IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 December 2023, accepted 12 December 2023, date of publication 15 December 2023,
date of current version 22 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3343638

RESEARCH ARTICLE

Implementation in Microcontrollers of an Algorithm for the Simple Generation of Speed Profiles in a Stepper Motor and Their Associated Kinematics

JORGE FONSECA-CAMPOS[1,2], ISRAEL REYES-RAMÍREZ[2], JUAN L. MATA-MACHUCA[2], LEONARDO FONSECA-RUIZ[2], PAOLA N. CORTEZ-HERRERA[2], LUIS BERNARDO FLORES-COTERA[1], AND RICARDO AGUILAR-LÓPEZ[1]

[1]Centro de Investigación y de Estudios Avanzados del IPN (CINVESTAV), Biotechnology and Bioengineering Department, Mexico City 07360, Mexico
[2]Instituto Politécnico Nacional (IPN), Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas (UPIITA), Mexico City 07340, Mexico

Corresponding author: Jorge Fonseca-Campos (jfonsecac@ipn.mx)

This work was supported by Secretaría de Investigación y Posgrado (SIP), Instituto Politécnico Nacional (IPN), under Grant 20231847.

ABSTRACT Electric motors are the most widely used actuators for converting electrical to mechanical energy. Stepper motors perform various automated tasks because their position control is accurate. The operation of these devices at high speeds reduces the time required to perform automated operations. However, this can create inconveniences such as loss or excess of steps, with a consequent positioning error. Controlling the motor using a speed profile is the most commonly employed technique for avoiding this problem. Its generation usually requires hardware with high computing power, such as digital signal processors or field programmable gate arrays, and algorithms requiring specifications not published in the motor datasheet. In this study, we implemented an algorithm to simplify the generation of speed profiles. In the acceleration and deceleration regions, the velocity follows a logarithmic, quasi-linear, quadratic, or sinusoidal pattern. The algorithm provides position and time arrays. The speed and acceleration can be calculated in terms of time using numerical derivatives. The simulation results were validated using two microcontrollers to control a stepper motor programmed on Arduino and Micropython platforms. Experimental measurements using an encoder, gyroscope, and accelerometer validated the kinematic response of the stepper motor. These values are in a good agreement with the simulation results.

INDEX TERMS Accelerometer, angular velocity control, gyroscope, kinematics, logarithmic speed profile, microcontroller, motion planning, residual vibration, speed profile, stepper motor.

I. INTRODUCTION

Motors have been one of the most widely employed electromechanical actuators for several years. The energetic transition from fossil fuels to renewable energies provides them with a relevant role in converting electric power into mechanical movement in the foreseeable future. Stepper motors are used extensively in applications requiring accurate position control, such as bioprinting [1], [2], [3], three–dimensional ultrasound imaging systems [4], and devices for automatic nucleic acid detection [5], among others. In these devices, the rotation of the shaft occurs discreetly as a digital signal called a step is applied [6]. The discrete angular displacement is a constant known as the step angle.

For several years, the unique characteristics of stepper motors have been identified, making them attractive for automation and control applications. For instance, they do not have mechanical brushes, have fewer mechanical problems, dissipate heat better than other motors, have a

The associate editor coordinating the review of this manuscript and approving it for publication was Yangmin Li.

© 2023 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

high torque-inertia ratio, and are usually inexpensive [7]. In addition, they have full torque at standstill if the coils are energized [8]. It is easy to control their position with high accuracy, and for this reason, they are traditionally used with open–loop control. However, at high speeds, stepper motors may lose synchronization in the classical open–loop control [9]. Also, changes in load torque might produce step loss in control without feedback [10]. Bangji and coworkers proposed an algorithm for optimizing the open–loop control of stepper motors [11]. A closed–loop control strategy that requires a field programmable gate array (FPGA) and digital signal processor (DSP) improves the motion of the linear translation stage. The most frequently employed open–loop control uses a programmable logic controller (PLC) or a computer to generate pulses [12]. Changes in the external load of the motor affects negatively its transient and steady response, by using a variable structure control (VSC) instead of the open–loop control this problem is mitigated [13].

The typical architecture for the operation of stepper motors consists of three blocks: controller, power driver, and stepper motor [6], [11]. This scheme can function with or without feedback. One of the most commonly employed strategies for operating a stepper motor consists of sending pulses with a PLC and performing other operations with a DSP. However, Isobe et al. used DSP for pulse generation and control of a stepper motor [14]. Microcontrollers are another option for pulse generation, and algorithms, including virtual ramps, have been used for this purpose [15]. The micro-stepping technique, in conjunction with an Arduino Mega and a L6470 driver, helped reduce jerk [16]. Other researchers have employed inexpensive microcontrollers such as PIC 16F887, to generate a trapezoidal speed profile [17]. Hardware with greater computing power as the FPGA helped develop a trapezoidal and parabolic speed profile for controlling a DC servo motor, finding that the parabolic profile had less energy consumption [18]. DSPs are another option for pulse generation [19]. Microcontrollers have been used in embedded systems for several years. They have hardware limitations, but nowadays, some can use 32 or 64-bit words and clocks of hundreds of MHz at an affordable cost [20]. FPGAs and DSPs offer greater computing power, but their prices are often much higher than those of microcontrollers, which are ideal for tasks that do not require much memory or extensive data processing [20].

Despite the hardware limitations of microcontrollers, these devices can perform more complex computations, such as neural networks [21]. Low and mid–level languages, such as C, C++, Rust, TinyGo, and Micropython are options for programming microcontrollers [22]. One of the most popular platforms for accomplishing this task is the Arduino IDE, which supports simplified versions of the C and C++ languages. Micropython is an interpreter for Python 3 microcontrollers, that inexperienced or beginner programmers often use because it is easier to develop programs [22]. Both programming platforms have been tested on several development boards, such as Espressif Systems ESP32 and STMicroelectronics' STM32, to evaluate their performance when subjected to the same tasks [21], [22], [23]. In the comparisons, the performance of the same microcontroller model was better for C than for Micropyhton. For example, when programming a neural network with 50 inputs, ten outputs, and 50 hidden nodes, the network execution period was almost 60 times longer in Micropython than in C [21]. Also, the execution times in Micropython compared to those of C language were much higher when a board ran the Fast Fourier Transform (FFT), Cyclic Redundancy Check (CRC), and Secure Hash Algorithm (SHA) algorithms [22].

The speed of the motor is one of the most relevant kinematic parameters because it defines the temporal duration of the movements. The higher the velocity, the less time it takes to perform the task. In stepper motors, the speed is proportional to the frequency and inversely proportional to the period of digital pulses or steps injected into the actuator [6], [8], [10], [12], [24]. Displacing the motor at the maximum speed can cause step loss and, of course, an error in angular positioning [10], [25], [26]. Additionally, the torque decreases as the velocity increases [27], [28], [29]. Moving the motor to its maximum speed requires a device with sufficient power to tolerate the acceleration impulses that occur at the beginning and end of the movement [30]. If exact positioning is the goal, a velocity profile should be used [11]. One of the most studied is the trapezoidal, which occurs in three stages. Initially, the angular velocity increases linearly until it reaches a maximum value, and in the intermediate stage, the speed remains constant. In the final part of the movement, the velocity decreases linearly until it reaches rest. A digital convolution technique can generate this profile [31]. Other researchers have used the pulse width modulation (PWM) technique in a microcontroller to create this velocity waveform [32]. Strategies for controlling the linear speed as cascaded proportional and proportional integral P-PI have been used [33]. Other authors have proposed a PI controller [34]. Optimization techniques are used as well with this profile for saving energy [35], and for reducing residual vibrations [36]. Control of the speed waveform of various stepper motor profiles requires an algorithm [37]. The acceleration in the first stage is a positive constant; later, it is zero. Finally, this variable is a negative constant. There are moments where the drastic change in acceleration in the trapezoidal profile generates a significant jerk with an impulse waveform, which produces unwanted vibrations [30], [38], [39], [40], [41], [42]. An alternative for reducing jerk is reached using an S-curve speed profile [10], [25], [26], [29], [30], [37], [38], [39], [42], [43] or patterns based on sinusoidal functions [30], [41]. Both profiles limit this variable, thereby reducing the residual vibrations. Despite the drawbacks of the trapezoidal speed profile, its use is widespread because it generates faster angular movements than other velocity patterns [26], [38].

Generating the speed profiles proposed in the literature requires parameters that are unknown in advance for the generic stepper motors. For example, in the work of Isobe and colleagues, the end user must input the acceleration and deceleration rates, which are not reported in most motor's datasheets [14]. Other researchers have highlighted the limitations of using ramp tables in which the timings for the actuator motion are precalculated [15]. To generate the S-curve profile, the maximum amplitudes of jerk, acceleration, and velocity must be known [41]. These values are not reported on the specifications of most motors. Therefore, a simplified algorithm for constructing velocity profiles that requires fewer unknown parameters is desirable.

Gyroscopes are devices mounted on a frame that allow the angular velocity to be measured when rotated [44]. Microelectromechanical system (MEMS) gyroscopes offer several advantages over other sensors used for the same purpose, highlighting their low cost, high accuracy, ease of integration, and the ability to be manufactured on a large scale [45], [46]. These devices have the potential to be applied in various fields, such as image stabilization [47], movement control of different robots [48], [49], [50], [51], navigation of unmanned aerial vehicles (UAV) [52], [53], detection of the state of movement of people [54], self-stabilization systems [55], [56], and locating the geographic north [57]. These sensors have been used for various purposes to measure the electrical motor parameters. For example, employing a MEMS circuit that detects movement in six axes of freedom with an accelerometer and a gyroscope model MPU-6050 of the TDK company Zhang et al. proposed a method for determining the rotor position of a permanent magnet synchronous motor (PMSM) [58]. Also, MEMS gyroscopes can help estimate the angular velocity of different motors [59], [60], [61].

Imbalances in the forces cause vibrations in the rotating systems. Sensing them is economical and indicates the mechanical condition of the systems [62], [63], [64]. It is possible to identify oscillations by measuring system displacement, velocity, or acceleration [64]. MEMS accelerometers are devices that have undergone several controlled mechanical tests to characterize their response when exposed to vibrations [63], [65], [66]; some of their attractiveness is related to their small dimensions and low cost compared to those based on the piezoelectric phenomenon [65]. MEMS accelerometers have great potential for extensive application in fault detection or diagnostics of bearings and motors [62], [64], [67], [68], [69], [70], [71], [72], [73], [74], [75], [76], [77], [78]. There are a variety of reports on MEMS accelerometers for measuring vibrations. Some of these include circuits with analog output from the company Analog Devices such as ADXL335 [16], [63], [64], [65], [72], [73], [76], ADXL356 [66], ADXL322 [68], and ADXL203 [77]; those of NXP USA Inc., such as MMA6270QT of [62], and MMA7260Q [43]. Various researchers have also employed those having digital output, some from Analog Devices: ADXL345 [16], [63], [65], [67], [74], and ADXL355 [73]; those of NXP which includes the MMA8451Q [75], and the FXLS8471Q [70]; the LSM6DS3TR of STMicroelectronics [71]; and TDK's MPU-6050 [63], [65], [78]. One of the mathematical tools to analyze oscillations consists of applying to the raw signals of MEMS accelerometers the discrete Fourier transform (DFT), using the FFT (Fast Fourier Transform) algorithm [62], [70], [72], [73], [75], [76], [77], [78]. An application of interest in this work involving these devices is related to the fact that they allow the estimation of the residual vibration introduced by the velocity profiles [16], [25], [29], [43].

This paper presents an algorithm for implementing four-speed profiles in a stepper motor. Additionally, we simulated and measured kinematic variables such as position, velocity, and acceleration associated with the implemented velocity profiles. Finally, through frequency analysis, we analyzed the effect of introducing residual vibrations when implementing the proposed profiles. Some of the main contributions of this study are that the proposed algorithm for generating the speed profiles in stepper motors is simple and does not require devices with high computing power. The input parameters of the algorithm do not require specifications that are not accessible in the technical datasheets of the motors or control drivers, such as prior knowledge of the magnitudes of the speed ramp or maximum acceleration. The algorithm can generate velocity profiles through a logarithmic function, producing faster movements than those generated by the trapezoidal velocity profile, which is widely used for this purpose. By employing MEMS inertial devices, it is possible to measure kinematic variables such as the speed and angular acceleration of the movement of the motor shaft. The remainder of this paper is organized as follows. Section II explains the algorithm used to generate the four–velocity profiles presented in this study. Section III reviews the Fourier analysis used to study residual vibrations. Section IV describes the experimental arrangements and experiments conducted. Section V briefly explains the implementation in a microcontroller of the algorithm described in Section II. Section VI presents the results and discussion. In the appendices are presented the code of the C variant offered by the Arduino platform and the one of Micropython for generating the speed profiles in a microcontroller.

II. TRAJECTORY PLANNING

A. CONSTANT SPEED MOVEMENT

The stepper motor produces angular movement when it receives a logic level signal with a duty cycle different from 0 % or 100 %, which is known as a step. The simplest way to control motor movement involves the application of a digital signal with a fixed period τ_c in s or a fixed frequency f_c in Hz, which produces a uniform time waveform. The angular displacement takes the total time t_T in s. If the motor goes from the initial angular position θ_0 in degrees to the final angular position θ_f in degrees,

$$t_T = N\tau_s, \tag{1}$$

where N is a positive integer corresponding to the total number of steps obtained by

$$N = \text{int}\left(\frac{\theta_f - \theta_0}{\theta_s}\right), \tag{2}$$

and θ_s is the step angle given in degrees per step. Rearranging (2), θ_f is

$$\theta_f = \theta_0 + N\theta_s. \tag{3}$$

The motor speed ω_s in revolutions per second (rps) is constant [6], [24], and is given by

$$\omega_s = \frac{\theta_s f}{360^\circ} = \frac{\theta_s}{360^\circ\,\tau}, \tag{4}$$

where f is the frequency in Hz, and τ is the period in s. The fastest movement, and therefore the minimum t_T from the initial to the final angular position, is achieved by operating the motor's driver at the maximum possible frequency or at the minimum permitted period. However, some motor movements require different speed profile strategies. For example, when the displacement involves liquids or objects with large inertia [38].

B. SPEED PROFILES

The control speed profiles proposed in this study produce three main movement stages. This motion is known as the point-to-point motion [9], [36], [38], [40], [41]. Initially, the speed increases until it reaches a maximum value, corresponding to the acceleration phase; the second stage maintains a constant speed for a specific time. The latter part of the movement reduces the velocity, and the motor shaft decelerates until it reaches the rest. S-curve speed profiles involve up to seven different dynamic stages, with changes in the concavity of the velocity profile, but they still follow the sequence of acceleration, constant speed, and deceleration [38].

The velocity profile varies between two extreme values, ω_{max} and ω_{min}, corresponding to the maximum and minimum angular speeds, respectively. According to (4), ω_{max} is reached when the controller operates at its maximum frequency f_{max}, or with minimum period τ_{min}. Otherwise, the minimum frequency f_{min}, associated with τ_{max} yields ω_{min}.

Considering an initial angle $\theta_0 = 0^\circ$, the profile generation requires some parameters, such as the angular movement direction, pulses or steps per revolution spr of the motor's power driver, final angular position θ_f in degrees, maximum period τ_{max} in μs, minimum time duration τ_{min} in μs, integer number of steps corresponding to the acceleration or deceleration p_N and waveform type. Evidently, $2p_N \leq N$ and the generated profile is symmetrical.

