Enfoque multitareas implementado en una minicomputadora para el control y monitoreo de un motor a pasos industrial

Implementation of multitasking approach on a minicomputer for the control and monitoring of an industrial stepper motor

Omar Trejoluna Hernández
Universidad de Guanajuato, Salamanca, Guanajuato, México
o.trejolunahernandez@ugto.mx

Felipe de Jesús Torres del Carmen
Universidad de Guanajuato, Salamanca, Guanajuato, México
fdj.torres@ugto.mx
ORCID: 0000-0001-5792-2098

Antonio de Jesús Balvantín García
Universidad de Guanajuato, Salamanca, Guanajuato, México
antonio.balvantin@ugto.mx

Miroslava Cano Lara
TecNM/Instituto Tecnológico Superior de Irapuato, Irapuato, Guanajuato, México
miroslava.cl@irapuato.tecnm.mx

Diego Alfredo Núñez Altamirano
Universidad de Guanajuato, Salamanca, Guanajuato, México
da.nunez@ugto.mx

María Concepción Alvarado Méndez
TecNM/ Instituto Tecnológico Superior de San Martín Texmelucan, Puebla, México
mariaam@smartin.tecnm.mx

doi: https://doi.org/10.36825/RITI.12.27.002

Recibido: Junio 22, 2024
Aceptado: Agosto 17, 2024

Resumen: Este artículo presenta la implementación del enfoque multitareas sobre una minicomputadora de placa única Raspberry Pi, con resultados experimentales, del control y monitoreo de un motor a pasos Nema 34 de tipo industrial a través de una interfaz de usuario que permite la configuración de la operación de control y visualización de datos para interactuar con los dispositivos físicos. Una de las tareas consiste en generar una modulación del ancho de pulso que se envía al *driver* DQ860HA para hacer girar al motor a pasos considerando 400 pulsos por revolución, mientras que, la segunda tarea realiza, de manera simultánea, un conteo del número de pulsos que produce un *encoder* que tiene una resolución de 100 pulsos por revolución, el cual es acoplado al eje del motor. El uso de equipo de cómputo de bajo costo y software *open-source* hace que el desarrollo presentado pueda ser

considerado como una aplicación de tecnología frugal para el control y monitoreo de un sistema industrial, con resultados experimentales que validan el correcto funcionamiento de dos tareas que deben ser ejecutadas de manera simultánea.

Palabras clave: *Multitareas, Minicomputadora, Motor a Pasos, Control, Monitoreo.*

Abstract: This paper presents the implementation of the multitasking approach on a Raspberry Pi single board minicomputer, with experimental results, of the control and monitoring of an industrial type Nema 34 stepper motor through a user interface that allows the configuration of the control operation and data visualization to interact with the physical devices. One of the tasks consists of generating a pulse width modulation that is sent to the DQ860HA driver to rotate the stepper motor considering 400 pulses per revolution, while the second task simultaneously counts the number of pulses produced by an encoder with a resolution of 100 pulses per revolution, which is coupled to the motor shaft. The use of low-cost computer equipment and open-source software makes the presented development can be considered as a frugal technology application for the control and monitoring of an industrial system, with experimental results that validate the correct operation of two tasks that must be executed simultaneously.

Keywords: *Multitasking, Minicomputer, Stepper Motor, Control, Monitoring.*

1. Introducción

En las aplicaciones de desarrollo tecnológico, comúnmente se hace uso de equipo con gran capacidad de software y hardware para llevar a cabo distintas implementaciones de control, monitoreo e industria 4.0 en sistemas mecatrónicos como robots manipuladores, motores eléctricos, sistemas electrohidráulicos, etc. Así, para un sistema de control industrial en lazo cerrado o bien, para propósitos de implementar un Sistema de Control y Adquisición de Datos (SCADA, por sus siglas en inglés), estas dos acciones del control industrial (control y monitoreo) deben realizarse de manera simultánea o paralelamente. Esto es posible usando un esquema de multitareas, el cual comúnmente es implementado en equipos de cómputo de gran capacidad de hardware y software, particularmente software bajo licencia. Sin embargo, existe una tendencia global hacia el uso de tecnología de código abierto (*open-source*) y bajo costo, la cual incluye software y hardware. La importancia de este enfoque se analiza desde el ámbito de la innovación frugal, entendida como "hacer más con menos". Así, la tecnología *open*-source ha sido ampliamente usada en aplicaciones básicas de tipo académicas, pero existe una brecha aún en aplicaciones de tipo industrial, lo cual es motivo de discusión entre la comunidad tecnológica ya que falta se demuestre la viabilidad de esta tecnología.

