Memorias del 2024 Congreso Nacional de Control Automático
8-11 de Octubre, 2024. Ciudad de México, México.
Copyright© AMCA, ISSN: 2594-2492

A Python-based open-source software for real-time control systems [★]

Bryan Rojas-Ricca [*] **Rubén Garrido** [*] **Sabine Mondié** [*]

[*] *Departamento de Control Automático, CINVESTAV-IPN, Av. Instituto Politécnico Nacional 2508, Col. San Pedro Zacatenco, Gustavo A. Madero, C.P. 07360, Ciudad de México, México. (e-mail: bryan.rojas@cinvestav.mx).*

Abstract: This paper introduces an open-source Python software that achieves a high performance for the sampled period, despite running on a general-purpose operating system instead of a real-time operating system. The software is designed to implement control systems and experiment with physical platforms, particularly the Quanser Qube-Servo 2. This article describes the coding design, numerical methods, modeling, and identification of the Qube-Servo 2, the control design, and its implementation. Finally, a quick guide to downloading via GitHub and installation requirements are provided.

Keywords: Software for system identification, Linear systems, Real-time algorithms, scheduling, and programming.

1. INTRODUCTION

The implementation of real-time control systems in industry and academia has been dominated by closed architecture software and hardware solutions. Software such as Matlab© and Simulink© are used in most applications (Sigarev et al., 2016; An et al., 2012), while among hardware solutions, Quanser products stand out, with the Quanser Qube-Servo 2 prototype being a clear example. This prototype allows to implement controller for a DC motor or a Furuta rotary pendulum. However, designing and implementing those controllers using this platform requires the use of both QUARC© and MATLAB© software (Maldonado et al., 2018; Morales et al., 2022; Hernández-Gallardo et al., 2024), a combination that, although powerful, this year increased its cost due to QUARC© changing its license to an annual pay plan, which may be out of the reach of many Mexican institutions.

Additionally, there exist real-time operating systems (RTOS) that address the real-time speed computation requirements. These operative systems are lightweight, efficient, and widely used in embedded systems (Li et al., 1997). Nevertheless, RTOS usually requires specialized hardware which increases the cost of practical implementations. To avoid this additional cost it is possible to use general-purpose operating systems (GPOS). However, GPOS are usually inefficient regarding computation speed requirements for real-time applications. In RTOS, tasks are managed and executed within specific time constraints. This differs from GPOS where tasks are scheduled based on priority. Despite the advantages of RTOS, GPOS remain more common, particularly in academics.

On the other hand, due to the digital nature of current microcontrollers and digital processors, control law implementation often requires additional discretization procedures for the controller (Fadali and Visioli, 2009). The discretized controller together with a digital signal processor are known as sampled-data systems because the inputs are acquired and processed in discrete-time instants. Several control laws can be discretized without problem through the z-transform, however, the limit of this mathematical tool is quickly reached when non-linear dynamics are considered. Another crucial characteristic of these systems is the sampling period, which quantifies the elapsed time between data samples.

In sampled-data systems, it is reasonable to expect that the sampling period is regular. Nevertheless, recent research addresses the cases where the sampling period is not regular, for example, Seuret (2012) considers the variation in the sampling period as a time-variant delay and extends some stability and robustness criteria of time-delay systems to sampled systems with irregular sampling times. In real-time applications the sampling period is crucial for numerically solving differential equations. Irregularities in the sampling period may lead to numerical problems in the computation methods (Holmes, 2007).

This article presents a Python-based open-source software designed specifically for the implementation and experimentation of real-time control systems running on GPOS. This software is designed to achieve the sampling performance of an RTOS. Moreover, the software interfaces effectively with Quanser's Qube-Servo 2 platform, positioning it as a viable alternative to the traditional QUARC© -MATLAB© setup.

The contribution is structured as follows. Section 2 describes the software design process and the main numerical methods implemented. Section 3 presents the modeling process and the parameter identification theory of the DC motor. Section 4 discusses the application of both modeling and identification to the Qube-Servo 2 platform.