The algorithm for generating any of the four-speed profiles is summarized as follows:

1) Calculate the integer N rewriting (2) in terms of spr as $N = \text{int}\left(({}^{spr}/_{360}) \times \theta_f\right)$.
2) Obtain f_{max}, and f_{min}, with τ_{min}, and τ_{max}, respectively. The units of both frequencies are in Hz. Therefore, the above periods require the conversion to seconds.
3) Obtain the discrete i^{th} frequency $f[i]$ using one of the equations from (5) to (8) to select the desired profile.
4) Considering a 50 % duty cycle input signal, calculate ${}^{\tau}/_2[i]$ with the equation ${}^{\tau}/_2[i] = {}^1/_{2f[i]}$. The i^{th} half–period must be an integer in μs.
5) Keep in high logic state the microcontroller by ${}^{\tau}/_2[i]$ μs, switch to low the step signal the same time.
6) Repeat the steps from 3 to 5, until $i = N$.

1) LINEAR

$$f_l[i] = \begin{cases} a_0 \cdot (i-1) + f_{min}, & \text{for} \quad 1 \leq i \leq p_N \\ f_{max}, & \text{for} \quad p_N < i < (N - p_N) \\ a_1 \cdot (i - N) + f_{min}, & \text{for} \quad (N - p_N) \leq i \leq N, \end{cases} \tag{5}$$

where $a_o = \left(\frac{f_{max} - f_{min}}{p_N - 1}\right)$, and $a_1 = \left(\frac{f_{min} - f_{max}}{p_N}\right)$.

2) QUADRATIC

$$f_q[i] = \begin{cases} a_2 i^2 + a_3, & \text{for} \quad 1 \leq i \leq p_N \\ f_{max}, & \text{for} \quad p_N < i < (N - p_N) \\ a_4 (i - N)^2 + f_{min}, & \text{for} \quad (N - p_N) \leq i \leq N, \end{cases} \tag{6}$$

where $a_2 = \frac{f_{max} - f_{min}}{p_N^2 - 1}$, $a_3 = f_{min} - a_2$, and $a_4 = \frac{f_{max} - f_{min}}{p_N^2}$.

3) SINUSOIDAL

$$f_s[i] = \begin{cases} a_5 \cos\left(\frac{\pi(i-1)}{p_N - 1}\right) + a_6, & 1 \leq i \leq p_N \\ f_{max}, \quad \text{for} \quad p_N < i < (N - p_N) \\ a_5 \cos\left(\frac{\pi(i-N)}{p_N}\right) + a_6, \ \text{for} \quad (N - p_N) \leq i \leq N, \end{cases} \tag{7}$$

where $a_5 = \frac{f_{min} - f_{max}}{2}$, and $a_6 = \frac{f_{max} + f_{min}}{2}$.

4) LOGARITHMIC

$$f_{log}[i] = \begin{cases} a_7 \ln(i) + f_{min}, & 1 \leq i \leq p_N \\ f_{max}, \ \text{for} \quad p_N < i < (N - p_N) \\ a_8 \ln(-i + N + 1) + f_{min}, \\ \text{for} \quad (N - p_N) \leq i \leq N, \end{cases} \tag{8}$$

where $a_7 = \frac{f_{max} - f_{min}}{\ln(pN)}$, and $a_8 = \frac{f_{max} - f_{min}}{\ln(pN+1)}$.

In a stepper motor the discrete angular position $\theta[i]$ in degrees at the i^{th} step is given by

$$\theta[i] = \theta_0 + i\theta_s = \theta_0 + i \times \frac{360}{spr}. \tag{9}$$

The algorithm produces $\tau[i] \neq \tau[i-1]$ during the acceleration and deceleration movement stages, where $\tau[i]$ corresponds to the discrete i^{th} period. The above condition satisfies the requirement for generating non–uniform time increments to modify the speed of the motor [8], [9], [12],

[14], [32], [37]. The microcontroller does not require saving the value of any $\tau[i]$ obtained in the Step 4 of the algorithm to generate the desired profile. However, saving these individual magnitudes and their accumulation creates a discrete time array $t[i]$, which, in combination with (9), helps to compute the t vs. θ curve of the angular movement. The first and second numerical derivatives of the above curves are the angular speed ω and angular acceleration α, respectively.

III. FOURIER ANALYSIS

It has been well known for decades that the FFT algorithm is a mathematical tool that efficiently computes the Discrete Fourier Transform (DFT) [79]. The FFT algorithm was applied to the acceleration data to explore the effect of moving a rotation stage with the velocity profiles proposed in this study. From the sampled signal $x[n]$, the DFT $X[k]$ [80], is given as

$$X[k] = \sum_{n=1}^{N_F} x[n] e^{-jk\frac{2\pi n}{N_F}}, \quad k = 1, \ldots, N_F, \tag{10}$$

where N_F denotes the total number of data points. Due to aliasing, the maximum frequency is resolved from the signal $x[n]$ with $n = 1, \ldots, N_F$ is $f_F[k = {}^{N_F}/_2] = {}^{f_s}/_2 = {}^1/_{2t_s}$, where f_s is the sampling rate in Hz, and t_s is the sampling time in s; considering that the individual times are equidistant from each other. Each of the Fourier coefficients $X[k]$ given by (10) has a discrete frequency [81], given by

$$f_F[k] = \frac{kf_s}{N_F} = \frac{k}{N_F t_s}, \tag{11}$$

where $k = 1, \ldots, \frac{N_F}{2}$.

The frequency composition of the acceleration Fourier spectrum is strongly correlated with the *jerk*. It has a greater magnitude when the system increases its speed [16].

When studying vibrations, the Power Spectral Density (PSD) has advantages over the FFT for frequency analysis [82]. The discrete PSD S[k] is given by

$$S[k] = \lim_{N_F \to \infty} \frac{2|X[k]|^2}{N_F}, \quad k = 1, \ldots, \frac{N_F}{2}. \tag{12}$$

IV. EXPERIMENTAL SETUP

The experimental setup consisted of three arrangements. The first experiment had the objective of validating (4) in stepper motors because it is the basis for generating the proposed speed profiles. In addition, this setup served to obtain experimental data from an encoder attached to the motor shaft to obtain the curves θ vs. t for different velocity waveforms. The second configuration used a MEMS gyroscope to perform the experimental measurement of ω of a rotation stage subjected to four-velocity profiles. The last experiment involved a MEMS accelerometer fixed to a mechanical rod attached to the central axis of the rotation stage to measure the tangential acceleration a_t and the temporal variations in the vertical acceleration due to the residual vibrations introduced by the four velocity profiles.

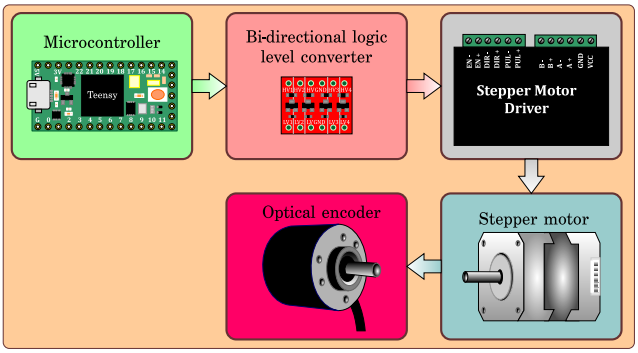

FIGURE 1. Block diagram of the first experimental setup.

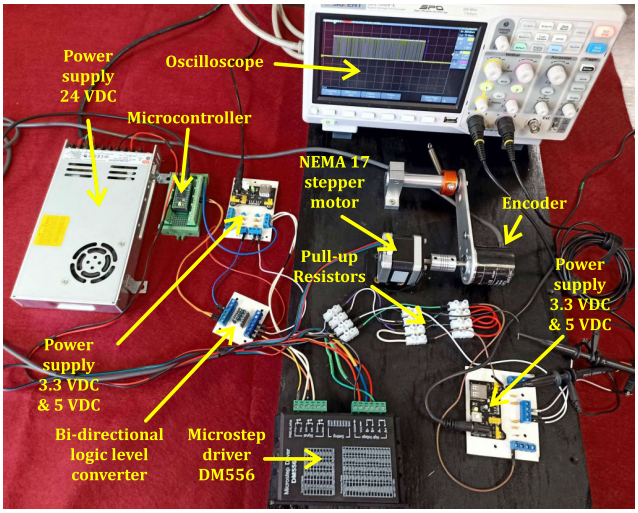

FIGURE 2. First experimental setup.

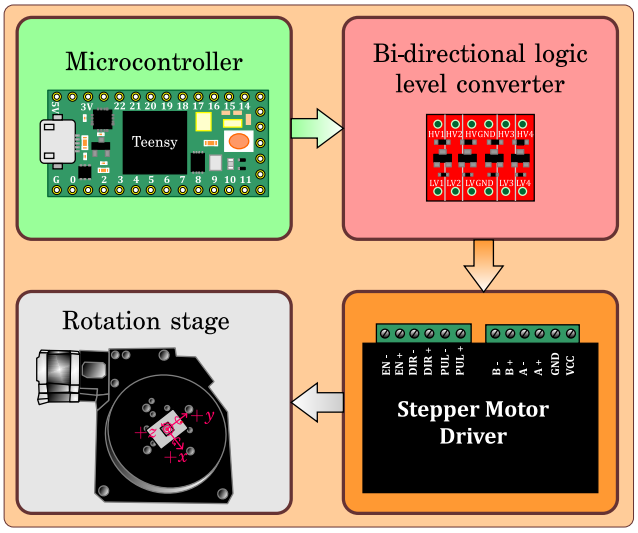

FIGURE 3. Block diagram of the second experimental setup.

A. FIRST EXPERIMENTAL ARRANGEMENT

Equation (4) describes motion with a constant angular velocity. Therefore, keeping θ_s constant and varying f, the

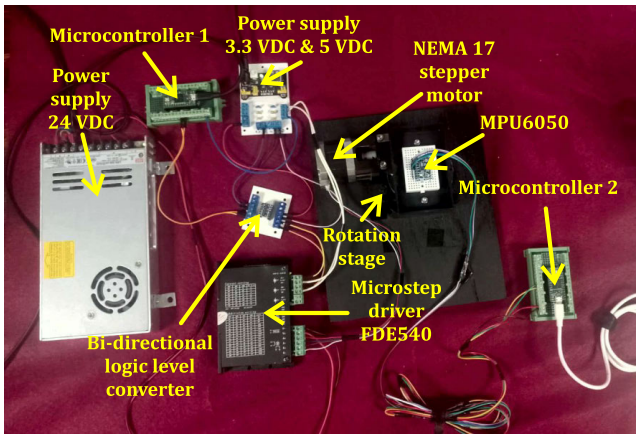

FIGURE 4. Second experimental setup.

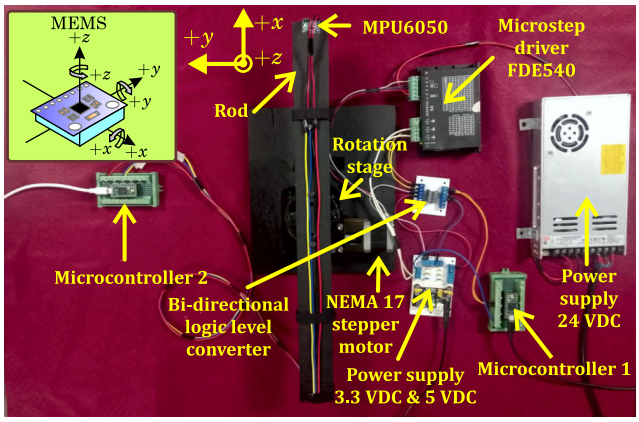

FIGURE 5. Third experimental setup.

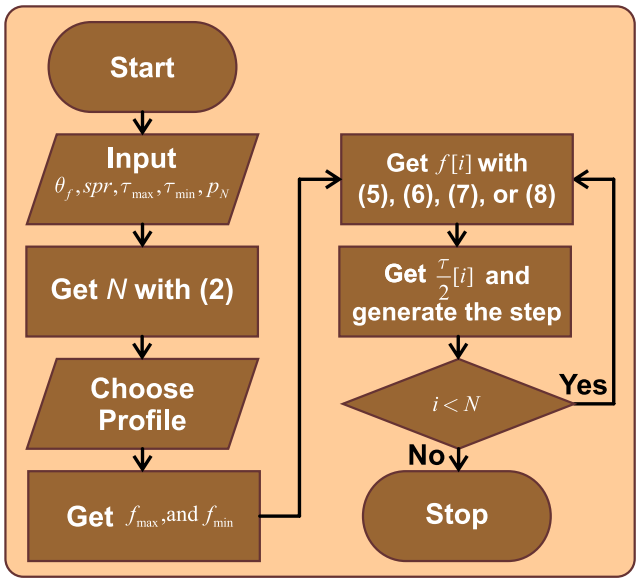

FIGURE 6. Flowchart of the algorithm.

relationship between ω and f is expected to exhibit a linear behavior. The first experiment involved applying 100 digital pulses with a duty cycle of 50 % and a fixed frequency to the motor. The speed was calculated indirectly by recording the response provided by the optical encoder. This device has a digital output that changes the logical state when its axis rotates; each logical transition having two rising edges and two falling edges corresponds to a pulse of the encoder and, of course, to its discrete angular movement. The calculation of the angular velocity once the optical sensor signal is recorded requires the determination of the total number of pulses N_e generated by the encoder and the measurement of the time difference between the last and first pulses Δt_e. The axis speed of the position sensor ω_e in degrees per second, calculated using these variables is as follows:

$$\omega_e = \frac{N_e \theta_{se}}{\Delta t_e}, \tag{13}$$

where θ_{se} is the angle per step of the encoder in degrees per step. For example, if the resolution of the optical position sensor is 2500 steps per revolution, $\theta_{se} = {}^{360}/_{2500}$. Varying the pulse frequency received by the motor and repeating the procedure described above gives the relationship between ω and f. Considering the tight coupling between the encoder and motor shafts, both move at the same speed.

The same experimental arrangement also serves to obtain the values of $\theta[i]$ and $t[i]$ originated from the application of the velocity profiles described by (5) to (8). The motor fulfills with (3), in the same way the encoder shaft has the i^{th} position given by $\theta_e[i] = i\theta_{se}$, if $\theta_0 = 0$. The discrete–time of the encoder $t_e[i]$ is obtained by measuring the duration of each pulse of the signal generated by the optical sensor. If $\theta_{se} \leq \theta_s$, then $\theta[i]$ and $t[i]$ of the motor are described by $\theta_e[i]$ and $t_e[i]$.