Bajo el enfoque de tecnología frugal, se ha utilizado minicomputadora Raspberry Pi (RPi) para realizar solo una de las tareas a la vez que requiere un proceso industrial. Así, para realizar la tarea del control de un motor a pasos, es necesario programar en la minicomputadora el algoritmo que permita la manipulación del ancho de pulso que hará girar al motor. Por ejemplo, en [1] se describe el uso del *driver* TB6600 para controlar un motor a pasos Nema 23 en una máquina CNC a través de una RPi y una interfaz Mach. En [2] se propone un sistema de apoyo al aprendizaje basado en dispositivos móviles y de bajo costo para un motor a pasos que no es de tipo industrial, donde el sistema completo es concebido como un laboratorio remoto basado en la web utilizando una interfaz de usuario amigable; en la que se hace uso de dispositivos de bajo costo como RPi, dos plataformas de Arduino, controlador de medio puente SN754410 y lenguaje de programación Node.js. En [3] se presenta una interfaz de usuario que puede controlar en lazo abierto a motores a pasos 28 BYJ-48 que no son de tipo industrial, donde el usuario puede indicar el número de pasos, la dirección y la velocidad; se ha utilizado una RPi y circuitos integrados ULN2003A como drivers de los motores a pasos.

La otra de las tareas necesarias del control industrial es el monitoreo de las variables de interés. Para lograrlo, se hace uso de sensores y elementos de adquisición de datos. En relación con el motor a pasos, el desplazamiento angular del eje del motor es la variable importante por monitorear. Para tal efecto se emplea algún dispositivo como el encoder incremental para conocer el giro realizado por el motor, el cual genera un tren de pulsos a partir del desplazamiento angular del eje del motor [4, 5].

Por tanto, las dos tareas de control y monitoreo para un motor a pasos deben realizarse de manera simultánea debido al requerimiento de enviar una señal de pulsos al driver que hace girar al motor, mientras se está analizando

la señal de pulsos que genera el *encoder*. En [6] se diseñó contenido de laboratorio experimental de un motor a pasos en el cual se le adaptó un sensor de tipo fotointerruptor en el husillo que está acoplado al motor y se usó una barra ranurada para generar pulsos durante el movimiento del recorrido. Sin embargo, en este caso se llevó a cabo ambas tareas (control y monitoreo) de manera simultánea a través de una computadora personal y para el monitoreo no se utilizó un *encoder* y en su lugar fue usado un fotointerruptor, el cual no condiciona acciones a tomar en cuenta como la frecuencia del pulso, la resolución del *encoder*, un circuito de filtro para evitar la señal de rebote, entre otros.

Dentro del contexto del enfoque de multitareas realizado en una minicomputadora de bajo costo, en [7] se integra una RPi, programada con código abierto, un espectrómetro, una fuente de alimentación CD de alto voltaje y una prueba de detección de plasma para desarrollar un sistema de monitoreo multitarea de elementos metálicos en solución. Se hace notar que la RPi genera una señal de modulación del ancho de pulso para controlar los pulsos de voltaje enviados de la prueba de plasma y, la misma RPi recibe los datos del espectrómetro a través de comunicación del bus universal serial (USB), por lo que el algoritmo de programación es secuencial y no necesariamente requiere de un enfoque de multitareas. En [8] se utiliza el enfoque multitareas para el procesamiento paralelo de imágenes, sin embargo, no se trata de una aplicación industrial.

El objetivo de este artículo consiste en implementar experimentalmente el enfoque multitareas para el control y monitoreo de un motor a pasos industrial a través de la programación en código abierto sobre una minicomputadora Raspberry Pi que, además, ejecuta acciones de interacción con el usuario por medio de una interfaz que permite la configuración del control del motor a pasos y la visualización de los datos generados por el *encoder*.

El resto del artículo se organiza de la siguiente manera: en la sección 2 se detallan las características del hardware y software *open-source*, en la sección 3 se describe la conexión de los componentes y el algoritmo para el enfoque multitareas, en la sección 4 se muestra la metodología para la implementación, en la sección 5 se presentan los resultados experimentales obtenidos, y en la sección 6 se dan las conclusiones del trabajo realizado.

2. Detalles de hardware y software open-source

2.1. Motor a pasos Nema 34

Esta máquina eléctrica es llamada motor a pasos porque desarrolla un pequeño giro, conocido como paso, cada vez que recibe un pulso eléctrico de un controlador o comúnmente llamado *driver*. En este trabajo se utiliza un motor a pasos industrial, como se muestra en la Figura 1, de doble eje, Nema 34 (dimensión de la placa frontal de 3.4” x 3.4”), ángulo de paso de 1.8 °, corriente por fase de 3.5 A, par motor de 115 kg/cm.

Figura 1. Motor a pasos Nema 34 de tipo industrial.

2.2. Driver DQ860HA

Todo motor a pasos requiere de un driver para su funcionamiento, particularmente en este trabajo se utiliza el DQ860HA, mostrado en la Figura 2, que es un controlador de motor a pasos híbrido de dos fases, diseñado para obtener el máximo rendimiento del motor. Este controlador se puede utilizar con motores que requieren una corriente de 2.1 a 7.2 A y cuenta con micropasos en un rango de 400 a 51200. Permite un voltaje de entrada de 24-110 VCD, y genera una señal de pulso dentro de una frecuencia de 0 a 100 kHz.

Figura 2. Driver DQ860HA de tipo industrial.