[★] This work was supported by Project Conahcyt A1-S-24796.

https://doi.org/10.58571/CNCA.AMCA.2024.092

Finally, concluding remarks and an installation guide are provided in Section 5.

2. SOFTWARE DESIGN

In this section, we discuss the main challenges of real-time implementation of controllers from a computational point of view. This section also presents the structure and design of the proposed software, which is developed to provide an open-source alternative for quickly and easily implementing real-time control systems using several platforms.

2.1 Improvement of tasks performance

Considering that the software is intended to run on a personal computer using Windows or Linux as the GPOS, it is natural to assume that the hardware has two or more physical cores. This assumption is useful in software design because it allows tasks to be performed simultaneously. This is highly relevant because real-time applications require careful attention in timed tasks.

From the outset, two processes are assigned to different cores. The main process handles the graphical user interface (GUI) management, while the second process executes the real-time sampling manager, which functions similarly to an RTOS. As illustrated in the flowchart (see Fig 1), the main process may establish a connection with the platform. Specifically, it manages the `usb_2` API provided by Quanser for Python interaction with the Qube-Servo 2 platform (Quanser Inc, 2024). Regardless of whether a valid connection exists, the main process sends instructions to start or stop the experiment to the secondary process. Additionally, upon experiment completion, it displays results on a plot created using the Python module `Matplotlib` (Python Software Foundation, 2001-2024). Finally, the application handles the platform disconnection during closure.

The real-time sampling manager is a dedicated routine that governs both the data acquisition and the computation of the control law. As mentioned before, the manager operates through the `multiprocessing` Python module (Python Software Foundation, 2001-2024). Note also that the GUI and the real-time sampling manager run on different cores. It is important to note that the real-time sampling manager is performed on a separate core to achieve the lowest possible latency and it is not thread-based. The routine contains an exhaustive loop thus ensuring accurate sampling time regardless of the overall system load or performance. More specifically, the real-time sampling manager is responsible for calling the reading functions of the input port, sending this information to the functions defined by the user for computing the control law, and writing the computed control signal to the output port. It guarantees that the whole operation is executed within each sampling period so that a highly accurate sampling time is obtained.

2.2 Improvement of sampling period precision

A common practice when coding computational loops for sampled systems is to use waiting functions. For instance, in Python the most used waiting function is `sleep()` from