Fig. 1 shows the block diagram of the first experimental arrangement. The first component in this figure consists of a microcontroller. The experiments with the Arduino platform involved the Teensy 4.0 development board of the PJCR company, which offers much higher performance than other boards of the Arduino organization, such as Uno and Mega. In Micropython, Raspberry Pi Pico board was chosen. Experiments to generate the velocity profiles described by (5)–(8) involved both devices. In both microcontrollers, the general purpose input/output (GPIO) works with a high logic level of 3.3 V. This is unsuitable for the motor power controller used in the experiments. The DM556 stepper driver operates at 5 V; for this reason, there is an intermediate device between both blocks, which is a bi-directional logic converter module which raises to 5 V the voltage level of the pulses generated by the GPIO of the microcontrollers. DM556 has switches that adjust the maximum operating current and motor handling with the microstepping technique [16], [29], [83]. In essence, this driver receives two logical signals that define both the direction of rotation and pulses for moving the motor. It also has two outputs for the two coils that the generic stepper motors have. The final block corresponds to the encoder. The one used in this experiment was E6B2-CWZ6C of the OMROM brand, which has outputs A, B, and Z. It can be powered from 5 to 24 V DC, and with a resolution

of 2500 steps per revolution. With an oscilloscope, and their respective pull-up resistors, it is possible to record outputs A and B of the optical sensor. The first experiment involved two generic stepper motors, NEMA 17 and NEMA 23. The remaining experiments only involved the NEMA 17 motor.

Fig. 2 shows a photograph of the elements used in the first experimental setup. It displays the NEMA17 motor, joined with a flexible shaft coupler to the E6B2-CWZ6C encoder; its outputs A and B required a pull-up resistor, whose magnitude in this experiment was 10 kΩ. The oscilloscope SIGLENT model SDS 1202X-E with a bandwidth of 200 MHz recorded signals A and B from the encoder operated at 5 V, generated by a power source. The DM556 motor driver received 5 V pulses from the bidirectional logic converter, whose inputs were the 3.3 V signals generated with the Teensy 4.0 board programmed by a computer. The motor driver operated at 24 V provided by the power supply MEAN WELL model LRS-350-24.

B. SECOND EXPERIMENTAL ARRANGEMENT

The second experimental arrangement was designed to measure the temporal evolution of ω when applying the speed profiles given by (5)–(8) in the rotation stage. The velocity sensor is a MEMS gyroscope. Fig. 3 shows a block diagram of the main elements used to determine ω. As stated, C offers better execution times than Micropython [22]. Thus, the first block involved a Teensy 4 board programmed using the Arduino IDE. The rotation stage depicted by the fourth block receives power from the NEMA 17 motor, which requires a driver to control it. The device employed in the third block was the FDE540 (FEREME). The second block refers to the voltage converter required to drive the FDE540. The rotation stage RTS9060 from Colibri Technologies Company used in this experiment has a mechanical reduction of 100:1. When the motor rotates 100 revolutions, the stage does one.

The gyroscope used in this experiment was a module based on the MPU6050 integrated circuit, which combines an accelerometer and a gyroscope, both of which can measure along three axes. Therefore, the MPU6050 has six degrees of freedom (DoF). The device communicates using the eye-squared-C (I^2C) protocol. The gyroscope measures ω by using the Coriolis effect. In this experiment, the sensor was firmly attached to the rotation stage with the $+z$ or yaw axis aligned vertically. This axis coincided with the rotation axis of the RTS9060. Thus, by measuring ω in $+z$, the angular speed for each velocity profile can be determined.

Fig. 4 shows the second experimental setup. By installing the Adafruit company library on microcontroller 2, MPU6050 transfers the angular velocity values to the computer. Fig. 4 also shows the alignment of the MEMS gyroscope with the rotation axis of the stage and the other elements used to generate the speed profile in the rotation stage.

C. THIRD EXPERIMENTAL ARRANGEMENT

The kinematic variables used to describe the movement of an object include its position, speed, and acceleration. The previous experiments aimed to measure the first two variables. The last experiment aimed to measure the accelerations generated by the different velocity profiles. Using MPU6050, we measured a_t [84], which is given by

$$a_t = r\frac{d\omega}{dt} = r\alpha, \tag{14}$$

where r is the radius. The dependence of (14) with r implies that the sensor should not be placed in the center of the rotation stage as in the previous experiment. For this reason, we fixed a 48 cm-long bar at the center of the base, and at the end, we placed a MEMS accelerometer. Fig. 5 shows the third experimental setup with the elements for generating the speed profiles in the rotation stage and microcontroller 2 for recording the acceleration data. With MPU6050, we obtained the measurements of a_t (corresponding to the $+y$ axis shown in fig. 5) and the acceleration of the vertical direction a_z (corresponding to the $+z$ axis shown in fig. 5). a_t gives α trough (14), and a_z is related to the residual vibrations introduced by the different speed profiles. The oscillations caused by the sudden variation of α negatively affects the angular motion [30], [38], [39], [40], [41], [42].

V. SOFTWARE IMPLEMENTATION

Fig. 6 shows the flowchart of the proposed algorithm. It displays the inputs required for the generation of the speed profile. After setting the parameters, the time that the microcontroller will be in the high and low states is calculated using (5), (6), (7), or (8). The algorithm is stopped when $i = N$. The implementation of the algorithm in the Arduino platform and in Micropython is described below.

A. ARDUINO PLATFORM

The generation of the speed profiles is based on the SpeedMotorProfile class, which is shown in Appendix A. When loaded into the microcontroller, its initialization requires two digital outputs, one intended to control the direction of movement, either clockwise or counterclockwise, and the other to inject the step signals into the motor's driver. The first method of the class, called steps, obtains the total number of steps N, their parameters are the final angle θ_f, and the steps per revolution of the motor's driver. Four methods are used to define the speed waveform. The parameters of these methods are N, pN corresponding to the number of steps in which acceleration and deceleration occur. Considering a symmetrical speed profile, the minimum period τ_{min} which is related to the maximum frequency through the equation $\tau_{min} = {}^1/_{f_{max}}$, the maximum period obtained utilizing $\tau_{max} = {}^1/_{f_{min}}$, and the direction of rotation defined by the values of 0, and 1. The program assumes that if $N < 20$, the motor rotates at ω_{min}.

The following code corresponds to an example for executing the algorithm in the Arduino IDE. This code calls

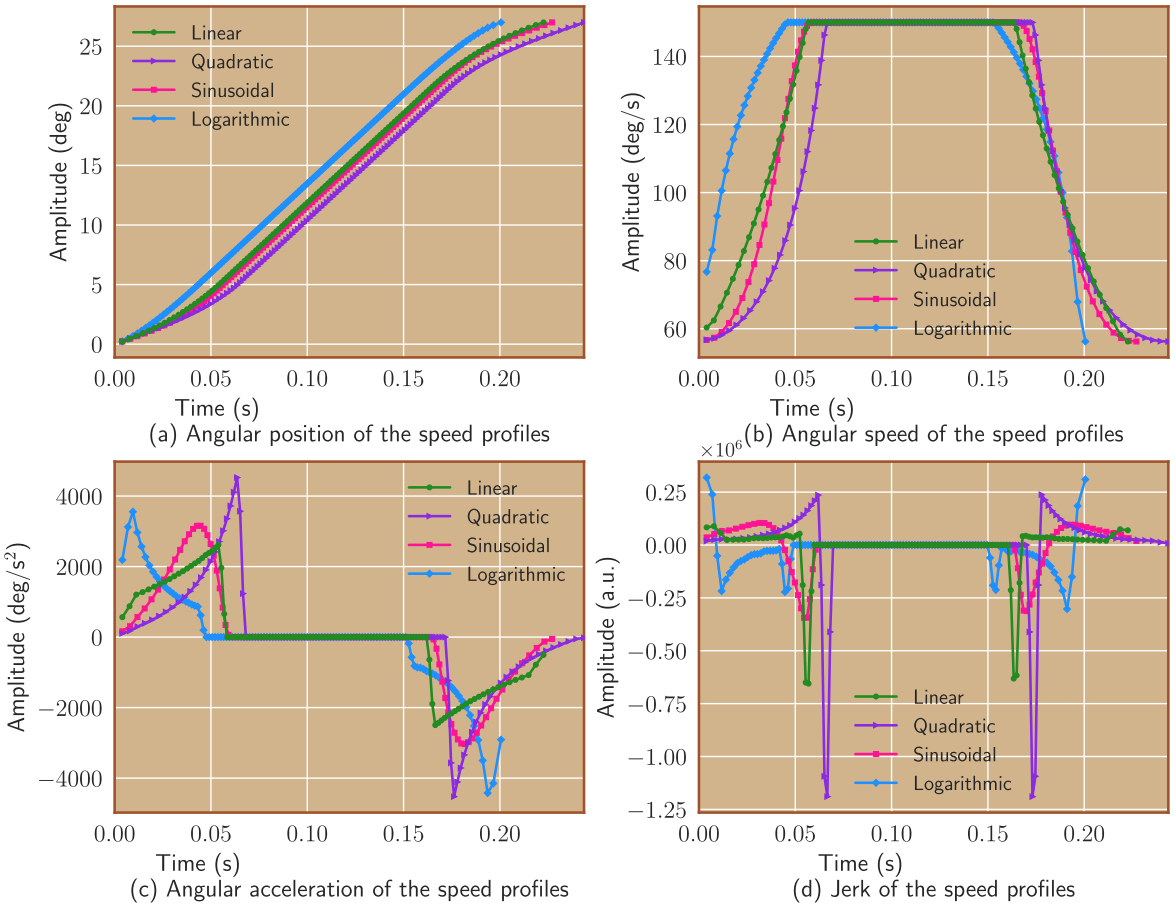

FIGURE 7. Simulations of position, speed, acceleration, and jerk in terms of time based on the computation of (5) to (8).

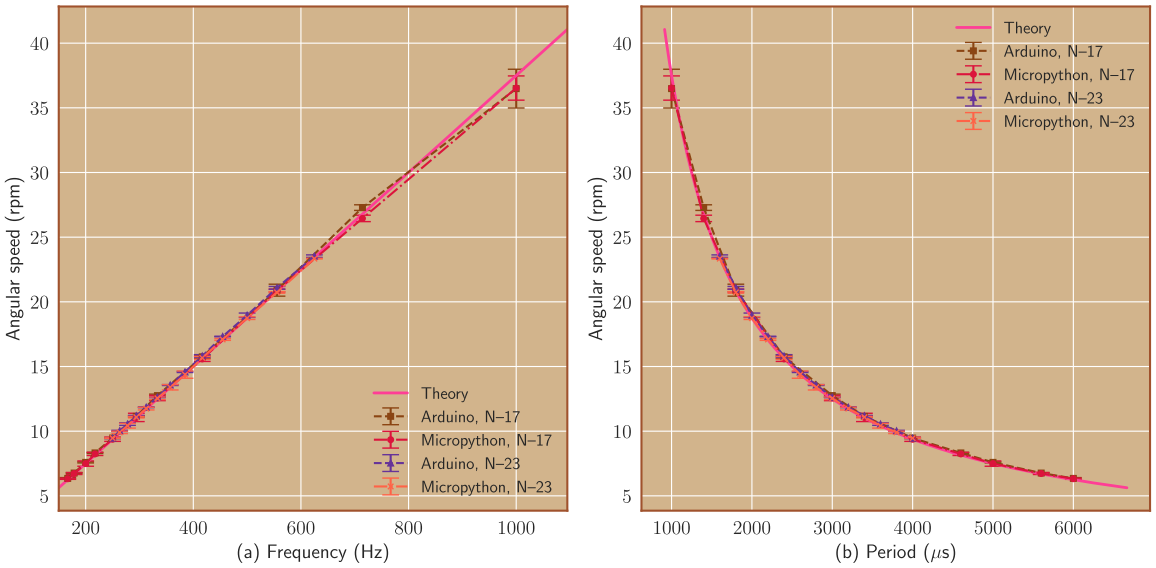

FIGURE 8. Angular speed versus frequency and period for the NEMA 17 and the NEMA 23 stepper motors.

the SpeedMotorProfile class to produce a single angular movement of 27° with a logarithmic speed profile, having a maximum period of 4000 μs and a minimum of 1500 μs, considering that the motor driver operates at 1600 revolutions

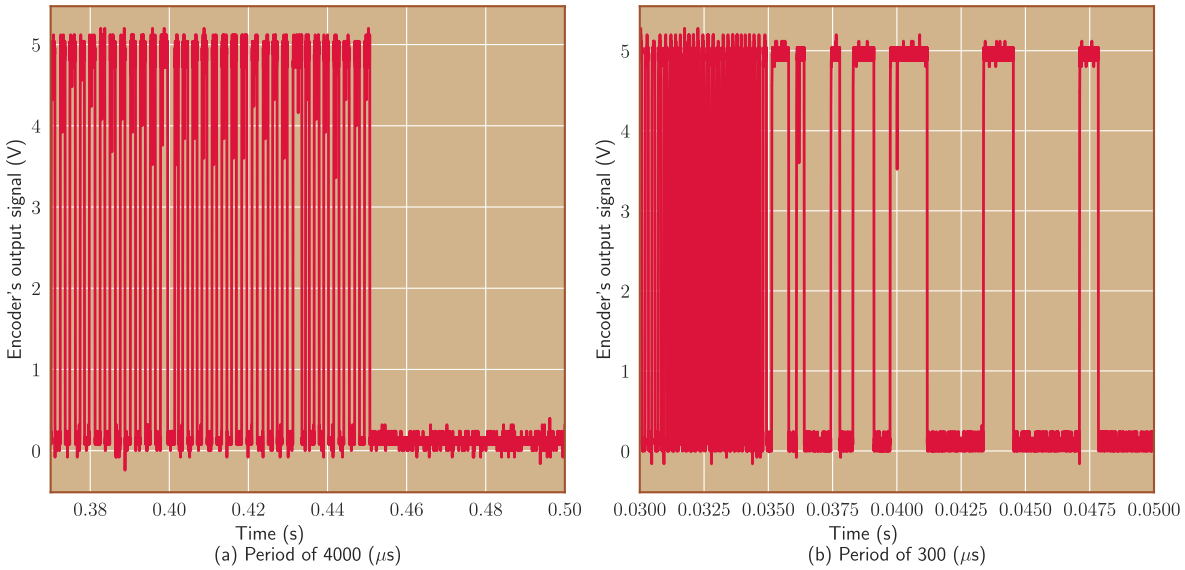

FIGURE 9. Waveforms of optical encoder when driving a stepper motor at the periods 4000 and 300 μs. In this experiment $\theta_{se} = 0.144$ degrees per step and $\theta_s = 0.225$ degrees per step.