Los bornes A+, A-, B+ y B- se conectan con el motor a pasos, los bornes DIR+, DIR-, PUL+ y PUL- son conectados a la señal de control que, en este trabajo, proviene de una minicomputadora.

2.3. Encoder 38S6G6-B-G24N

Se trata de un codificador rotatorio incremental que determina el sentido de giro de un movimiento y el desplazamiento angular recorrido en ese movimiento, por tanto, este dispositivo puede determinar el ángulo de posición del eje de rotación a partir de realizar cuentas incrementales, donde cada posición es completamente única. Particularmente el *encoder* 38S6G6-B-G24N (Fig. 3) tiene una resolución de 100 pulsos por revolución, es alimentado de 5 a 24 V CD, su eje tiene un diámetro de 6 mm y genera dos señales de pulsos, A y B.

Figura 3. *Encoder* rotatorio incremental 38S6G6-B-G24N.

2.4. Raspberry Pi 3B+

La Raspberry Pi modelo 3B+ de la Figura 4, es una placa de desarrollo definida como una computadora en placa única, la cual es concebida como una minicomputadora por su tamaño similar al de una tarjeta de crédito, que soporta sistema operativo basado en Linux, particularmente Raspbian que es el recomendado por la fundación Raspberry Pi. La manipulación de sus 40 pines de propósito general de entrada/salida (GPIO) permite interactuar con el exterior a través de su procesador Broadcom BCM2837B0, Cortex-A53 (ARMv8) de 64-bit SoC a una frecuencia de reloj de 1.4 GHz y 1 GB de memoria RAM, para implementar acciones de control y monitoreo industrial bajo el enfoque de multitareas.

Figura 4. Raspberry Pi modelo 3B+.

2.5. Python

El lenguaje de programación Python es considerado como un lenguaje orientado a objetos que está teniendo gran auge en todo el mundo por ser pragmático en la estructura de la sintaxis utilizada. El sistema operativo Raspbian tiene precargadas plataformas de programación para código en Python, por lo que es un lenguaje que puede manipular los recursos de hardware de la Raspberry Pi.

2.6. Enfoque multitareas

El enfoque multitareas se ha convertido en uno de los temas más importantes en los sistemas basados en microprocesadores, concretamente en las aplicaciones de automatización. A medida que aumenta la complejidad de los proyectos, se les exige una mayor funcionalidad y dichos proyectos requieren el uso de varias tareas interrelacionadas que se ejecutan en el mismo sistema y comparten la CPU (o varias CPU) para implementar las operaciones necesarias de manera prácticamente simultánea o en paralelo. Como consecuencia de ello, la importancia del funcionamiento multitareas en las aplicaciones basadas en microprocesadores no ha dejado de crecer en los últimos años. En la actualidad, muchos proyectos de automatización complejos utilizan algún tipo de núcleo multitareas [9].

3. Conexión entre componentes y algoritmo de programación

3.1. Diagrama de conexión

Los dispositivos físicos que se han descrito anteriormente deben ser conectados entre sí de tal manera que la Raspberry Pi realice las tareas del control del motor a pasos a través del driver DQ860HA y el monitoreo de la rotación angular del motor por medio del *encoder* incremental; ambas, de manera simultánea. Por lo que el circuito de conexión es el presentado en la Figura 5.

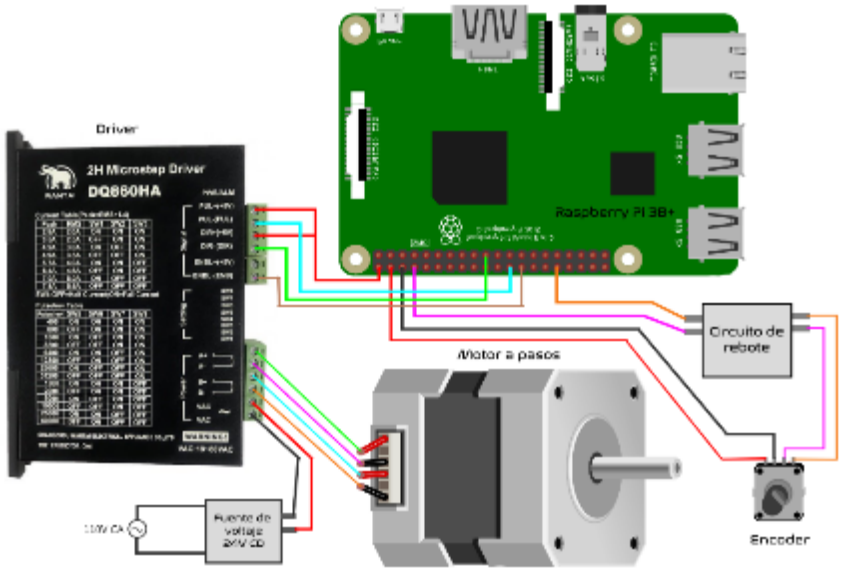

Figura 5. Diagrama de conexión de los dispositivos físicos.