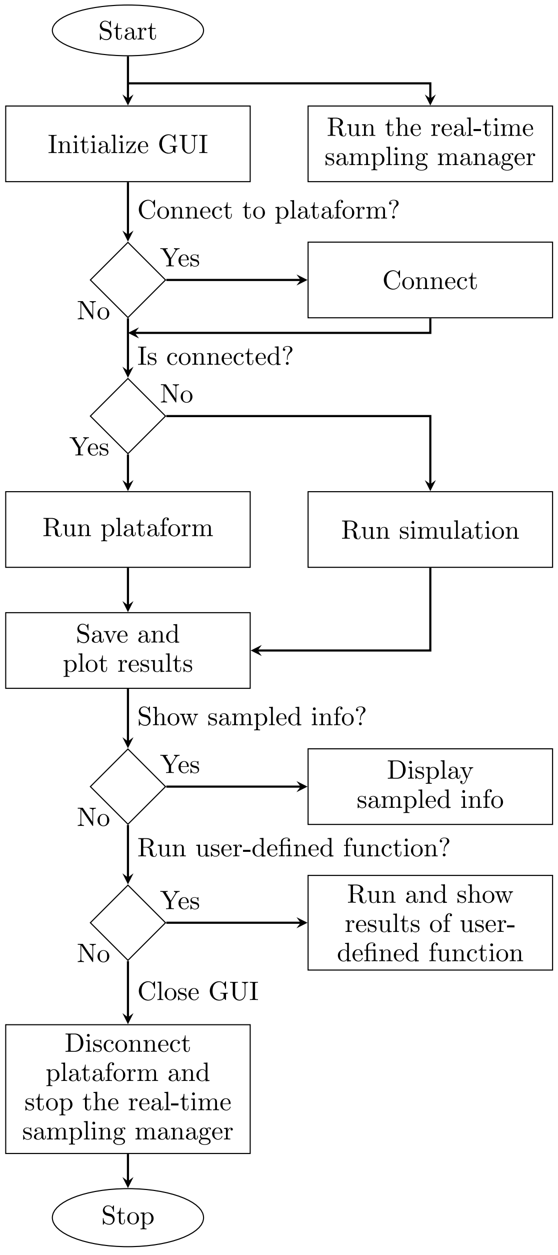


Fig. 1. Main flowchart of the software using Python.

the `time` module (Python Software Foundation, 2001-2024). The problem with this function in a GPOS is that the task manager interprets this function as a low-priority process, so it postpones its execution to attend other tasks thus resulting in a longer waiting time than the time setting in the waiting function. Therefore, avoiding waiting functions in GPOS is preferable.

An alternative to the `sleep()` function is to compute the sampling step in the loop through an active mode, which means that after data acquisition and control computation, the loop does nothing more than exhaustively verify the elapsed time. This allows the loop to preserve high priority in the task manager. However, the elapsed time should be computed with the processor's performance counter instead of the system clock which drives events at the processor level and quantifies the machine cycles of each instruction. This way of computing the elapsed time allows greater precision of the waiting time of each

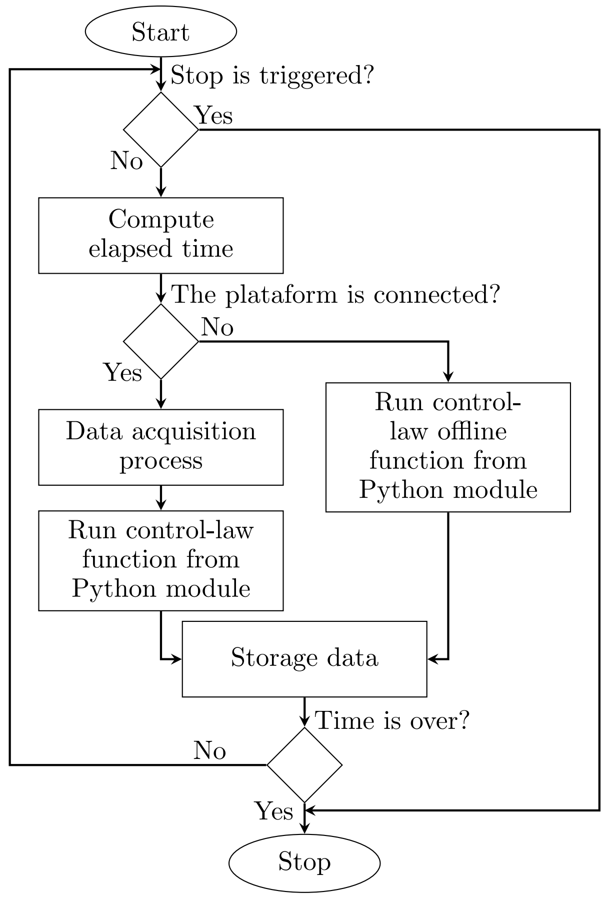

Start
Stop is triggered?
Yes
No
Compute elapsed time
The plataform is connected?
No
Yes
Data acquisition process
Run control-law offline function from Python module
Run control-law function from Python module
Storage data
Time is over?
No
Yes
Stop

Fig. 2. Flowchart of the real-time sampling manager.

sampling period. The processor's performance counter in Python is accessed through the `perf_counter()` function of the `time` module.