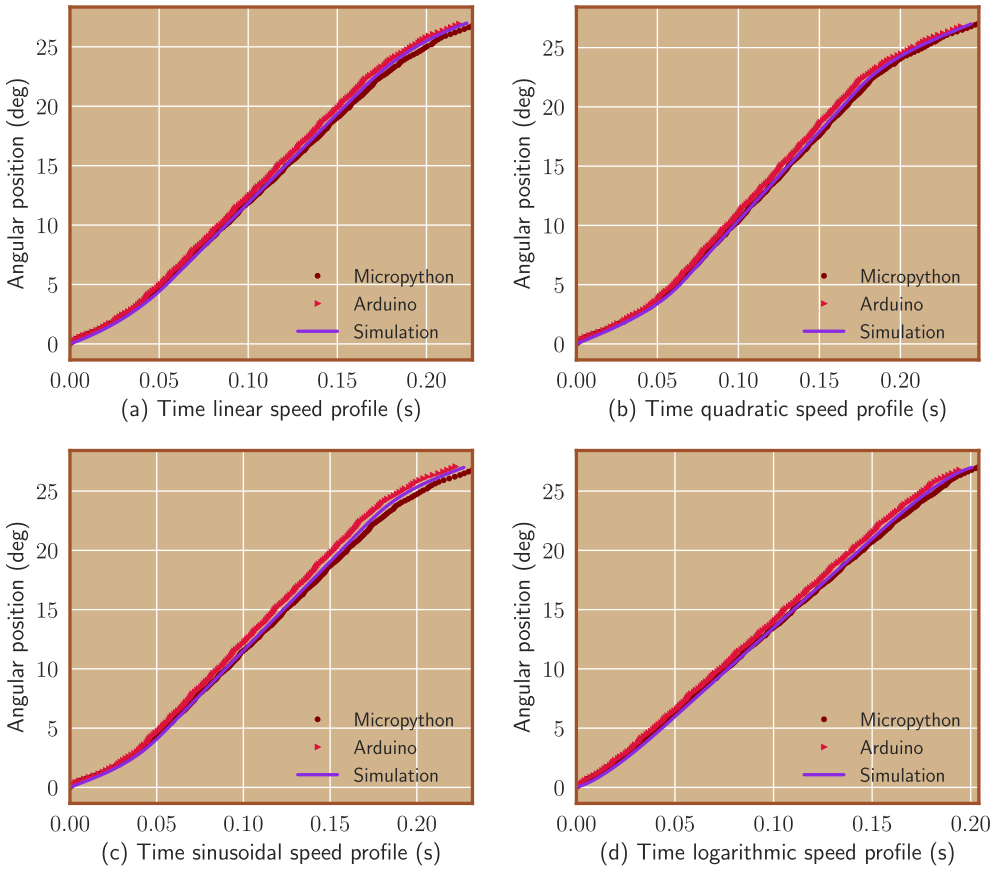

FIGURE 10. Experimental data and simulations of angular position versus time based on the computation of (5) to (8).

per step and that 20 % of N, corresponds to the value of p_N. This example considers that digital output 18 controls the motor's direction, and output 19 sends pulses to the motor's driver.

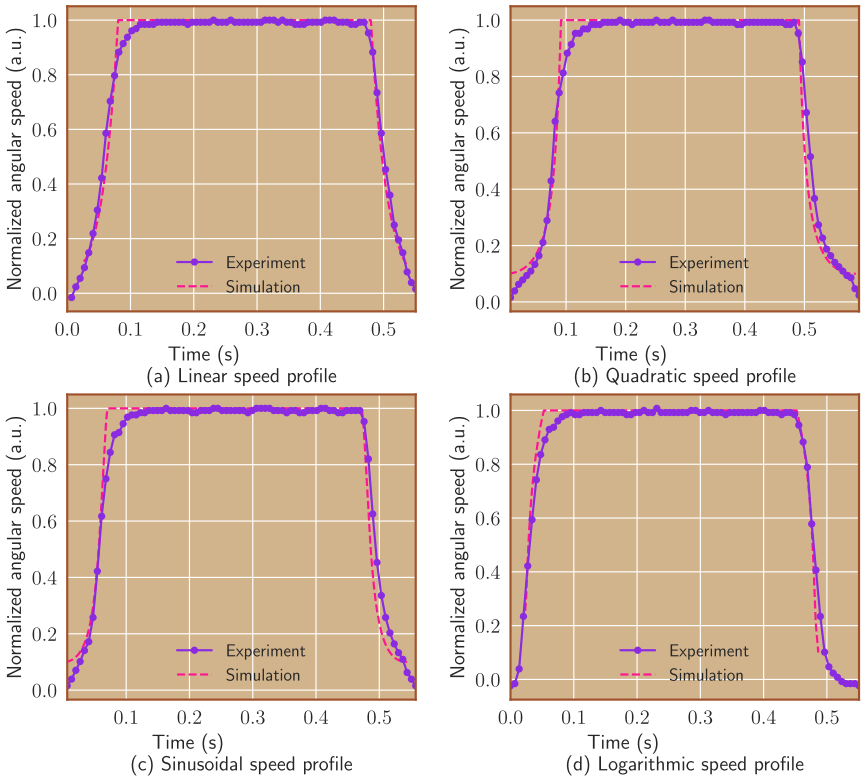

FIGURE 11. Experimental and simulated angular speed profile based on the computation of (5) to (8).

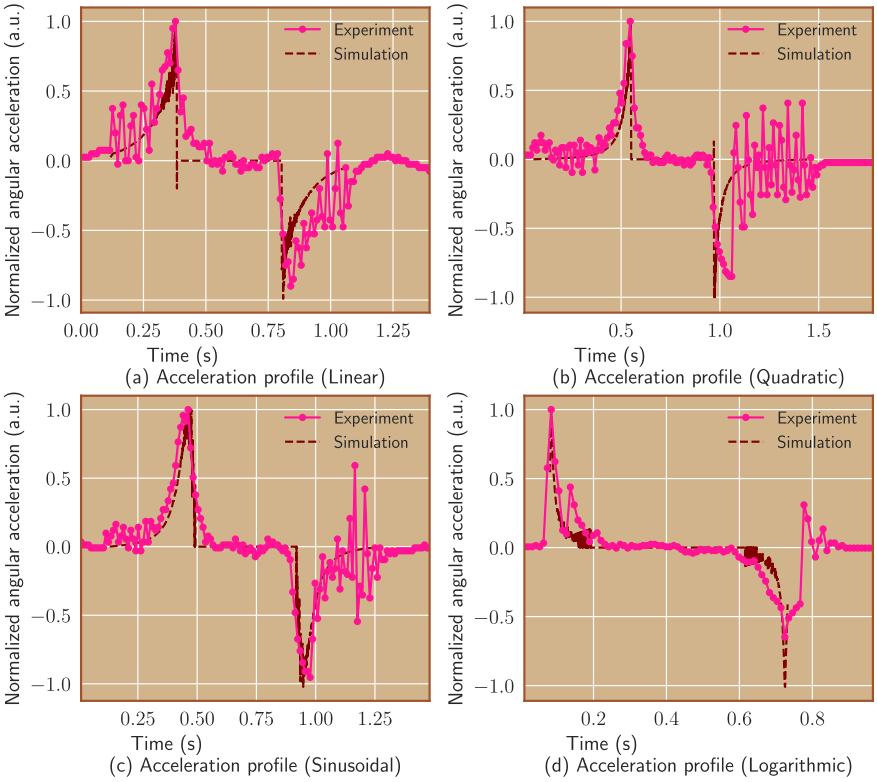

FIGURE 12. Experimental and simulated angular speed based on the computation of (5) to (8).

```
#include "SpeedMotorProfile.h"

#define dirPin 18
#define stepPin 19
const int maxPeriod =4000;
const int minPeriod =1500;
const int stepsPerRev = 1600;

float angle = 0;
int N = 0;
int count = 0;
int pN = 0;
int direction = 0;
SpeedMotorProfile speedMotorProfile(dirPin,stepPin);
int steps = 0;

void setup() {
  pinMode(dirPin, OUTPUT);
  pinMode(stepPin,OUTPUT);
}//setup()

void loop() {

  if (count == 0){
    delay(100);
    count = 1;
    angle = 27;
    direction = 0;
    N = speedMotorProfile.steps(angle, stepsPerRev);
    pN = (int)(float(N)*0.2);
    speedMotorProfile.logarithmic(N,pN,minPeriod,
     maxPeriod,direction);
  }//if (count == 0)
}//loop
```

B. MICROPYTHON

The following code repeats the same operation explained in the previous section but for Micropython. The only difference is that the output pin controlling the motor's direction now corresponds to terminal 26, and the steps injected into the motor's driver originate from pin 27.

```
from speedprofile import SpeedMotorProfile
from time import sleep

motor_profile = SpeedMotorProfile(26,27)

s = motor_profile.steps(27,1600)
pN = int(s*.2)
print(pN)
direc = 1
min_period = 1500
max_period = 4000
sleep(1)
motor_profile.logarithmic(s,pN,min_period,max_period,
    direc)
```

The Appendix B shows the SpeedMotorProfile class written in Micropython.

VI. RESULTS AND DISCUSSION

A. SIMULATION

Computational simulation is a tool that provides information of interest to the object of study without conducting experiments, thereby saving time. For this reason, we simulated θ, ω, α, and *jerk* in terms of time using the proposed algorithm. The parameters used to generate fig. 7 are listed in Table 1. The graph of θ versus t shows that the logarithmic profile is the fastest to reach a final position of 27°, followed by the linear and sinusoidal profiles. The longest time required to reach a final position corresponds to the profile generated by using the quadratic function. The logarithmic profile reduces almost 10 % of the time to reach the final position compared with the linear profile and approximately 18 % compared with the quadratic profile. fig. 7 shows the three stages of movement, where the position in terms of time is nonlinear during the acceleration and deceleration phases, and a straight line is generated when the motor travels at a constant velocity.

The algorithm generates the values of $\theta[i]$ and $t[i]$. Starting from these values and performing the numerical derivative using the gradient function of the NumPy package of Python, we obtained ω versus $t[i]$. α and *jerk* were obtained by performing further numerical derivatives.

Fig. 7 shows a plot of angular velocity. The profile obtained using the linear equation did not generate a perfectly straight line. This phenomenon is related to the fact that the values of $\tau[i]$ owing to hardware constraints can only be integer multiples of one microsecond. Because $f = 1/\tau$, the frequency, and consequently, the velocity does not have an infinite resolution. Both variables become discrete, based on the truncated value of $\tau[i]$. The numeric derivative causes the initial velocity points of all the profiles to not match. The graphs for the quadratic, sinusoidal, and logarithmic functions follow the expected waveform governed by (6) to (8), respectively.

Fig. 7 shows a graph of α in terms of the time associated with the different velocity profiles. A visual inspection revealed that the acceleration of the profile generated using the linear equation did not follow a step function. Initially, the acceleration gradually increases and descends to the constant velocity region less abruptly than the theory predicts for a trapezoidal profile. The behavior in the acceleration stage is similar to that presented in the deceleration stage; they would almost match if any of the graphs in these two stages suffered a reflection along the x–axis and then another along the y–axis. The acceleration generated by the quadratic equation has a greater amplitude. The smoothest α function corresponded to the profile calculated using the sinusoidal equation.

Abrupt changes in the amplitude of the acceleration or deceleration profile are sources of residual vibrations [38]. The α waveform corresponding to the logarithmic profile in the first phase of the movement exhibited rapid changes in acceleration at the beginning and near the region where the motor travels at a constant speed. α of the quadratic profile starts varying smoothly, but near the zone of constant velocity, it undergoes abrupt changes in acceleration compared with the other α profiles. This behavior influences *jerk* and generates more residual vibrations. Applying a sinusoidal waveform with smooth acceleration enhances the durability and maintenance of the motors, if that is the goal.

Fig. 7 also shows *jerk* in terms of t. The temporal evolution of this variable, generated by using a sinusoidal equation has a smoother waveform. The plots produced with the other equations exhibit peaks resembling an impulse function, with the greater amplitude shown for the waveform created by using (6). Despite the rapid speed change for the curve obtained by using (8), its amplitude is not as pronounced.

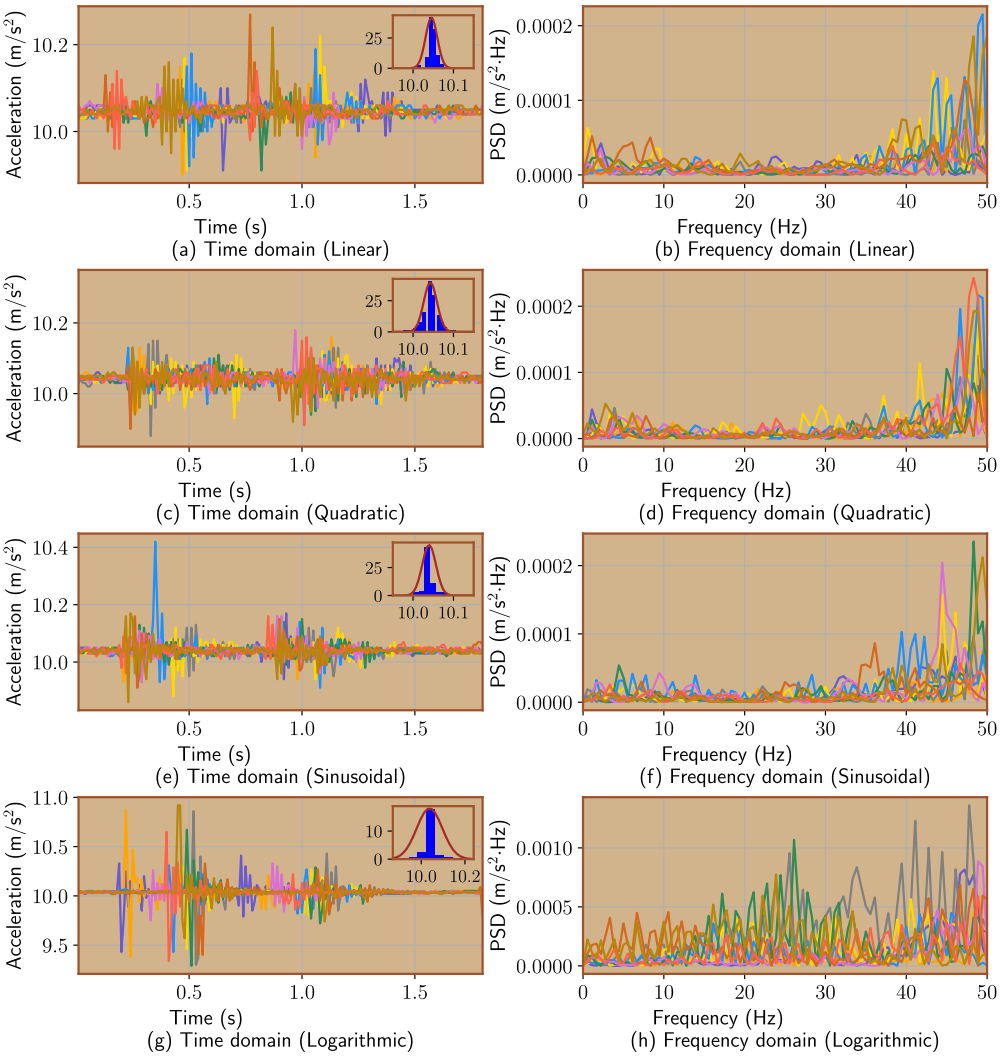

FIGURE 13. Time and frequency domain waveforms of the vertical acceleration based on the computation of (5) to (8).

TABLE 1. Simulation parameters.

Parameter	Magnitude
θ_f	27°
θ_s	$\frac{360°}{1600}$
τ_{max}	4000 μs
τ_{min}	1500 μs
p_N	int(0.2N)

jerk exhibits the expected behavior, because both the sinusoidal waveform for α and its related *jerk* are smooth. In the quadratic profile, the *jerk* amplitude is significant in the regions where the motor moves from acceleration to constant speed and from this regime to deceleration. Modifying the quadratic speed profile by adding another function before the constant velocity phase might mitigate this problem.

B. ANGULAR SPEED VERSUS FREQUENCY AND PERIOD

Fig. 8 shows the data from the first experiment. In a wide frequency range from 165 to 625 Hz, the behavior described by (4) agrees with the experimental measurements. No significant deviation exists in the linear relationship between the angular velocity and step frequency for any of the tested motors. The two microcontrollers used in the tests exhibited essentially the same responses. At higher frequencies of approximately 1000 Hz, the experimental data no longer followed a linear behavior, for the reasons we will explain in the next paragraph. The second plot in fig. 8 shows an inversely proportional relationship between ω and τ. For this reason, the velocity profiles generated with (5)–(8) use the frequency rather than the period as an independent variable.