Es importante destacar que el *encoder* genera una señal de pulsos que, debido a la frecuencia del cambio de la señal analógica de 0 V CD a 5 V CD, existe un fenómeno conocido como rebote, el cual es mostrado en la Figura 6 y se requiere de un circuito electrónico que es usado para evitar el rebote de la señal antes de ser analizada por la Raspberry Pi debido a que el efecto causado por el rebote hace que el conteo de los pulsos sea incorrecto y no corresponda a la resolución del *encoder*.

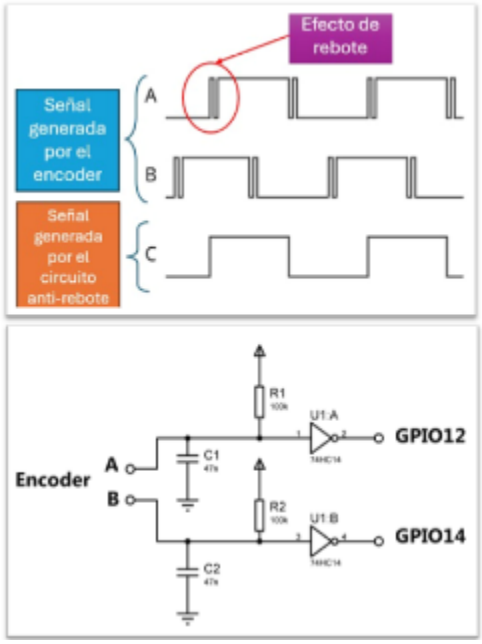

Figura 6. Señales del *encoder* con rebote y circuito electrónico anti-rebote.

3.2. Algoritmo de programación

El proceso industrial que se ha implementado requiere de dos subtareas de programación que deben ejecutarse de manera simultánea por medio de código en lenguaje Python sobre una Raspberry Pi. Una subtarea consiste en el control de la puesta en marcha del motor a pasos, el cual es definido por el usuario a través de una interfaz que permite la interacción con el dispositivo. La otra subtarea está relacionada con el *encoder*, en el cual el código de programación contará los pulsos generados por el *encoder* y mostrará el conteo en la pantalla de la interfaz con el usuario. Es importante notar que estas subtareas no pueden ejecutarse de manera secuencial debido a las características descritas a continuación.

3.2.1 Subtarea del control del motor a pasos

Para la programación de esta subtarea se identificaron las características técnicas del motor a pasos y del *driver* DQ860HA que se relacionen con modalidades de operación que el usuario pueda seleccionar y así, tener la interacción deseada. En la Figura 7 se muestra la secuencia de señal de control por medio de pulsos que la Raspberry Pi debe enviar al *driver* DQ860HA para que este a su vez, pueda hacer girar al motor a pasos.

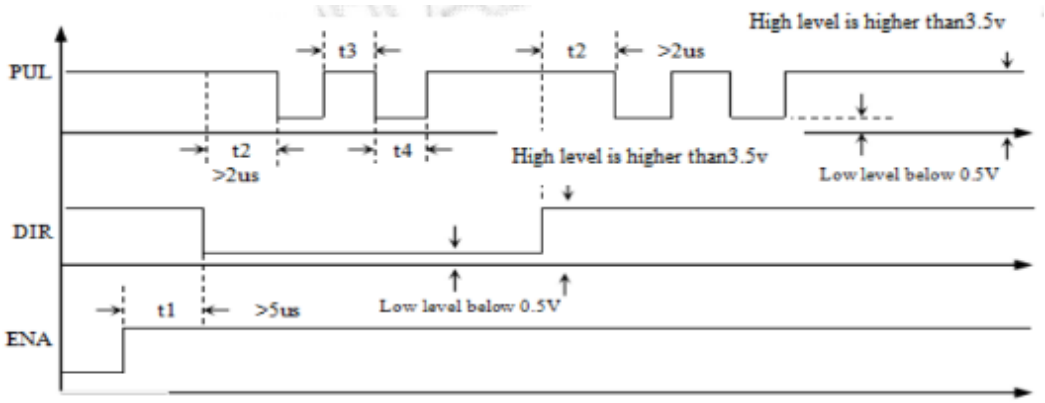

Figura. 7. Diagrama de secuencia de señal de control que recibe el driver desde la Raspberry Pi.

Como puede observarse, la Raspberry Pi se conecta con el driver DQ860HA por medio de 2 pines configurados como salida de la RPi para enviar las señales DIR y PUL. Así, debe enviarse la señal DIR para indicar el sentido de giro (horario o antihorario), después se inicia el envío de la señal de pulsos PUL, la cual es generada por la

Raspberry Pi ejecutando un ciclo que envía un valor alto (5 V CD) durante cierto tiempo y luego envía un valor bajo (0 V CD) durante otro tiempo. Cada vez que el *driver* DQ860HA recibe un pulso, hace girar al motor a pasos y se requiere de 400 pulsos para dar una vuelta completa. De acuerdo con la frecuencia máxima del pulso generado por el driver de 100 kHz y, además, se requiere un mínimo de 400 pulsos para que el motor a pasos gire 1 revolución completa, se puede calcular que la velocidad máxima que alcanzaría el motor para cualquiera de los micropasos que sean especificados en el driver, es de 150 revoluciones por minuto (rpm). Así, el pseudocódigo implementado en la Raspberry Pi es el siguiente:

1. Importar la librería *RPi.GPIO* para controlar los pines *GPIO* de la Raspberry Pi.
2. Importar la librería time para gestionar la modulación por ancho de pulso.
3. Desactivar las advertencias de *GPIO* para evitar mensajes no deseados.
4. Definir las constantes *DIR* (pin de dirección), *PUL* (pin de pulsos), *CW* (sentido horario) y *CCW* (sentido antihorario).
5. Configurar el modo de los pines *GPIO* a *BCM*, establecer *GPIO8* para el pin *DIR* y *GPIO10* para el pin *PUL*.
6. Iniciar un bucle:
    - Declarar las variables *spin* (sentido de giro), *rev* (rpm del motor), *puls* (cantidad de pulsos a mandar por el pin *GPIO10*), *sel* (selección entre cantidad de revoluciones o de pulsos). El usuario introduce los valores que desee a estas variables a través de la interfaz de usuario de la siguiente manera.
    - Solicitar al usuario la velocidad angular deseada ω para el motor (en un rango permitido de 0-150 rpm).
    - Verificar si la velocidad ingresada está dentro del rango permitido, de no ser así se mostrará un mensaje de error.
    - Solicitar al usuario el sentido de giro del motor verificando que elija únicamente una de las dos opciones (horario o antihorario).
    - Configurar el sentido de giro del motor según la selección del usuario.
    - Solicitar al usuario si desea especificar el número de revoluciones o de pulsos verificando que elija únicamente una de las dos opciones.
    - Calcular el período *T* de la señal de pulsos en función de la velocidad ingresada $T = \frac{3}{20 \cdot \omega}$.
    - Si se elige especificar el número de revoluciones:
        - Solicitar al usuario el número de revoluciones deseadas validando que sea un valor mayor a cero.
        - Calcular el número de pulsos requeridos para las revoluciones ingresadas (400 pulsos por revolución).
        - Generar los pulsos necesarios para completar las revoluciones.
    - Configurar el pin de pulsos *PUL* como salida en alto para iniciar el pulso.
    - Esperar la mitad del periodo $(T/2)$ para mantener el estado alto.
    - Enviar en el pin de pulsos *PUL* un valor de bajo para finalizar el pulso.
    - Esperar la mitad del periodo $(T/2)$ para mantener el estado bajo.
    - Si se elige especificar el número de pulsos:
        - Solicitar al usuario el número de pulsos deseados validando que sea un valor mayor a cero.
        - Generar los pulsos determinados por el usuario.

3.2.2 Subtarea del conteo de pulsos del *encoder*

El *encoder* se acopla de manera directa con el eje del motor a pasos, por lo cual, el *encoder* detecta el giro que desarrolla el motor. La resolución del *encoder* es de 100 pulsos por revolución, esto quiere decir que el *encoder* genera una señal de pulsos, similar a la señal de pulsos que es enviada al driver DQ860HA, pero la señal de pulsos del *encoder* es una señal de entrada para la Raspberry Pi, por lo que el algoritmo de programación debe estar revisando el pin de la RPi que recibe la señal generada por el *encoder* para determinar si existe un flanco de subida, bajada o ambos; con la finalidad de identificar la existencia de un pulso. Así, cada vez que se detecta un pulso, se

incrementa un contador y se manda visualizar en la pantalla el valor del contador. El pseudocódigo para el control del *encoder* es dado así:

1. Importar la librería RPi.GPIO para controlar los pines GPIO de la Raspberry Pi.
2. Desactivar las advertencias de GPIO para evitar mensajes no deseados.
3. Inicializar variables cont y cnt para contar pulsos y revoluciones respectivamente.
4. Configurar el modo de los pines GPIO a BCM y establecer los pines GPIO12 y GPIO14 como entrada con una resistencia pull-up, para las señales A y B desde el *encoder*, respectivamente.
5. Iniciar un bucle:
    - Esperar a que se detecte un cambio en el pin 12 (flanco de subida y bajada).
    - Incrementar el contador de pulsos (*cont*).
    - Imprimir el número actual de pulsos.
    - Si el contador de pulsos alcanza el valor de 100 (revolución completa del *encoder*):
        - Incrementar el contador de revoluciones *(cnt)*.
        - Reiniciar el contador de pulsos a 0.
        - Imprimir el número actual de revoluciones.

