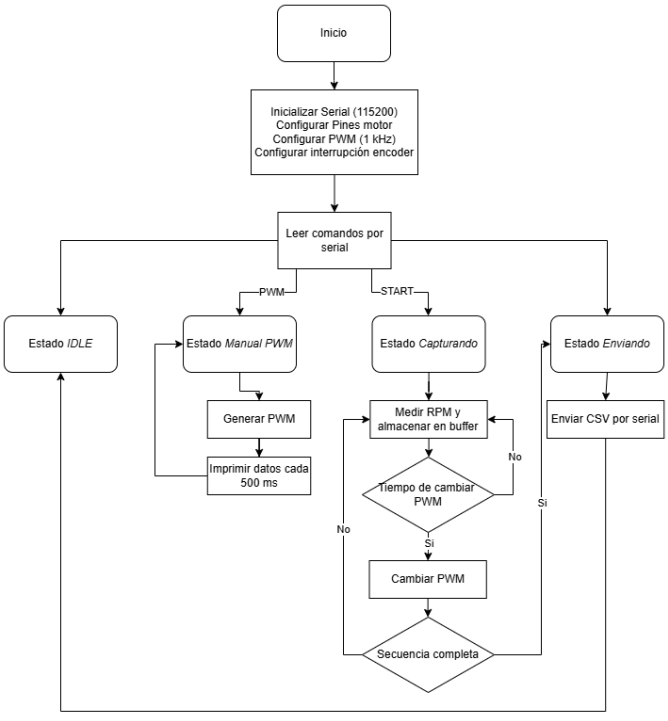

Fig. 2. Diagrama de flujo del código en Arduino.

C. Proceso de caracterización

Se aplicaron cambios en escalón en el PWM (20% de incremento) con intervalos de 2 segundos, tiempo suficiente para que el motor alcanzara su estado estable. Los datos de RPM y PWM fueron almacenados en un buffer y transmitidos vía UART en formato CSV para su posterior análisis.

IV. Resultados

A. Curva de reacción

La caracterización dinámica del motor se realizó mediante el análisis de su curva de reacción ante entradas escalón de PWM. Durante la captura de datos, se identificó que el encoder óptico introdujo ruido en las mediciones de RPM, lo cual distorsionaba temporalmente la curva de reacción. Para mitigar este problema, se implementó un filtro de media móvil en el software, el cual promedia un conjunto de lecturas consecutivas del encoder.

La metodología consistió en aplicar incrementos escalonados del 20% en el ciclo de trabajo del PWM, desde 0% hasta 100%, manteniendo cada nivel durante 2 segundos para garantizar que el motor alcanzara su estado estable. Para cada escalón, se registró la respuesta temporal de velocidad con una frecuencia de muestreo de 250 Hz. Como resultado, la curva de reacción reflejó de manera fiable la relación entre el PWM aplicado y la velocidad del motor, mostrando claramente las tres regiones características: el tiempo muerto inicial, la respuesta transitoria exponencial y el estado estable final.

La caracterización experimental del motor DC se realizó mediante la implementación independiente en Arduino y MicroPython, cuyos resultados se presentan en las Figuras 3 y 4 respectivamente. Ambas gráficas muestran la respuesta del sistema ante excitaciones escalonadas del PWM, evidenciando el comportamiento dinámico característico del motor.

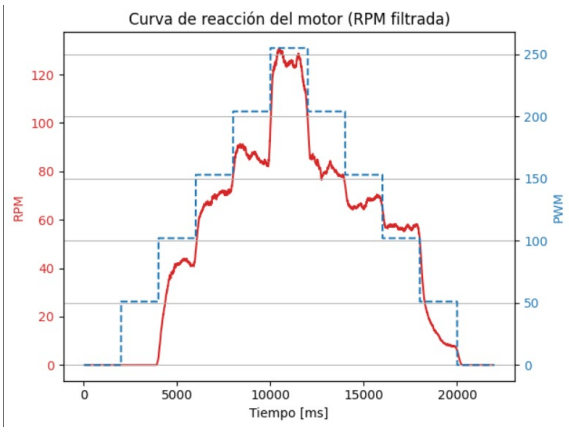

Fig. 3. Curva de reacción obtenida con la implementación en Arduino. La gráfica muestra la relación entre el PWM aplicado (línea roja) y la velocidad del motor en RPM (línea azul) en función del tiempo. Los escalones de 20% en el ciclo de trabajo permitieron caracterizar completamente la respuesta del sistema.