Fig. 9 shows the waveforms of the two readings made with the encoder when the motor received pulses of two different periods $\tau = 4000$ μs, and $\tau = 300$ μs. In this

TABLE 2. Simulation parameters for getting the speed profiles.

Parameter	Magnitude
θ_f	$30°$
θ_s	$\frac{360°}{20000}$
τ_{max}	4000 μs
τ_{min}	400 μs
p_N	int$(0.1N)$

TABLE 3. Configuration parameters for the gyroscope MPU6050.

Parameter	Magnitude
t_s	7 ms
Gyroscope range	$\pm$250 deg/s
Filter bandwidth	10 Hz

TABLE 4. Simulation parameters for getting the angular acceleration curves.

Parameter	Magnitude
θ_f	$4.5°$
θ_s	$\frac{360°}{160000}$
τ_{max}	5000 μs
τ_{min}	300 μs
p_N	int$(0.15N)$

experiment the step angle for the motor and optical sensor were $\theta_s = {}^{360°}/_{1600}$ and $\theta_{se} = {}^{360°}/_{2500}$, respectively. The condition $\theta_{se} < \theta_s$ ensured that the optical sensor had a better angular position resolution than that associated with the motor. Under ideal circumstances the application of $N = 100$ pulses to the motor according to (3) should produce $\theta_f = {}^{45}/_{2}$ degrees and the pulses generated for the encoder should be $N_e = \text{int}\,({}^{625}/_{4})$. Depending on the initial position of the encoder, the oscilloscope waveform consists of 156 or 157 steps. At $\tau = 4000$ μs, the signal contained 157 pulses. With the control signal having $\tau = 300$ μs, the oscilloscope waveform registered 163 steps, generating an inaccurate angular displacement of the motor of 0.864 degrees. The second plot in Fig. 9 shows the excess steps and their temporal distortion. The oscillograph with $\tau = 4000$ μs in a portion of it exhibited temporal uniformity in the region related to the logical level transitions, as opposed to the other waveform for the signal of $\tau = 300$ μs where the duration of the steps of the encoder varied for the last pulses. This oscilloscope's signal exhibited temporal uniformity at the beginning, which implies that the hardware can operate at that speed; however the motor's inertia becomes a problem. Introducing a speed profile helps reduce the angular positioning errors owing to excessive step generation.

C. ANGULAR POSITION IN TERMS OF TIME

Fig. 10 shows the measurements of θ versus t obtained with the experimental arrangement represented in fig. 1 and its simulation using the parameters listed in Table 1. The experimental data were very similar to those of the simulation. The four graphs show that the microcontroller programmed with Micropython took longer time to reach the final position than that controlled with the variant of the C language of the Arduino platform. This result is not surprising because Micropython is slower than C [22]. As in the simulation shown previously, the waveform with the shortest duration was generated from the logarithmic equation, and the waveform produced through the quadratic function required the longest time to reach θ_f.

D. SPEED PROFILES

Verifying that the algorithm generates the proposed speed profiles is one of the most relevant aspects of this study. Fig. 11 shows the plots of ω in terms of t for the simulated waveforms and the data measured using the gyroscope MPU6050. The experiment used the setup shown in fig. 3. The curves in the figure were normalized to favor a straightforward visual comparison. We did not calibrate the gyroscope, and it provided a reading of -0.03 deg/s when the rotation stage was at rest. For this reason, some experimental points had negative values. Table 2 lists the parameters used in the simulations. The magnitude of θ_s displayed in this table considered the 100:1 reduction of the rotation stage. We used Adafruit's library for the Arduino to configure the MPU6050 module. Table 3 lists the parameters used in the experiment; t_s denotes the sampling time. Without exception, the experimental profiles had a longer duration than that predicted by the simulation, a situation due to the delay times introduced in the operations carried out with hardware and software. In the experiment, the rotation stage started at rest, contrary to the behavior of the simulated speed starting at ω_{min}. For this reason, we see an increase in the velocity from zero to ω_{max} at the leading edge and reaching rest at the trailing edge in the experimental curves.

The logarithmic velocity profile was the fastest, and the quadratic profile was the slowest. There is a reasonable agreement between the experimental curves and the simulation results. However, on the leading edge, they exhibit a curvature that does not predict the simulation having sharp edges. In the experiments, we found a trade-off: reducing the magnitude of the gyroscope filter bandwidth produced smoother curves, but simultaneously introduced a distortion in the speed profile. Additionally, we did not estimate the error introduced by the drift inherent to these devices [46], [85], which may have modified the appearance of the experimental curve.

E. ANGULAR ACCELERATION

Fig. 12 shows α in terms t for a simulation with the parameters given in Table 4, and the experimental values of a_t ($+x$–axis) measured with the accelerometer MPU6050 configured with the parameters shown in Table 5. The

TABLE 5. Configuration parameters for the accelerometer MPU6050.

Parameter	Magnitude
t_s	10 ms
Accelerometer range	$\pm 2g$
Filter bandwidth	5 Hz

TABLE 6. Speed profile parameters for generating residual vibration.

Parameter	Magnitude
θ_f	$4.5°$
θ_s	$\frac{360°}{160000}$
τ_{max}	3000 μs
τ_{min}	300 μs
p_N	int$(0.2N)$

TABLE 7. Magnitudes of average and standard deviation for the vertical acceleration.

Profile	Average (m/s^2)	Standard deviation (m/s^2)
Linear	10.045	0.025
Quadratic	10.042	0.026
Sinusoidal	10.039	0.022
Logarithmic	10.038	0.088

experiment required the setup described in Fig. 5. The values of α and a_t were normalized for easy comparison by visual inspection. The simulation of α has low amplitude noise owing to the truncation of $\tau[i]$ explained above. The experimental data in the acceleration region were very similar to the simulated waveforms for α. However, there is noise in the signals, most likely due to residual vibrations, which will be discussed later. The effects of introducing non-sharp edges in the data previously observed in the experimental velocity profiles are also observed mainly in the profile generated by using the linear equation. This behavior occurs in the region where the acceleration decreases until reaching the constant velocity zone. The acceleration function obtained using the sinusoidal equation was the smoothest; however when the rotation stage almost reached rest, the residual vibration distorted the waveform.

F. RESIDUAL VIBRATION

The setup shown in Fig. 5 was used to study the residual vibration. Table 6 lists the parameters used to generate the velocity profiles. Table 5 lists the accelerometer configuration. Fig. 13 shows for each of the velocity profiles ten time-domain acceleration waveforms measured along the vertical direction (z–axis of Fig. 5). Fig. 13 also shows the spectral power density for the previous ten curves calculated using the SciPy package for Python, occupying Welch's method. During this experiment, it was possible to capture the transient behavior of the acceleration of the vertical acceleration a_z because the sampling period was minor, by a few orders of magnitude compared to the motor's movement duration. The first target speed of the profile is small but not zero; when the movement starts, the amplitude of a_z in the time domain changes as soon as the speed profile is generated, exhibiting the first significant peak. The second prominent peak occurred when the deceleration phase began. The occurrence of this behavior was corroborated by comparing the data for a_t and a_z, which were obtained simultaneously. As soon as the rotation stage reached rest, a_z remained constant. The baseline of the vertical acceleration in the time domain was approximately 10.04 m/s^2 because the acceleration of gravity g was present in that direction. Table 7 shows the values for the average and standard deviation σ of a_z for the ten waveforms in the time domain and for each of the speed profiles generated by using (6) to (8). The average varied from approximately 10.038 to 10.045 m/s^2. The standard deviation changed from approximately 0.022 to 0.088 m/s^2, and the sinusoidal velocity profile had the smallest value. The magnitude of this variable was approximately 1.13, 1.18, and 4 times more significant for the linear, quadratic, and logarithmic profiles, respectively, compared to σ of the sinusoidal profile. Therefore, the logarithmic profile introduced more residual vibrations than the sinusoidal waveform. Fig. 13 shows the insets of the histograms of the waveforms related to the experimental data in the time domain. The plot for each of the speed profiles consisted of ten time-domain acceleration waveforms, and any single waveform had 180 values of a_z. Multiplying both variables gives 1800 values of a_z, which were the ones used to generate the histogram. The insets include the Gaussian envelope curve calculated using the mean and the standard deviation of the 1800 values. The number of bins used for the histogram generation was 40. The histograms exhibit symmetry with respect to the central value (mean), and they approximate the Gaussian envelope to some extent, confirming that the residual vibration introduced by *jerk* has a strong random component. None of the velocity profiles generated one or more dominant frequencies in the frequency domain. The PSD had higher amplitudes in the 40 to 50 Hz region than in other frequencies in the linear, quadratic, and logarithmic profiles. The logarithmic profile generated a PSD with a large amplitude that was distributed along various frequency regions. Thus, the residual vibrations were random. The PSD also suggests that randomness dominates the vertical acceleration behavior.

G. SPEED PROFILES ADVANTAGES AND DISADVANTAGES

A comparison between the generation of the linear, quadratic, sinusoidal, and logarithmic profiles under the same conditions of θ_f, θ_s, f_{max}, f_{min}, and P_N shown in Table 1 reveals some relevant points. The logarithmic was approximately 10 %, 12 %, and 18 % faster than the linear, sinusoidal, and quadratic, respectively. When several movements are involved in an automated task, this reduction in time is an

advantage. However, the choice of the speed profile not only depends on its duration. Other aspects, such as the nature of the mechanical system and the type of the transported material should be considered. For example, when moving liquids that do not have to be spilled, the best option is a sinusoidal profile, which is faster than the parabolic profile and slightly slower than the linear one, but with an angular acceleration that temporally evolves much more smoothly, which introduces fewer residual vibrations.

VII. CONCLUSION

In this study, we presented an algorithm for calculating angular position values in terms of time, which led to the generation of a quasi-linear, quadratic, sinusoidal, or logarithmic velocity profile in the regions of acceleration or deceleration when applied to a stepper motor.

The values of the position versus time can be derived numerically to obtain the angular velocity and acceleration curves in terms of time. Knowledge of the kinematic variables through simulation would allow the design of angular movements without the need to conduct experiments with a consequent saving in time.

The algorithm requires few resources and is suitable for devices with limited computing power, such as microcontrollers. In this study, we used the Arduino and Micropython platforms for the programming. The codes for the microcontrollers are presented in the appendices.

The algorithm uses an equation related to the proportionality between the motor's angular velocity and the frequency of the pulses as a starting point. Therefore, we experimentally characterized this relationship with two bipolar stepper motors, NEMA 17 and NEMA 23. The data exhibit linear behavior for frequencies below 700 Hz. Operating the motors at higher frequencies can cause errors in angular positioning. Therefore, it is advisable to use a speed profile.

The angular position in terms of the time generated by the algorithm, is very similar to the measurements made using an optical encoder.

We applied the velocity profiles to the rotation stage. In this mechanical device, the simulated speed profiles approximate the angular velocity measurements of the MEMS gyroscope.

Applying a logarithmic velocity profile can produce significant time savings in complex automation tasks that involve multiple angular movements. For example, when velocity profiles are generated with the parameters shown in Table 1, a time reduction of approximately 10 % for linear and approximately 18 % for quadratic is achieved.

We measured a_t and a_z with an MPU6050 MEMS accelerometer using a fixed bar at the base of the rotation stage. The tangential acceleration approximated the time evolution of α obtained through the simulation. The vertical acceleration shifted from its baseline of approximately 10.04 m/s^2 during the application of the velocity profiles owing to residual vibration. The standard deviation of a_z for the logarithmic profile in the time domain was four 4 larger than the σ of the sinusoidal profile. Therefore, the former introduced more residual vibrations than the latter. Analysis of the frequency domain through PSD indicated that the noise was mainly random.

APPENDIX A
HEADER AND CLASS FILES FOR GENERATING THE SPEED PROFILES

This appendix shows the header file for the class SpeedMotorProfile and the.cpp source file written with the C variant of the Arduino platform, aimed at generating the velocity profiles.

A. HEADER FILE

```
#ifndef SpeedMotorProfile_h
#define SpeedMotorProfile_h
#include "Arduino.h"
class SpeedMotorProfile{
  public:
    SpeedMotorProfile(int dirPin, int stepPin);
    int steps(float angle, int steps_per_revolution);
  void linear(int N, int pN, int minPeriod, int maxPeriod
     , int direction);
  void quadratic(int N, int pN, int minPeriod, int
     maxPeriod, int direction);
  void sinusoidal(int N, int pN, int minPeriod, int
     maxPeriod, int direction);
  void logarithmic(int N, int pN, int minPeriod, int
     maxPeriod, int direction);
  private:
    int _dirPin;
    int _stepPin;
    int _maxPeriod;
    int _minPeriod;
};
#endif
```