2.3 Graphic user interface design

The GUI, which is designed with a user-friendly interface, is focused on operating with a Qube-Servo 2 platform. The GUI features a single window as depicted in Fig. 3. This streamlined design aims at simplifying the process of conducting experiments with the platform while facilitating real-time data visualization. This enables users to gain immediate insights into their experimental results. Furthermore, the application can export the results in various widely used formats including PDF, JPG, PNG, SVG, and EPS, to name a few. The following paragraphs provide a detailed description of each GUI element numbered in Fig. 3:

(1) Connection Button: This interactive element initiates the connection process between the interface and the Qube-Servo 2 platform. Upon successful connection, the platform's specifications are displayed, and its status color changes from red to blue.
(2) Platform Information and Specifications: This section provides the technical specifications of the connected platform.
(3) Running Time and Sampled Period: These labels display the total running time of the current experiment and the period at which data is sampled, respectively.

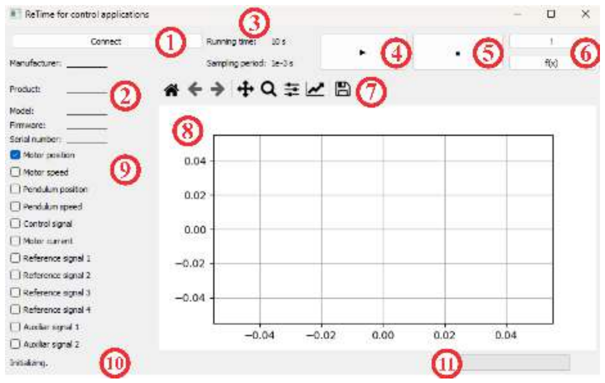


Fig. 3. Graphic user interface.

(4) Run Button: This control initiates the execution of experiments on the connected platform, if available. Otherwise, a simulation can be executed.
(5) Stop Button: This control interrupts the currently running experiment and stops the platform, allowing users to end their experiment at any time.
(6) Additional Function Buttons: These controls provide access to the sampling time information and execute an additional user-defined function.
(7) Data Visualization Toolbar: This control panel, powered by Matplotlib, provides various tools for manipulating the data visualization, such as zooming, panning, and saving the current view.
(8) Data Visualization Panel: This is the main area where the data from the real-time experiments is plotted after the execution.
(9) Signal Selector: These checkboxes allow users to select which signals are displayed on the Data Visualization Panel.
(10) Status Bar: This section displays the software and connected platform's status, warnings, and errors.
(11) Progress Bar: This indicator visually represents the progress of the current experiment.

2.4 Numerical methods for controller implementation

In Python, there are powerful modules that facilitate the analysis and simulation of control systems, such as `control`, `sciPy`, and `simuPy`. However, these modules are not designed to address real-time implementations. Therefore, we decided to include a Python script `reTimeSytems.py`, which codes numerical derivatives and integrals and can apply transfer functions to signals obtained in real-time. Holmes (2007) provide detailed descriptions of numerical methods summarized in this section.

In the `reTimeSytems.py` file, we approximate derivatives using the first order of backward finite differences. The first-order algorithm is chosen by its simplicity and the backward type is selected because no time-ahead data is available in real-time. This method is based on the backward expansion of a function $f(x)$ in Taylor series:

$$f(x_{i-1}) = f(x_i) - Tf'(x_i) + \frac{T^2}{2!}f''(x_i) - \cdots \tag{1}$$

where $T = x_i - x_{i-1}$. Truncating the series expansion at the first derivative, we obtain:

$$f'(x_i) \approx \frac{f(x_{i-1}) - f(x_i)}{T} \tag{2}$$

which provides a first-order approximation for the first derivative of $f(x)$ at $x = x_i$.

To compute the integral of a function $f(x)$, we apply the composite trapezoid rule, which is used to compute the integral of f in a subdivided interval $[a, b]$ partitioned as:

$$a = x_0 < x_1 < x_2 < \cdots < x_p = b \tag{3}$$

Hence:

$$\int_a^b f(x)dx \approx \frac{1}{2}\sum_{i=1}^{p} T\left[f(x_{i-1}) + f(x_i)\right] \tag{4}$$

with h as defined above.