En la Figura 3 se aprecia con claridad la respuesta típica de primer orden con retardo, donde cada incremento del 20% en PWM genera un cambio proporcional en la velocidad

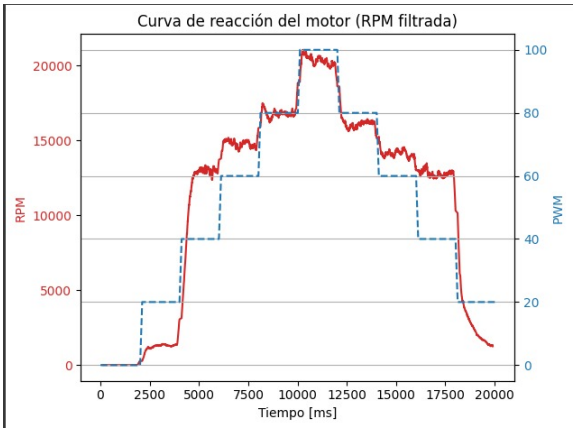

Fig. 4. Curva de reacción obtenida con la implementación en MicroPython. Aunque la escala difiere levemente de la versión Arduino, se observa el mismo patrón de respuesta, validando la consistencia de las mediciones entre ambas plataformas.

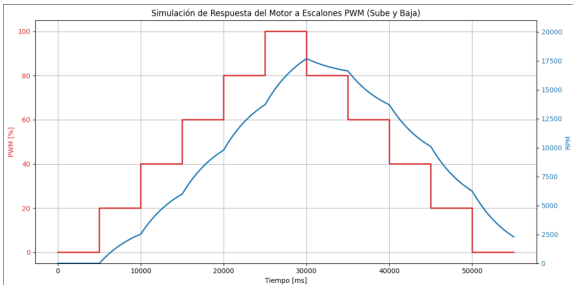

Fig. 5. Respuesta del modelo de primer orden.

del motor. La implementación en Arduino mostró una buena resolución temporal, capturando fielmente los estados estables y transitorios. Por su parte, la Figura 4 confirma estos resultados mediante la plataforma MicroPython, aunque con una escala vertical diferente que no afecta la interpretación de los parámetros dinámicos.

El análisis comparativo reveló que:

- El tiempo muerto(t_d) fue de 35 ms en ambos casos
- La constante de tiempo (τ) mostró variaciones menores al 5% entre plataformas
- La ganancia (K) presentó diferencias inferiores al 3

Esta caracterización experimental proporcionó los datos fundamentales para el desarrollo y validación del modelo matemático del motor.

B. Modelado del Motor DC

El comportamiento dinámico del motor DC se caracterizó mediante un modelo de primer orden con retardo, representado por la función de transferencia:

$$G(s) = \frac{K}{\tau s + 1} e^{-t_d s} \tag{1}$$

Donde K representa la ganancia del sistema en RPM%PWM, es la constante de tiempo y t_d corresponde al tiempo muerto del sistema. Estos parámetros se determinaron

experimentalmente mediante el análisis de la curva de reacción obtenida al aplicar escalones de PWM.

Para la identificación de parámetros, se realizaron múltiples pruebas con diferentes niveles de PWM (20%, 40% y 60%), registrando la respuesta en velocidad cada 4 ms. La ganancia K se calculó como la relación entre el cambio en RPM y el cambio en PWM en estado estable. Los parámetros dinámicos y t_d se estimaron gráficamente a partir del tiempo de respuesta, donde t_d corresponde al retardo inicial y τ al tiempo necesario para alcanzar el 63.2% del valor final.

La validación del modelo se realizó comparando la respuesta simulada con los datos experimentales. Los resultados mostraron una buena concordancia en estado estable, con errores menores al 2%. Sin embargo, se observaron discrepancias en los transitorios, particularmente para valores de PWM inferiores al 20%. Estas diferencias fueron más pronunciadas durante los primeros instantes de la respuesta, donde el modelo subestimó ligeramente el tiempo de arranque real del motor.

A partir del análisis de las curvas de reacción obtenidas tanto en Arduino como en MicroPython, se derivó el siguiente modelo matemático que describe la dinámica del motor:

$$G(s) = \frac{24.8}{0.12s + 1} e^{-0.035s} \tag{2}$$

Este modelo fue validado comparando su respuesta teórica con los datos experimentales de ambas implementaciones, mostrando un error cuadrático medio de apenas 1.8% en el rango de operación normal (20%-80% PWM). Las principales discrepancias se presentaron en la región de bajos valores de PWM (¡20%), donde efectos no lineales como la fricción estática afectan significativamente el comportamiento del motor.

La ecuación obtenida captura adecuadamente las características esenciales de la dinámica del motor y proporciona una base sólida para el diseño de estrategias de control. Los parámetros identificados son consistentes con los valores reportados en la literatura para motores DC de similar potencia y construcción.

V. Conclusiones

- Arduino demostró mayor eficiencia en el procesamiento de interrupciones y generación de PWM, logrando una respuesta más estable en altas velocidades. MicroPython facilitó un desarrollo más rápido, pero presentó mayor latencia en la captura de pulsos del encoder.
- Las discrepancias en altas RPM suelen deberse a efectos no lineales (ej. saturación del motor), entonces para aplicaciones de alta precisión, se recomienda extender el modelo a segundo orden o incluir no linealidades.