B. SOURCE FILE

```
#include "Arduino.h"
#include "math.h"
#include "SpeedMotorProfile.h"

int _steps = 0;
int _maxPeriod = 0;
int _minPeriod = 0;

SpeedMotorProfile::SpeedMotorProfile(int dirPin, int
     stepPin){
  pinMode(dirPin, OUTPUT);
  pinMode(stepPin,OUTPUT);
  _dirPin = dirPin;
  _stepPin = stepPin;
}//Constructor

int SpeedMotorProfile::steps(float angle, int
     steps_per_rev){
  float result = (steps_per_rev/360.0)*angle;
  return (int)result;
}//SpeedMotorProfile::steps

void SpeedMotorProfile::linear(int N, int pN, int
     minPeriod, int maxPeriod, int direction){
  _maxPeriod = maxPeriod;
  _minPeriod = minPeriod;
  int steps = N;
  float fMin = 1.0/(maxPeriod*1e-6);
  float fMax = 1.0/(minPeriod*1e-6);
  float a0 = (float)(fMax-fMin)/(pN-1);
  float a1 = (float)(fMin-fMax)/pN;
  float tempFreq = 0.0;
  int tempPeriod = 0;
if (direction == 0){
  digitalWrite(_dirPin,LOW);
}
else{
  digitalWrite(_dirPin,HIGH);
}
```

```
    if (steps < 20){
      for (int i=1; i <steps + 1; i++){
        digitalWrite(_stepPin,HIGH);
        delayMicroseconds((int)(_maxPeriod/2));
        digitalWrite(_stepPin,LOW);
        delayMicroseconds((int)(_maxPeriod/2));
      }//for (int i=1; i <steps + 1; i++)
    }// if (steps < 20)
    else{
      for (int i = 1; i < steps + 1; i++){
        if (i <= pN){
        tempFreq = a0 * (i-1) + fMin;
        tempPeriod =  (int)((1.0/tempFreq)*1e6);
        digitalWrite(_stepPin,HIGH);
        delayMicroseconds((int)(tempPeriod/2));
        digitalWrite(_stepPin,LOW);
        delayMicroseconds((int)(tempPeriod/2));
        }//if (i <= pN)
        else if (i <(steps- pN)){
          digitalWrite(_stepPin,HIGH);
        delayMicroseconds((int)(_minPeriod/2));
        digitalWrite(_stepPin,LOW);
        delayMicroseconds((int)(_minPeriod/2));
        }//else if (i <(steps- pN)){
        else{
        tempFreq = a1 * (i-N) + fMin;
        tempPeriod =  (int)((1.0/tempFreq)*1e6);
        digitalWrite(_stepPin,HIGH);
        delayMicroseconds((int)(tempPeriod/2));
        digitalWrite(_stepPin,LOW);
        delayMicroseconds((int)(tempPeriod/2));
        } //else
      }//for (int i = 1; i < steps + 1; i++)
    }//else
  }//SpeedMotorProfile::linear

  void SpeedMotorProfile::quadratic(int N, int pN, int
       minPeriod, int maxPeriod, int direction){
    _maxPeriod = maxPeriod;
    _minPeriod = minPeriod;

    int steps = N;
    float fMin = 1.0/(maxPeriod*1e-6);
    float fMax = 1.0/(minPeriod*1e-6);
    float a2 = (float)((fMax-fMin)/(-1.0+pN*pN));
    float a3 = (float)(fMin-a2);
    float a4 = (float)((fMax-fMin)/(pN*pN));
    float tempFreq = 0.0;
    int tempPeriod = 0;


    if (direction == 0){
      digitalWrite(_dirPin,LOW);
    }
    else{
      digitalWrite(_dirPin,HIGH);
    }

    if (steps < 20){
      for (int i=1; i <steps + 1; i++){
        digitalWrite(_stepPin,HIGH);
        delayMicroseconds((int)(_maxPeriod/2));
        digitalWrite(_stepPin,LOW);
        delayMicroseconds((int)(_maxPeriod/2));
      }//for (int i=1; i <steps + 1; i++)
    }// if (steps < 20)
    else{
      for (int i = 1; i < steps + 1; i++){
        if (i <= pN){
        tempFreq = a2*(i*i) + a3;
        tempPeriod =  (int)((1.0/tempFreq)*1e6);
        digitalWrite(_stepPin,HIGH);
        delayMicroseconds((int)(tempPeriod/2));
        digitalWrite(_stepPin,LOW);
        delayMicroseconds((int)(tempPeriod/2));
        }//if (i <= pN)
      else if (i <(steps- pN)){
        digitalWrite(_stepPin,HIGH);
      delayMicroseconds((int)(_minPeriod/2));
      digitalWrite(_stepPin,LOW);
      delayMicroseconds((int)(_minPeriod/2));
      }//else if (i <(steps- pN)){
        else{
        tempFreq = a4*(i-N)*(i-N) + fMin;
        tempPeriod =  (int)((1.0/tempFreq)*1e6);
        digitalWrite(_stepPin,HIGH);
        delayMicroseconds((int)(tempPeriod/2));
        digitalWrite(_stepPin,LOW);
        delayMicroseconds((int)(tempPeriod/2));
        } //else
      }//for (int i = 1; i < steps + 1; i++)
    }//else
  }//SpeedMotorProfile::quadratic

  void SpeedMotorProfile::sinusoidal(int N, int pN, int
       minPeriod, int maxPeriod, int direction){
    _maxPeriod = maxPeriod;
    _minPeriod = minPeriod;
    int steps = N;
    float fMin = 1.0/(maxPeriod*1e-6);
    float fMax = 1.0/(minPeriod*1e-6);
    float a5 = (fMin - fMax)/2.0;
    float a6 = (fMin + fMax)/2.0;
    float tempFreq = 0.0;
    int tempPeriod = 0;

    if (direction == 0){
      digitalWrite(_dirPin,LOW);
    }
    else{
      digitalWrite(_dirPin,HIGH);
    }

    if (steps < 20){
        for (int i=1; i <steps + 1 ; i++){
          digitalWrite(_stepPin,HIGH);
          delayMicroseconds((int)(_maxPeriod/2));
          digitalWrite(_stepPin,LOW);
          delayMicroseconds((int)(_maxPeriod/2));
        }//for (int i=1; i <steps + 1; i++)
    }// if (steps < 20)

    else{
      for (int i = 1; i < steps + 1; i++){
        if (i <= pN){
          tempFreq = a5 * cos((M_PI*(i-1))/(pN-1)) + a6;
          tempPeriod =  (int)((1.0/tempFreq)*1e6);
          digitalWrite(_stepPin,HIGH);
          delayMicroseconds((int)(tempPeriod/2));
          digitalWrite(_stepPin,LOW);
          delayMicroseconds((int)(tempPeriod/2));
        }//if (i <= pN)
        else if (i <(steps- pN)){
          digitalWrite(_stepPin,HIGH);
          delayMicroseconds((int)(_minPeriod/2));
          digitalWrite(_stepPin,LOW);
          delayMicroseconds((int)(_minPeriod/2));
        }//else if (i <(steps- pN)){
        else{
          tempFreq = a5 * cos((M_PI*(i-N))/pN) + a6;
          tempPeriod =  (int)((1.0/tempFreq)*1e6);
          digitalWrite(_stepPin,HIGH);
          delayMicroseconds((int)(tempPeriod/2));
          digitalWrite(_stepPin,LOW);
          delayMicroseconds((int)(tempPeriod/2));
        } //else
      }//for (int i = 1; i < steps + 1; i++)
    }//else
  }//SpeedMotorProfile::sinusoidal

  void SpeedMotorProfile::logarithmic(int N, int pN, int
       minPeriod, int maxPeriod, int direction){
    _maxPeriod = maxPeriod;
    _minPeriod = minPeriod;
    int steps = N;
    float fMin = 1.0/(maxPeriod*1e-6);
    float fMax = 1.0/(minPeriod*1e-6);
    float a7 = (fMax - fMin)/log((float)pN);
    float a8 = (fMax - fMin)/log(1.0+(float)pN);
    float tempFreq = 0.0;
    int tempPeriod = 0;

    if (direction == 0){
      digitalWrite(_dirPin,LOW);
    }
```

```
    else{
      digitalWrite(_dirPin,HIGH);
    }

    if (steps < 20){
        for (int i=1; i <steps + 1 ; i++){
        digitalWrite(_stepPin,HIGH);
        delayMicroseconds((int)(_maxPeriod/2));
        digitalWrite(_stepPin,LOW);
        delayMicroseconds((int)(_maxPeriod/2));
      }//for (int i=1; i <steps + 1; i++)
    }// if (steps < 20)
    else{
      for (int i = 1; i < steps + 1; i++){
        if (i <= pN){
          tempFreq = a7 * log((float)i) + fMin;
          tempPeriod =  (int)((1.0/tempFreq)*1e6);
          digitalWrite(_stepPin,HIGH);
          delayMicroseconds((int)(tempPeriod/2));
          digitalWrite(_stepPin,LOW);
          delayMicroseconds((int)(tempPeriod/2));
        }//if (i <= pN)
        else if (i <(steps- pN)){
          digitalWrite(_stepPin,HIGH);
        delayMicroseconds((int)(_minPeriod/2));
        digitalWrite(_stepPin,LOW);
        delayMicroseconds((int)(_minPeriod/2));
        }//else if (i <(steps- pN)){
        else{
        tempFreq = a8 * log(float(-i+N+1)) + fMin;
        tempPeriod =  (int)((1.0/tempFreq)*1e6);
        digitalWrite(_stepPin,HIGH);
        delayMicroseconds((int)(tempPeriod/2));
        digitalWrite(_stepPin,LOW);
        delayMicroseconds((int)(tempPeriod/2));
        } //else
      }//for (int i = 1; i < steps + 1; i++)
    }//else
  }//SpeedMotorProfile::logarithmic
```

APPENDIX B
MICROPYTHON CLASS FOR GENERATING THE SPEED PROFILES

This appendix shows the code of the class SpeedMotorProfile written in Micropython, aimed at generating the velocity waveforms.

```
# -*- coding: utf-8 -*-
from machine import Pin
import time
from math import pi, cos, floor, log

class SpeedMotorProfile():
    def __init__(self,dir_pin,step_pin):
        self._dir_pin = Pin(dir_pin, Pin.OUT)
        self._step_pin = Pin(step_pin,Pin.OUT)
    def steps(self, angle, steps_per_revolution):
        """

        Parameters

        angle : float
            Final angle in degrees.
        steps_per_revolution : float
            Number of microsteps given by the driver.

        Returns

        steps: integer
            integer number of steps.

        """
        return (floor((steps_per_revolution/360.0)*angle)
    )
    def linear(self,N,pN,min_period,max_period,direction)
    :
        """

        Parameters

        N : integer
            Number of steps.
        pN : integer
            Number of points where the movement presents
            acceleration or deacceleration.
        min_period : integer
            Minimum period in microseconds.
        max_period : integer
            Maximum period in microseconds.
        direction : integer
            clockwise or counter clockwise direction,
            valid values are 0 or 1.

        Returns

        None.

        """
        self._dir_pin(int(direction))
        f_min = 1/(max_period*1e-6)
        f_max = 1/(min_period*1e-6)
        a0 = (f_max - f_min)/(pN - 1)
        a1 = (f_min - f_max)/pN
        if (N < 20):
            self._step_pin(1)
            time.sleep_us(int(max_period/2))
            self._step_pin(0)
            time.sleep_us(int(max_period/2))
        else:
            for i in range(1,N+1):
                if (i <= pN):
                    temp_freq = a0 * (i-1) + f_min
                    temp_period = int((1/temp_freq)*1e6)
                    temp_half_period = int(temp_period/2)
                    self._step_pin(1)
                    time.sleep_us(temp_half_period)
                    self._step_pin(0)
                    time.sleep_us(temp_half_period)
                elif (i <(N-pN)):
                    self._step_pin(1)
                    time.sleep_us(int(min_period/2))
                    self._step_pin(0)
                    time.sleep_us(int(min_period/2))
                else:
                    temp_freq = a1 * (i-N) + f_min
                    temp_period = int((1/temp_freq)*1e6)
                    temp_half_period = int(temp_period/2)
                    self._step_pin(1)
                    time.sleep_us(temp_half_period)
                    self._step_pin(0)
                    time.sleep_us(temp_half_period)

    def quadratic(self,N,pN,min_period,max_period,
     direction):
        """

        Parameters

        N : integer
            Number of steps.
        pN : integer
            Number of points where the movement presents
            acceleration or deacceleration.
        min_period : integer
            Minimum period in microseconds.
        max_period : integer
            Maximum period in microseconds.
        direction : integer
            clockwise or counter clockwise direction,
            valid values are 0 or 1.

        Returns

        None.

        """
        self._dir_pin(int(direction))
        f_min = 1/(max_period*1e-6)
        f_max = 1/(min_period*1e-6)
        a2 = (f_max - f_min)/(- 1+pN**2)
        a3 = f_min - a2
        a4 = (f_max - f_min)/(pN**2)
        if (N < 20):
```

```
            self._step_pin(1)
            time.sleep_us(int(max_period/2))
            self._step_pin(0)
            time.sleep_us(int(max_period/2))
        else:
            for i in range(1,N+1):
                if (i <= pN):
                    temp_freq = a2 * i**2  + a3
                    temp_period = int((1/temp_freq)*1e6)
                    temp_half_period = int(temp_period/2)
                    self._step_pin(1)
                    time.sleep_us(temp_half_period)
                    self._step_pin(0)
                    time.sleep_us(temp_half_period)
                elif (i <(N-pN)):
                    self._step_pin(1)
                    time.sleep_us(int(min_period/2))
                    self._step_pin(0)
                    time.sleep_us(int(min_period/2))
                else:
                    temp_freq = a4 * (i-N)**2 + f_min
                    temp_period = int((1/temp_freq)*1e6)
                    temp_half_period = int(temp_period/2)
                    self._step_pin(1)
                    time.sleep_us(temp_half_period)
                    self._step_pin(0)
                    time.sleep_us(temp_half_period)

    def sinusoidal(self,N,pN,min_period,max_period,
     direction):
        """

        Parameters

        N : integer
            Number of steps.
        pN : integer
            Number of points where the movement presents
            acceleration or deacceleration.
        min_period : integer
            Minimum period in microseconds.
        max_period : integer
            Maximum period in microseconds.
        direction : integer
            clockwise or counter clockwise direction,
            valid values are 0 or 1.

        Returns

        None.

        """
        self._dir_pin(int(direction))
        f_min = 1/(max_period*1e-6)
        f_max = 1/(min_period*1e-6)
        a5 = (f_min - f_max)/2
        a6 = (f_min + f_max)/2
        if (N < 20):
            self._step_pin(1)
            time.sleep_us(int(max_period/2))
            self._step_pin(0)
            time.sleep_us(int(max_period/2))
        else:
            for i in range(1,N+1):
                if (i <= pN):
                    temp_freq = a5 * cos((pi*(i-1))/(pN
-1)) + a6
                    temp_period = int((1/temp_freq)*1e6)
                    temp_half_period = int(temp_period/2)
                    self._step_pin(1)
                    time.sleep_us(temp_half_period)
                    self._step_pin(0)
                    time.sleep_us(temp_half_period)
                elif (i <(N-pN)):
                    self._step_pin(1)
                    time.sleep_us(int(min_period/2))
                    self._step_pin(0)
                    time.sleep_us(int(min_period/2))
                else:
                    temp_freq = a5 * cos((pi*(i-N))/pN) +
 a6
                    temp_period = int((1/temp_freq)*1e6)
                    temp_half_period = int(temp_period/2)
                    self._step_pin(1)
                    time.sleep_us(temp_half_period)
                    self._step_pin(0)
                    time.sleep_us(temp_half_period)
    def logarithmic(self,N,pN,min_period,max_period,
     direction):
        """

        Parameters

        N : integer
            Number of steps.
        pN : integer
            Number of points where the movement presents
            acceleration or deacceleration.
        min_period : integer
            Minimum period in microseconds.
        max_period : integer
            Maximum period in microseconds.
        direction : integer
            clockwise or counter clockwise direction,
            valid values are 0 or 1.

        Returns

        None.

        """
        self._dir_pin(int(direction))
        f_min = 1/(max_period*1e-6)
        f_max = 1/(min_period*1e-6)
        a7 = (f_max - f_min)/log(pN)
        a8 = (f_max - f_min)/log(pN + 1)
        if (N < 20):
            self._step_pin(1)
            time.sleep_us(int(max_period/2))
            self._step_pin(0)
            time.sleep_us(int(max_period/2))
        else:
            for i in range(1,N+1):
                if (i <= pN):
                    temp_freq = a7*log(i) + f_min
                    temp_period = int((1/temp_freq)*1e6)
                    temp_half_period = int(temp_period/2)
                    self._step_pin(1)
                    time.sleep_us(temp_half_period)
                    self._step_pin(0)
                    time.sleep_us(temp_half_period)
                elif (i <(N-pN)):
                    self._step_pin(1)
                    time.sleep_us(int(min_period/2))
                    self._step_pin(0)
                    time.sleep_us(int(min_period/2))
                else:
                    temp_freq = a8 * log(-i+N+1) + f_min
                    temp_period = int((1/temp_freq)*1e6)
                    temp_half_period = int(temp_period/2)
                    self._step_pin(1)
                    time.sleep_us(temp_half_period)
                    self._step_pin(0)
                    time.sleep_us(temp_half_period)
```

ACKNOWLEDGMENT

The authors would like to thank Secretaría de Investigación y Posgrado del Instituto Politécnico Nacional (SIP-IPN) and Comisión de Operación y Fomento de Actividades Académicas (COFAA-IPN) for the aid provided to do this work. Jorge Fonseca-Campos would like to thank Osvaldo Valentín Sánchez Lemus for helping them to acquire data during some experiments and to Centro de Investigación y de Estudios Avanzados (CINVESTAV), Departmento de Biotecnología y Bioingeniería, for the aid provided to do this work.