3.2.3 Algoritmo para el proceso de multitareas

Una vez realizadas las subtareas de control y *encoder* por separado, se llevó a cabo la implementación del pseudocódigo que permite la ejecución simultánea de ambas subtareas, de la siguiente manera:

1. Importar las librerías necesarias: RPi.GPIO para controlar los pines GPIO, _thread para manejar los multiprocesos y time para gestionar los tiempos de espera y generar la modulación del ancho de pulso.
2. Desactivar las advertencias de GPIO para evitar mensajes no deseados.
3. Definir las constantes DIR (pin de dirección), PUL (pin de pulsos), CW (sentido horario) y CCW (sentido antihorario).
4. Configurar el modo de los pines GPIO.
5. Iniciar un bucle para el control del motor:
    - Declarar las variables *spin, rev, puls, sel, cont* y *cnt*.
    - Solicitar al usuario la velocidad deseada para el motor (0-150 rpm).
    - Verificar si la velocidad ingresada está dentro del rango permitido, de no ser así, mostrar un mensaje de error.
    - Solicitar al usuario el sentido de giro del motor (horario o antihorario).
    - Configurar el sentido de giro del motor según la selección del usuario.
    - Solicitar al usuario si desea especificar el número de revoluciones o de pulsos.
    - Calcular el período (*T*) en función de la velocidad ingresada.
    - Si se elige especificar el número de revoluciones:
        - Solicitar al usuario el número de revoluciones deseadas.
        - Calcular el número de pulsos requeridos para las revoluciones ingresadas.
        - Establecer una variable bandera con un estado *False* para que inicie el ciclo de la función de control del *encoder*.
        - Iniciar un nuevo multiproceso con ayuda de *Thread* para la función del *encoder "control_detect()"*.
        - Generar los pulsos necesarios para completar las revoluciones.
        - Cambiar el estado de la bandera (a un valor *True*) para finalizar el ciclo de la función *control_detect()*.
    - Si se elige especificar el número de pulsos:
        - Solicitar al usuario el número de pulsos deseados.
        - Establecer la bandera con un estado *False*.
        - Iniciar un nuevo multiproceso para la función *control_detect()*.
        - Generar los pulsos determinados por el usuario.
        - Cambiar el estado de la bandera (a un valor *True*) para finalizar el ciclo de la función *control_detect()*.
6. Definir la función control_detect() para el control del *encoder*.

- Declarar las variables globales *cont, cnt* y *flag* dentro de la función.
- Iniciar un bucle que depende del estado de la bandera (*flag*).
- Esperar a que se detecte un cambio en el pin 23 (flanco de subida y bajada).
- Incrementar el contador de pulsos *(cont)*.
- Imprimir el número actual de pulsos.
- Si el contador de pulsos alcanza 100 (revolución completa del *encoder*).
- Incrementar el contador de revoluciones *(cnt)*.
- Reiniciar el contador de pulsos a 0.
- Imprimir el número actual de revoluciones.

4. Metodología para la implementación física

Los elementos que se han descrito en la sección 2 deben ser adquiridos e interconectados siguiendo el diagrama presentado en la Figura 5. En este caso, el motor a pasos puede ser de algún otro número de marco, sin embargo, debe cumplir con el criterio de ser un motor a pasos de tipo industrial. Respecto a la minicomputadora Raspberry Pi, se detalló un modelo 3B+, el cual es el usado para mostrar resultados experimentales, sin que se limite el uso de otros modelos más recientes como el 4B+ o 5B+. Por su parte, el *encoder* incremental de 100 pulsos por revolución, puede ser sustituido por algún otro *encoder* con mayor o menor número de pulsos por revolución, si así fuera el caso se tendría que modificar la condición puesta en el algoritmo 5.4 para completar una revolución completa.

Para la codificación de los algoritmos de puesta en marcha y monitoreo del motor a pasos, así como del enfoque de multitareas y la creación de la interfaz gráfica de usuario, se ha utilizado Python que se ejecuta de manera correcta en una minicomputadora Raspberry Pi. Se considera importante realizar subprogramas de código que permitan identificar posibles fallas. Es decir, en una primera etapa se estaría programando solo la puesta en marcha del motor a pasos en donde se posibilite la configuración del comportamiento del motor. Posteriormente se estaría codificando la sección del algoritmo que detecte el movimiento rotacional del motor a través del encoder incremental. Por último, se llevaría a cabo la unión de ambos códigos por medio del enfoque de multitareas para que se considere un esquema de control y monitoreo de motor a pasos de manera simultánea, con la habilitación de la interacción con el usuario.

5. Resultados experimentales

En la Figura 8 se presenta la plataforma experimental que se construyó a partir de la interconexión física de los componentes: motor a pasos, *encoder* incremental, minicomputadora Raspberry Pi y circuito de eliminación de rebote.

Figura 8. Plataforma experimental de esquema multitareas para control y monitoreo de motor a pasos.