REFERENCES

[1] C. McElheny, D. Hayes, and R. Devireddy, "Design and fabrication of a low-cost three-dimensional bioprinter," *J. Med. Devices*, vol. 11, no. 4, Dec. 2017, Art. no. 0410019, doi: 10.1115/1.4037259.

[2] J. Lee, K. E. Kim, S. Bang, I. Noh, and C. Lee, "A desktop multi-material 3D bio-printing system with open-source hardware and software," *Int. J. Precis. Eng. Manuf.*, vol. 18, no. 4, pp. 605–612, Apr. 2017, doi: 10.1007/s12541-017-0072-x.

[3] J. A. Reid, P. A. Mollica, G. D. Johnson, R. C. Ogle, R. D. Bruno, and P. C. Sachs, "Accessible bioprinting: Adaptation of a low-cost 3D-printer for precise cell placement and stem cell differentiation," *Biofabrication*, vol. 8, no. 2, Jun. 2016, Art. no. 025017, doi: 10.1088/1758-5090/8/2/025017.

[4] L. Ličev, M. Krumnikl, J. Škuta, M. Babiuch, and R. Farana, "Advances in the development of an imaging device for plaque measurement in the area of the carotid artery," *Biotechnol. Biotechnological Equip.*, vol. 28, no. 2, pp. 355–359, Jul. 2014, doi: 10.1080/13102818.2014.910362.

[5] C. Wang, H. Zhu, Z. Chen, Y. Deng, E. Su, and P. Xiao, "Control methods of mechanical arms motion for automatic nucleic acid detection system based on magnetic nanoparticles," *J. Nanosci. Nanotechnol.*, vol. 16, no. 12, pp. 12455–12459, Dec. 2016, doi: 10.1166/jnn.2016.13761.

[6] K. Z. Zaferullah, R. Bansode, S. N. Pethe, M. Vidwans, and K. Dsouza, "Speed control of stepper motor for collimator jaws positioning based on FPGA implementation," in *Proc. Int. Conf. Circuits, Syst., Commun. Inf. Technol. Appl. (CSCITA)*, Mumbai, India, Apr. 2014, pp. 353–357, doi: 10.1109/CSCITA.2014.6839286.

[7] A. J. Blauch, M. Bodson, and J. Chiasson, "High-speed parameter estimation of stepper motors," *IEEE Trans. Control Syst. Technol.*, vol. 1, no. 4, pp. 270–279, Dec. 1993, doi: 10.1109/87.260272.

[8] M. Y. Stoychitch, "Generate stepper motor linear speed profile in real time," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 294, no. 1, 2018, Art. no. 012055, doi: 10.1088/1757-899X/294/1/01205.

[9] F. Betin, D. Pinchon, and G.-A. Capolino, "Fuzzy logic applied to speed control of a stepping motor drive," *IEEE Trans. Ind. Electron.*, vol. 47, no. 3, pp. 610–622, Jun. 2000, doi: 10.1109/41.847902.

[10] N. Le, J. Cho, and J. Jeon, "Application of velocity profile generation and closed-loop control in step motor control system," in *Proc. SICE-ICASE Int. Joint Conf.*, 2006, pp. 3593–3598, doi: 10.1109/SICE.2006.314747.

[11] W. Bangji, L. Qingxiang, Z. Lei, Z. Yanrong, L. Xiangqiang, and Z. Jianqiong, "Velocity profile algorithm realization on FPGA for stepper motor controller," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Aug. 2011, pp. 6072–6075, doi: 10.1109/AIMSEC.2011.6009864.

[12] Z. Qiu, S. Shi, X. Li, L. Zhang, and W. Wu, "Implementation of motion control technique for stepper motor translation stages in online detection system," in *Proc. Int. Conf. Mech. Sci., Electr. Eng. Comput. (MEC)*, Jilin, China, Aug. 2011, pp. 71–75, doi: 10.1109/MEC.2011.6025403.

[13] C.-K. Lai, B.-W. Lin, H.-Y. Lai, and G.-Y. Chen, "FPGA-based hybrid stepper motor drive system design by variable structure control," *Actuators*, vol. 10, no. 6, p. 113, May 2021, doi: 10.3390/act10060113.

[14] D. Isobe, S. Kawai, and T. Nguyen–Van, "A digital speed profile generator for stepper motor," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Las Vegas, NV, USA, Jan. 2022, pp. 1–6, doi: 10.1109/ICCE53296.2022.9730578.

[15] J. Divic, J. Đuric, and K. Vrancic, "Microcontroller implementation of dynamically adaptable control of stepper motor with continuous second derivative of speed curve," in *Proc. 37th Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, Opatija, Croatia, May 2014, pp. 146–149, doi: 10.1109/MIPRO.2014.6859550.

[16] S. Mandal, S. K. Singh, S. Mandal, A. Kumar, and Nagahanumaiah, "Residual jerk reduction in precision positioning stages using sliding microstep-based switching," *J. Control, Autom. Electr. Syst.*, vol. 25, no. 3, pp. 311–318, Jun. 2014, doi: 10.1007/s40313-013-0099-x.

[17] R. R. Torres-Camacho, H. R. Frias-Fonseca, J. R. Rivera-Guillen, and M. A. I. Manzano, "Computationally-efficient algorithm for applying motion-controlled kinematics in stepper motors," in *Proc. IEEE Int. Autumn Meeting Power, Electron. Comput. (ROPEC)*, Ixtapa, Mexico, Nov. 2019, pp. 1–5, doi: 10.1109/ROPEC48299.2019.9057095.

[18] V. Montalvo, A. A. Estévez-Bén, J. Rodríguez-Reséndiz, G. Macias-Bobadilla, J. D. Mendiola-Santíbañez, and K. A. Camarillo-Gómez, "FPGA-based architecture for sensing power consumption on parabolic and trapezoidal motion profiles," *Electronics*, vol. 9, no. 8, p. 1301, Aug. 2020, doi: 10.3390/electronics9081301.

[19] H. Li, Z. Gong, W. Lin, and T. Lippa, "A new motion control approach for jerk and transient vibration suppression," in *Proc. IEEE Int. Conf. Ind. Informat.*, Singapore, Aug. 2006, pp. 676–681, doi: 10.1109/INDIN.2006.275642.

[20] A. Malinowski and H. Yu, "Comparison of embedded system design for industrial applications," *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, pp. 244–254, May 2011, doi: 10.1109/TII.2011.2124466.

[21] K. Dokic, B. Radisic, and M. Cobovic, "MicroPython or Arduino C for ESP32-efficiency for neural network edge devices," in *Proc. ISICS*, 2020, pp. 33–43, doi: 10.1007/978-3-030-43364-2_4.

[22] I. Plauska, A. Liutkevičius, and A. Janavičiūtė, "Performance evaluation of C/C++, MicroPython, rust and TinyGo programming languages on ESP32 microcontroller," *Electronics*, vol. 12, no. 1, p. 143, Dec. 2022, doi: 10.3390/electronics12010143.

[23] V. M. Ionescu and F. M. Enescu, "Investigating the performance of MicroPython and c on ESP32 and STM32 microcontrollers," in *Proc. IEEE 26th Int. Symp. Design Technol. Electron. Packag. (SIITME)*, Pitesti, Romania, Oct. 2020, pp. 234–237, doi: 10.1109/SIITME50350.2020.9292199.

[24] S. Palani, "Sevomotors," in *Control System Engineering*, 2nd ed. New Delhi, India: Tata, 2010, ch. 10, sec. 10.3.7, pp. 10–13.

[25] D.-M. Tsay and C.-F. Lin, "Asymmetrical inputs for minimizing residual response," in *Proc. IEEE Int. Conf. Mechatronics (ICM)*, Taipei, Taiwan, Jul. 2005, pp. 235–240, doi: 10.1109/ICMECH.2005.1529259.

[26] K. D. Nguyen, T.-C. Ng, and I.-M. Chen, "On algorithms for planning S-curve motion profiles," *Int. J. Adv. Robotic Syst.*, vol. 5, no. 1, p. 11, Mar. 2008, doi: 10.5772/5652.

[27] P. J. Clarkson and P. P. Acarnley, "Closed-loop control of stepping motor systems," *IEEE Trans. Ind. Appl.*, vol. 24, no. 4, pp. 685–691, Jul. 1988, doi: 10.1109/28.6122.

[28] M. Scarpino, "Preliminary concepts," in *Motors for Makers: A Guide to Steppers, Servos, and Other Electrical Machines*, 1st ed. Indianapolis, IN, USA: Que Publication, 2015, ch. 2, pp. 13–18.

[29] J.-W. Yoo, J.-H. Kim, J.-H. Kim, and E. Kim, "Design of a variable reference current controller for micro-stepping motor based on vibration and missing step characteristic data," *Int. J. Precis. Eng. Manuf.*, vol. 24, no. 5, pp. 877–886, Mar. 2023, doi: 10.1007/s12541-023-00789-5.

[30] V. M. Arevalo, "Sinusoidal velocity profiles for motion control," in *Proc. ASPE Contrl. Precis. Syst.*, Philadelphia, PA, USA, 2001, pp. 18–20.

[31] J. Wook Jeon and Y. Youl Ha, "A generalized approach for the acceleration and deceleration of industrial robots and CNC machine tools," *IEEE Trans. Ind. Electron.*, vol. 47, no. 1, pp. 133–139, Feb. 2000, doi: 10.1109/41.824135.

[32] M. Bennaya, M. A. El-Noor, and S. M. Abdelkhalek, "Generic solution for generating towing tank carriage linear speed profile using stepper motor," in *Proc. 12th Int. Conf. Electr. Eng. (ICEENG)*, Cairo, Egypt, Jul. 2020, pp. 476–481, doi: 10.1109/ICEENG45378.2020.9171712.

[33] H.-J. Heo, Y. Son, and J.-M. Kim, "A trapezoidal velocity profile generator for position control using a feedback strategy," *Energies*, vol. 12, no. 7, p. 1222, Mar. 2019, doi: 10.3390/en12071222.

[34] M. Reyes-García, O. Sergiyenko, M. Ivanov, L. Lindner, J. C. Rodríguez-Quiñonez, D. Hernandez-Balbuena, W. Flores-Fuentes, V. Tyrsa, L. O. Moreno-Ahedo, and F. N. Murrieta-Rico, "Defining the final angular position of DC motor shaft using a trapezoidal trajectory profile," in *Proc. IEEE 28th Int. Symp. Ind. Electron. (ISIE)*, Vancouver, BC, Canada, Jun. 2019, pp. 1694–1699, doi: 10.1109/ISIE.2019.8781093.

[35] Z. Yu, C. Han, and M. Haihua, "A novel approach of tuning trapezoidal velocity profile for energy saving in servomotor systems," in *Proc. 34th Chin. Control Conf. (CCC)*, Hangzhou, China, Jul. 2015, pp. 4412–4417, doi: 10.1109/ChiCC.2015.7260323.

[36] H. Chen, H. Mu, and Y. Zhu, "Real-time generation of trapezoidal velocity profile for minimum energy consumption and zero residual vibration in servomotor systems," in *Proc. Amer. Control Conf. (ACC)*, Boston, MA, USA, Jul. 2016, pp. 2223–2228, doi: 10.1109/ACC.2016.7525248.

[37] D. O. Carrica, S. A. González, and M. Benedetti, "A high speed velocity control algorithm of multiple stepper motors," *Mechatronics*, vol. 14, no. 6, pp. 675–684, Jul. 2004, doi: 10.1016/j.mechatronics.2004.01.006.

[38] C. Lewin, *Mathematics of Motion Control Profiles*. Boston, MA, USA: Performance Motion Devices, Inc. (PMD), 2007. [Online]. Available: https://www.pmdcorp.com/resources/type/articles/get/mathematics-of-motion-control-profiles-article

[39] P. H. Meckl and P. B. Arestides, "Optimized s-curve motion profiles for minimum residual vibration," in *Proc. Amer. Control Conf. (ACC)*, Philadelphia, PA, USA, Jun. 1998, pp. 2627–2631, doi: 10.1109/ACC.1998.688324.

[40] H. Z. Li, Z. M. Gong, W. Lin, and T. Lippa, "Motion profile planning for reduced jerk and vibration residuals," *SIMTech Tech. Rep.*, vol. 8, no. 1, pp. 32–37, Jan. 2007, doi: 10.13140/2.1.4211.2647.

[41] H. Li, M. D. Le, Z. M. Gong, and W. Lin, "Motion profile design to reduce residual vibration of high-speed positioning stages," *IEEE/ASME Trans. Mechatronics*, vol. 14, no. 2, pp. 264–269, Apr. 2009, doi: 10.1109/TMECH.2008.2012160.

[42] S.-C. Shin, C.-H. Choi, J.-H. Youm, T.-K. Lee, and C.-Y. Won, "Position control of PMSM using jerk-limited trajectory for torque ripple reduction in robot applications," in *Proc. IECON 38th Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2012, pp. 2400–2405, doi: 10.1109/IECON.2012.6388868.

[43] C.-W. Ha, K.-H. Rew, and K.-S. Kim, "A complete solution to asymmetric S-curve motion profile: Theory & experiments," in *Proc. Int. Conf. Control, Autom. Syst.*, Oct. 2008, pp. 2845–2849, doi: 10.1109/ICCAS.2008.4694244.

[44] V. M. N. Passaro, A. Cuccovillo, L. Vaiani, M. De Carlo, and C. E. Campanella, "Gyroscope technology and applications: A review in the industrial perspective," *Sensors*, vol. 17, no. 10, p. 2284, Oct. 2017, doi: 10.3390/s17102284.

[45] S. Han, Z. Meng, O. Omisore, T. Akinyemi, and Y. Yan, "Random error reduction algorithms for MEMS inertial sensor accuracy Improvement—A review," *Micromachines*, vol. 11, no. 11, p. 1021, Nov. 2020, doi: 10.3390/mi11111021.

[46] D. Wang and H. Lin, "A new MEMS gyroscope drift suppression method for the low-cost untwisting spin platform," *IEEJ Trans. Electr. Electron. Eng.*, vol. 12, no. 2, pp. 262–268, Mar. 2017, doi: 10.1002/tee.22373.

[47] W. Chen, G. Xin, and F. Wang, "Research of electronic image stabilization technology based on MEMS gyroscope," in *Proc. IEEE 2nd Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Chengdu, China, Dec. 2017, pp. 1680–1683, doi: 10.1109/ITNEC.2017.8285081.

[48] X. Xinjilefu, S. Feng, and C. G. Atkeson, "A distributed MEMS gyro network for joint velocity estimation," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Stockholm, Sweden, May 2016, pp. 1879–1884, doi: 10.1109/ICRA.2016.7487334.

[49] F. Nie, H. Wei, F. Li, and J. Jiang, "Design of motion control system for fire fighting robot," *J. Physics: Conf. Ser.*, vol. 1635, no. 1, Nov. 2020, Art. no. 012059, doi: 10.1088/1742-6596/1635/1/012059.

[50] K. R. Ooi, M. A. A. Rosli, A. R. A. Latiff, W. A. F. W. Othman, S. S. N. Alhady, and A. A. A. Wahab, "Design of hoeckens linkage based walking robot with MPU6050 IMU as navigation sensor," *J. Phys., Conf. Ser.*, vol. 1969, no. 1, Jul. 2021, Art. no. 012004, doi: 10.1088/1742-6596/1969/1/012004.

[51] I. Rifajar and A. Fadlil, "The path direction control system for lanange jagad dance robot using the MPU6050 gyroscope sensor," *Int. J. Robot. Control Syst.*, vol. 1, no. 1, pp. 27–40, Feb. 2021, doi: 10.31763/ijrcs.v1i1.225.

[52] Z. He, Y. Chen, Z. Shen, E. Huang, S. Li, Z. Shao, and Q. Wang, "Ard-μ-copter: A simple open source quadcopter platform," in *Proc. 11th Int. Conf. Mobile Ad-Hoc Sensor Netw. (MSN)*, Shenzhen, China, Dec. 2015, pp. 158–164, doi: 10.1109/MSN.2015.9.

[53] A. Tagay, A. Omar, and M. H. Ali, "Development of control algorithm for a quadcopter," *Proc. Comput. Sci.*, vol. 179, pp. 242–251, Jan. 2021, doi: 10.1016/j.procs.2021.01.003.

[54] B. Dziadak and M. Czumak, "Personal fall detector using MEMS sensors," in *Proc. 23rd Int. Conf. Comput. Problems Electr. Eng. (CPEE)*, Zuberec, Slovakia, Sep. 2022, pp. 1–4, doi: 10.1109/CPEE56060.2022.9919414.

[55] V. Popelka, "A self stabilizing platform," in *Proc. 15th Int. Carpathian Control Conf. (ICCC)*, Velke Karlovice, Czech Republic, May 2014, pp. 458–462, doi: 10.1109/CarpathianCC.2014.6843648.

[56] V. Vidya, P. Poornachandran, V. G. Sujadevi, and M. M. Dharmana, "Imu sensor based self stabilizing cup for elderly and parkinsonism," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Udupi, India, Sep. 2017, pp. 2264–2269, doi: 10.1109/ICACCI.2017.8126183.

[57] D. Bülz, R. Forke, K. Hiller, S. Weidlich, S. Hahn, A. Shaporin, D. Wünsch, S. Konietzka, T. Motl, D. Billep, and H. Kuhn, "North finding with a single double mass MEMS gyroscope—A performance demonstrator," in *Proc. DGON Inertial Sensors Syst. (ISS)*, Braunschweig, Germany, Sep. 2021, pp. 1–16, doi: 10.1109/ISS52949.2021.9619804.

[58] X. Zhang, Z. Fu, and J. Liu, "A novel rotor position detection method of PMSM based on MEMS inertial sensor," in *Proc. 20th Int. Conf. Electr. Mach. Syst. (ICEMS)*, Sydney, NSW, Australia, Aug. 2017, pp. 1–4, doi: 10.1109/ICEMS.2017.8055940.

[59] M. S. Mustafa, P. Cheng, and B. Oelmann, "Stator-free low angular speed sensor based on a MEMS gyroscope," *IEEE Trans. Instrum. Meas.*, vol. 63, no. 11, pp. 2591–2598, Nov. 2014, doi: 10.1109/TIM.2013.2283153.

[60] W. Decheng, Z. Jingwei, and D. Yufei, "One design of motor speed control system based on MEMS gyroscope," in *Proc. 18th Int. Conf. Electr. Mach. Syst. (ICEMS)*, Pattaya, Thailand, Oct. 2015, pp. 1297–1300, doi: 10.1109/ICEMS.2015.7385239.

[61] Y. Rong, Q. Wang, S. Lu, G. Li, Y. Lu, and J. Xu, "Improving attitude detection performance for spherical motors using a MEMS inertial measurement sensor," *IET Electr. Power Appl.*, vol. 13, no. 2, pp. 198–205, Feb. 2019, doi: 10.1049/iet-epa.2018.5195.

[62] G. S. Maruthi and V. Hegde, "Mathematical analysis of unbalanced magnetic pull and detection of mixed air gap eccentricity in induction motor by vibration analysis using MEMS accelerometer," in *Proc. IEEE 1st Int. Conf. Condition Assessment Techn. Electr. Syst. (CATCON)*, Kolkata, India, Dec. 2013, pp. 207–212, doi: 10.1109/CATCON.2013.6737499.

[63] J. V. O. Rodrigues, M. P. G. Pedroso, F. F. B. Silva, and R. G. Leão Junior, "Performance evaluation of accelerometers ADXL345 and MPU6050 exposed to random vibrational input," *Res., Soc. Develop.*, vol. 10, no. 15, Nov. 2021, Art. no. e286101523082, doi: 10.33448/rsd-v10i15.23082.

[64] N. V. S. Shankar, V. S. N. V. Ramana, A. Sravani, P. Sreenivasulu, and K. S. Vikas, "IoT for vibration measurement in engineering research," *Mater. Today, Proc.*, vol. 59, pp. 1792–1796, Jan. 2022, doi: 10.1016/j.matpr.2022.04.380.

[65] M. Varanis, A. Silva, A. Mereles, and R. Pederiva, "MEMS accelerometers for mechanical vibrations analysis: A comprehensive review with applications," *J. Brazilian Soc. Mech. Sci. Eng.*, vol. 40, no. 11, pp. 1–18, Oct. 2018, doi: 10.1007/s40430-018-1445-5.

[66] A. Rossi, G. Bocchetta, F. Botta, and A. Scorza, "Accuracy characterization of a MEMS accelerometer for vibration monitoring in a rotating framework," *Appl. Sci.*, vol. 13, no. 8, p. 5070, Apr. 2023, doi: 10.3390/app13085070.

[67] V. P. Raj, K. Natarajan, and Sri. T. G. Girikumar, "Induction motor fault detection and diagnosis by vibration analysis using MEMS accelerometer," in *Proc. Int. Conf. Emerg. Trends Commun., Control, Signal Process. Comput. Appl. (C2SPCA)*, Bangalore, India, Oct. 2013, pp. 1–6, doi: 10.1109/C2SPCA.2013.6749391.

[68] G. S. Maruthi and V. Hegde, "Application of MEMS accelerometer for detection and diagnosis of multiple faults in the roller element bearings of three phase induction motor," *IEEE Sensors J.*, vol. 16, no. 1, pp. 145–152, Jan. 1, 2016, doi: 10.1109/JSEN.2015.2476561.

[69] L. A. Dos Santos Pedotti, R. M. Zago, and F. Fruett, "Fault diagnostics in rotary machines through spectral vibration analysis using low-cost MEMS devices," *IEEE Instrum. Meas. Mag.*, vol. 20, no. 6, pp. 39–44, Dec. 2017, doi: 10.1109/MIM.2017.8121950.

[70] J. Chen, "Vibration condition measure instrument of motor using MEMS accelerometer," *AIP Conf. Proc.*, vol. 1955, no. 1, Apr. 2018, Art. no. 030014, doi: 10.1063/1.5033613.

[71] M. Zekveld and G. P. Hancke, "Vibration condition monitoring using machine learning," in *Proc. IECON 44th Annu. Conf. IEEE Ind. Electron. Soc.*, Washington, DC, USA, Oct. 2018, pp. 4742–4747, doi: 10.1109/IECON.2018.8591167.

[72] G. K. Gopalakrishna and S. P. Padubidre, "Development of low cost system for condition monitoring of rolling element bearing using MEMS based accelerometer," *AIP Conf Proc.*, vol. 2080, no. 1, Mar. 2019, Art. no. 050003, doi: 10.1063/1.5092931.

[73] I. Koene, R. Viitala, and P. Kuosmanen, "Internet of Things based monitoring of large rotor vibration with a microelectromechanical systems accelerometer," *IEEE Access*, vol. 7, pp. 92210–92219, 2019, doi: 10.1109/ACCESS.2019.2927793.

[74] F. Aswin, I. Dwisaputra, and R. Afriansyah, "Online vibration monitoring system for rotating machinery based on 3-axis MEMS accelerometer," *J. Phys., Conf. Ser.*, vol. 1450, no. 1, Feb. 2020, Art. no. 012109, doi: 10.1088/1742-6596/1450/1/012109.

[75] L. A. dos Santos Pedotti, R. M. Zago, M. Giesbrecht, and F. Fruett, "Low-cost MEMS accelerometer network for rotating machine vibration diagnostics," *IEEE Instrum. Meas. Mag.*, vol. 23, no. 7, pp. 25–33, Oct. 2020, doi: 10.1109/MIM.2020.9234762.

[76] S. Krishnaveni, S. S. Raja, T. Jayasankar, and P. S. Babu, "Analysis and control of the motor vibration using Arduino and machine learning model," *Mater. Today, Proc.*, vol. 45, pp. 2551–2555, May 2021, doi: 10.1016/j.matpr.2020.11.261.
[77] M. Hayouni, Z. Bousselmi, T.-H. Vuong, F. Choubani, and J. David, "Wireless IoT approach for testing in situ motor's axis vibration monitoring," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, Harbin City, China, Jun. 2021, pp. 92–97, doi: 10.1109/IWCMC51323.2021.9498761.
[78] D. Kurnia, M. N. Hakim, and I. S. Nurmala, "Implementation of FFT in industrial induction motor monitoring via mobile phone," *J. Appl. Sci. Adv. Eng.*, vol. 1, no. 1, pp. 5–10, Jan. 2023, doi: 10.59097/jasae.v1i1.8.
[79] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, Apr. 1965, doi: 10.2307/2003354.
[80] D. Sundararajan, "The discrete Fourier transform," in *Fourier Analysis—A Signal Processing Approach*, vol. 42, 1st ed. Singapore: Springer, 2018, ch. 2, sec. 2.3.1, p. 39.
[81] B. D. Malamud and D. L. Turcotte, "Self-Affine Time Series: I. Generation and Analyses," *Adv. Geophys.*, vol. 40, pp. 1–90, 1999, doi: 10.1016/S0065-2687(08)60293-9.
[82] A. P. Schneider, B. Paoletti, X. Ottavy, and C. Brandstetter, "Experimental monitoring of vibrations and the problem of amplitude quantification," *J. Phys., Conf. Ser.*, vol. 2511, no. 1, May 2023, Art. no. 012017, doi: 10.1088/1742-6596/2511/1/012017.
[83] D. R. Gaan, M. Kumar, and C. G. Majumder, "Variable rate based microstepping of stepper motor using MATLAB GUI," in *Proc. Indian Control Conf. (ICC)*, Guwahati, India, Jan. 2017, pp. 385–390, doi: 10.1109/INDIANCC.2017.7846505.
[84] R. C. Hibbeler, "Planar kinematics of a rigid body," in *Engineering Mechanics: Dynamics*, 14th ed. Hoboken, NJ, USA: Pearson, 2016, ch. 13, p. 325.
[85] H. Xing, B. Hou, Z. Lin, and M. Guo, "Modeling and compensation of random drift of MEMS gyroscopes based on least squares support vector machine optimized by chaotic particle swarm optimization," *Sensors*, vol. 17, no. 10, p. 2335, Oct. 2017, doi: 10.3390/s17102335.

JORGE FONSECA-CAMPOS was born in CDMX, Mexico, in 1973. He received the B.S. degree in engineering physics from Universidad Autónoma Metropolitana, CDMX, in 1997, and the M.S. degree in sciences with a specialty in optical physics from Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California, Mexico, in 2021. He is currently pursuing the Ph.D. degree with the Department of Biotechnology and Bioengineering, Centro de Investigación y de Estudios Avanzados del IPN (CINVESTAV). He has been a Professor with UPIITA-IPN, since 2001. His research interests include data acquisition systems, solar energy, electronic instrumentation, data analysis, water quality monitoring, and biotechnology.

ISRAEL REYES-RAMÍREZ was born in State of Mexico, Mexico, in 1975. He received the B.S., M.S., and Ph.D. degrees in physics from the School of Physics and Mathematics, National Polytechnic Institute (IPN), Mexico, in 1998, 2006, and 2010, respectively. He has been a Career Teacher with IPN, since 2000. He is also a National Researcher Level 1 CONACyT, Mexico. He has participation as a Subject Teacher with the UPIITA-IPN Master Program in Advanced Technologies and an Academic Coordinator, from 2018 to 2020. His research stay (2015–2016) with USAL, Spain. His research interests include seismic precursors, physiological and geoscience time series, studies on hybrid energy conversion systems, and various topics related to complex systems, including pollution and water quality.

JUAN L. MATA-MACHUCA received the M.Sc. and Ph.D. degrees in automatic control from CINVESTAV, National Polytechnic Institute (IPN), Mexico, in 2009 and 2013, respectively. He is currently a Researcher with the Department of Advanced Technologies, IPN. He has been a member of the National System of Researchers (level I), since 2015. He is the author or coauthor of works in international journals, conferences, proceedings, and books. His research interests include chaos synchronization, monitoring and control of nonlinear dynamical systems, nonlinear observers, fault diagnosis, and control of fractional-order systems.

LEONARDO FONSECA-RUIZ received the bachelor's degree in mechanical and electrical engineering from the National Autonomous University of Mexico (UNAM), in 2002, and the M.S. degree from the Department of Electrical Engineering with the Specialty in Bio-Electronics, Center for Research and Advanced Studies of the National Polytechnic Institute (CINVESTAV), National Polytechnic Institute (IPN), in 2006. He is currently a full-time Research Professor with UPIITA, IPN. He is the author or coauthor of 18 works published in magazines and congress. His research interests include PLC, DSP, microcontrollers, automation, design, PCB development, and CNC systems.

PAOLA N. CORTEZ-HERRERA received the M.S. degree in computer science from Instituto Politécnico Nacional, in 2008, and the M.S. degree in education from TecMilenio, in 2022. From 2013 to 2015, she was the Head of the Engineering Academic Department, Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas (UPIITA). She was the Coordinator of the Tutoring Office, from 2018 to 2021, for superior-level education. She has two patent registrations for the bachelor's thesis. She is the author of articles in congress and journals. Her research interests include social media analysis, recommendation systems, and TIC in education. She has a certification in project management. She participates in an organization regarding gender equality.

LUIS BERNARDO FLORES-COTERA received the B.S. degree in chemical engineering and the Ph.D. degree in biochemical sciences from the Faculty of Chemistry, National Autonomous University of México (UNAM), and the M.S. degree in process engineering from the Manhattan College, NY, USA. He has been a Professor with DBB-CINVESTAV for over 30 years. Previously, he was an Academic Coordinator. He is currently a Researcher and the Head of the Department of Biotechnology and Bioengineering (DBB), CINVESTAV. He is also the coauthor of more than 50 publications, including three U.S. patents and 33 published in internationally indexed journals (ISI Web of Science), which have received more than 700 citations. His research interests include the study (fundamental aspects and applied biotechnology) of microorganisms associated (endophytes) to higher plants native to Mexico and the identification and analysis of bioactive compounds synthesized by these and other microorganisms. In terms of external work, he has been a reviewer of more than 14 international scientific journals and has provided consulting advice to several companies, associations, and institutions, especially regarding the acquisition and evaluation of biochemical technology.

RICARDO AGUILAR-LÓPEZ was born in Mexico City, Mexico. He received the Sc.D. degree in chemical engineering from Universidad Autónoma Metropolitana, Mexico, and the Sc.D. degree in automatic control from Centro de Investigación y de Estudios Avanzados (CINVESTAV). Since 2007, he has been with CINVESTAV. He has published approximately 165 technical papers in different international journals and 80 contributions in chapters of specialized control books and conference proceedings. His research interest includes modeling and dynamic analysis.