La plataforma experimental interactúa con el usuario a través del monitor como medio de visualización y del teclado para introducir los valores deseados o, la selección que el usuario considere. Por ejemplo, en la captura de pantalla de una prueba experimental que se muestra en la Figura 9, se identifican los parámetros de configuración que el usuario puede introducir o seleccionar para la puesta en marcha del motor a pasos. Así, se describe que el usuario ha realizado lo siguiente:

- En la primera línea de texto del acercamiento del recuadro en rojo, se aprecia la leyenda:
  Velocidad (0-150 rpm): 18
  Lo cual significa que el usuario ha establecido que el motor a pasos Nema 34 rote a 18 rpm.
- Posteriormente, en ese mismo acercamiento del recuadro en rojo, se observan los textos:
  Sentido de giro
  1. Horario
  2. Antihorario
  Selección: 1
  Por lo que el usuario ha seleccionado un sentido de giro horario.
- Luego se selecciona la modalidad de la puesta en marcha del motor a pasos para que sea a través del número de revoluciones, con las expresiones mostradas como:
  Menú
  1. Revoluciones
  2. Pulsos
  Selección: 1
- Por último, el usuario ha indicado que desea que el motor gire 1 revolución a través de la etiqueta:
  # Revoluciones: 1

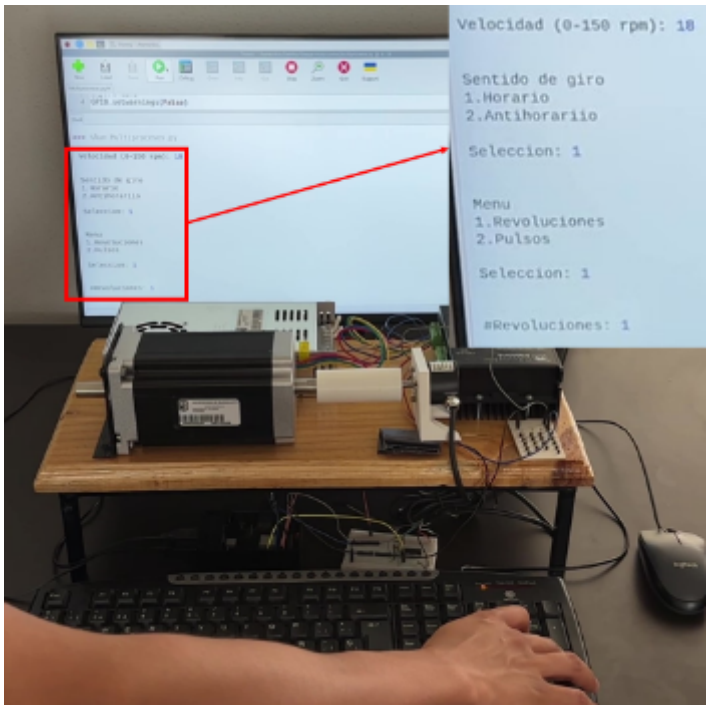

Figura 9. Interfaz de usuario para interactuar con el equipo físico en el control del motor a pasos.

Una vez que el usuario ha terminado la configuración del control del motor a pasos, la Raspberry Pi envía la señal de pulsos al *driver* DQ860HA para que éste a su vez provoque el giro deseado en el motor. De manera paralela, mientras está siendo enviada la señal de pulsos al *driver* desde la Raspberry Pi, el *encoder* genera una señal de pulsos que es recibida por la propia Raspberry Pi, es en este momento que la minicomputadora debe ejecutar simultáneamente ambas subtareas y, además visualizar en la pantalla, durante el giro del motor, el número de pulsos que está detectando el *encoder* para dar evidencia de la acción de monitoreo.

Esta condición se muestra en la Figura 10 en donde se observa la impresión en pantalla de una secuencia numérica, identificándose en el acercamiento del recuadro en rojo que se está visualizando el número 80, 81, 82,

etc. hasta el número 100. Estas cantidades representan el conteo de los pulsos generados por el encoder dentro de la subtarea de monitoreo.

Por último, se muestra al final del acercamiento que en la pantalla aparece un mensaje de texto con la leyenda "1 revoluciones", el cual se genera cuando se termina el giro del motor.

Para la prueba de funcionamiento que se está describiendo, el número de revoluciones dadas por el motor físicamente coincide con la configuración que el usuario realizó y, además, con el conteo de los pulsos recibidos por el encoder que tiene una resolución de 100 pulsos por revolución. Esto implica que es posible comprobar el correcto funcionamiento del enfoque multitareas implementado en una minicomputadora para un esquema de control y monitoreo de tipo industrial.

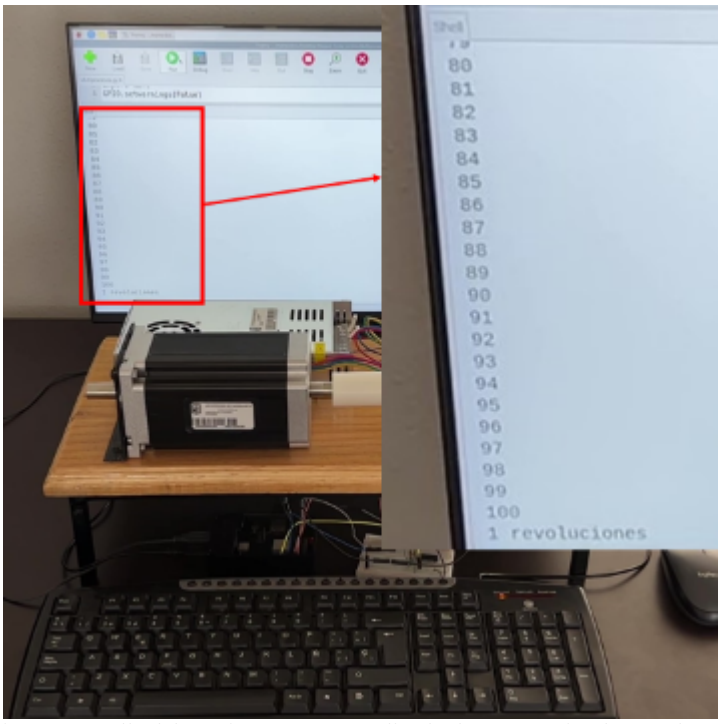

Figura. 10. Visualización en pantalla del monitoreo que es realizado de manera simultánea al control del motor a pasos.

El video completo de la prueba de funcionamiento puede ser consultado en el *link*:
https://drive.google.com/drive/folders/18afKTXZIP94GB-iPtUvR3ch6iLmlj-kQ

6. Conclusiones

El enfoque multitareas ha sido implementado en una minicomputadora de bajo costo, Raspberry Pi, con lenguaje de programación de código abierto Python para realizar simultáneamente las tareas de control y monitoreo de un motor a pasos Nema 34 de tipo industrial. En donde se han obtenido resultados experimentales que validan el correcto funcionamiento del esquema propuesto como parte del uso de tecnología frugal para aplicaciones industriales, debido a que el funcionamiento físico del motor coincide con la configuración del control dado por el usuario y con el número de pulsos que debe ser detectado por el *encoder*.

En ambas tareas que se ejecutan de manera simultánea desde la Raspberry Pi, se trata de una señal de pulsos, donde para la acción del control, es una señal de salida que genera la minicomputadora, mientras que, en la acción del monitoreo, es una señal de entrada que es analizada por la Raspberry Pi.

Además, se ha incorporado una interfaz de usuario mediante la cual se interactúa con los dispositivos físicos que permite la configuración del control del motor a pasos y la visualización del monitoreo del giro realizado.

En trabajos futuros se implementarán herramientas tecnológicas del internet industrial de las cosas para que el enfoque multitareas incluya la conectividad a la nube y se logre un control y monitoreo remoto en tiempo real de un motor a pasos, basado en la programación de una minicomputadora Raspberry Pi, lo cual incluirá el desafío del uso del enfoque multitareas para acciones que complementen el trabajo realizado, como establecer la conexión a internet, configurar la propia Raspberry Pi como server socket para enviar y recibir datos.

8. Agradecimientos

Los autores desean agradecer el apoyo brindado por la Universidad de Guanajuato a través de la Convocatoria Institucional de Investigación Científica CIIC 2024 para la realización de este trabajo.

9. Referencias

[1] Khairudin, M., Asnawi, R., Shah, A. (2020). The characteristics of TB6600 motor driver in producing optimal movement for the Nema23 stepper motor on CNC machine. *Telkomnika*, *18* (1), 343-350. http://doi.org/10.12928/telkomnika.v18i1.12781

[2] Fukumoto, H., Yamaguchi, T., Ishibashi, M., Furukawa, T. (2020). Developing a remote laboratory system of stepper motor for learning support. *IEEE Transactions on Education*, *64* (3), 292-298. http://doi.org/10.1109/TE.2020.3042595

[3] Yılmazlar, E., Kuşçu, H., Erdemir, V., Güllü, A. (2018). Design Of Stepper Motor Control Interface With Embedded Systems. *International Journal of Engineering Research and Development, 14* (6), 17-22.

[4] Dominik, K., Kunović, I., Matić, J., Sovic Krzic, A. (2023). Ball Detection Using Deep Learning Implemented on an Educational Robot Based on Raspberry Pi. *Sensors*, *23* (8), 1-21. https://doi.org/10.3390/s23084071

[5] Karahan, O., Hökelek, H. (2020). Mobile Robot Position Controlling System Based On IoT Through Raspberry Pi, *Journal of Intelligent Systems: Theory and Applications*, *3* (2), 25-30. https://doi.org/10.38016/jista.652908

[6] Habib, M., Nagata, F., Watanabe, K. (2021). Mechatronics: Experiential learning and the stimulation of thinking skills. *Education Sciences*, *11* (2), 1-22. https://doi.org/10.3390/educsci11020046

[7] Chang, T., Wang, C., Hsu, C. (2024). Development of a real-time and multitasking system for long-term monitoring of aqueous metallic elements using plasma spectroscopy. *Talanta*, *271*, 1-9. https://doi.org/10.1016/j.talanta.2024.125688

[8] Hosny, K., Salah, A., Magdi, A. (2023). Parallel image processing applications using Raspberry Pi. En K. M. Hosny, A. Salah (Ed.) *Recent Advances in Computer Vision Applications Using Parallel Processing* (107-119). Springer. https://doi.org/10.1007/978-3-031-18735-3_6

[9] Ibrahim, D. (2020). *ARM-Based Microcontroller Multitasking Projects: Using the FreeRTOS Multitasking Kernel*. Newnes.