Additionally, the module `reTimeSytems.py` implements the computation of a general transfer function of the form

$$\frac{Y(s)}{U(s)} = \frac{b_m s^m + \ldots + b_1 s + b_0}{a_n s^n + \ldots + a_1 s + a_0} \tag{5}$$

with $m \leq n$. The module computes the output $y(t)$ using the backward first-order approximation for all the derivatives. Naturally, this requires m initial conditions for u and n initial conditions for y. The implemented algorithm sets all initial conditions to zero unless otherwise stated. Hence, the first-order approximation of the k-th derivative of $f(x)$ is given by

$$f^{(k)}(x_i) = \sum_{j=0}^{k} \binom{k}{j} (-1)^j f(x_{i-j}) \tag{6}$$

where $\binom{k}{j}$ is the binomial coefficient between k and j. Therefore, (5) is solved in the time domain for $y(t)$ and expressed in terms of $y(t - jT)$, with $j \in \{1, 2, \ldots, k\}$ and T the sampling period.

3. MODELING PLATFORM AND ITS PARAMETER IDENTIFICATION

This section presents a review of the modeling and identification process, with a practical focus on the Qube-servo 2 platform by Quanser. While this platform can be used as both DC motor and Furuta rotary pendulum systems, the subsequent discussion concentrates solely on the DC motor system.

3.1 DC motor modeling

The DC motor, a prevalent example in control textbooks and applications, comprises an electrical circuit that describes the behavior of the motor armature, and a rotational mechanical system that characterizes the rotor's behavior. As depicted in Fig. 4, the input voltage, denoted by $V_s(t)$, is applied across the armature. The armature, characterized by resistance R_a and inductance L_a, has a current $i_a(t)$ flowing through it. The position and speed of the motor are represented by $\theta(t)$ and $\dot{\theta}(t)$ respectively. The counter-electromotive force is generated due to the motion of the armature in the magnetic field and it bridges the electrical and mechanical behaviors. The counter-electromotive force is given by

$$V_b(t) = k_b\dot{\theta} \tag{7}$$

where k_b is known as the electromotive force constant. This force opposes the input voltage $V_s(t)$. The motor

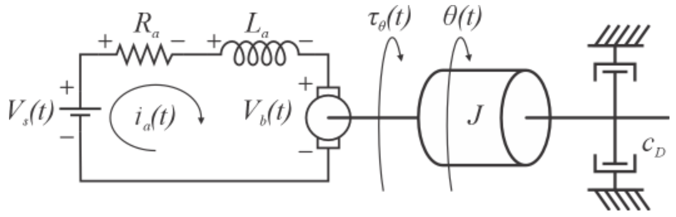


Fig. 4. DC motor diagram.

torque is directly proportional to the armature current and is responsible for the rotation of the armature:

$$\tau_\theta(t) = k_\tau i_a(t) \tag{8}$$

where k_τ is the motor torque constant. Finally, the damping rate, c_D, represents the friction force that opposes the motion of the motor.

The differential equations describing the electrical and mechanical behaviors of the DC motor are:

$$R_a i_a(t) + L_a \dot{i}_a(t) + V_b(t) = V_s(t) \tag{9a}$$

$$J\ddot{\theta}(t) + c_D\dot{\theta}(t) = \tau_\theta(t) \tag{9b}$$

Considering (7) and (8), and defining the state $x = [x_1\ x_2\ x_3]^\top$, with $x_1 = i_a$, $x_2 = \theta$ and $x_3 = \dot{\theta}$ yields:

$$\dot{x}(t) = \begin{bmatrix} -\frac{R_a}{L_a} & 0 & -\frac{k_b}{L_a} \\ 0 & 0 & 1 \\ -\frac{k_\tau}{J} & 0 & -\frac{c_D}{J} \end{bmatrix} x(t) + \begin{bmatrix} \frac{1}{L_a} \\ 0 \\ 0 \end{bmatrix} u(t) \tag{10}$$

where $u(t) = V_s(t)$ is the input. The output of (10) is

$$y(t) = [0\ 1\ 0]\, x(t) \tag{11}$$

which describes the most usual physical implementation in DC motor platforms.

The open loop transfer function of (10-11) is given by

$$G(s) = \frac{k_\tau}{s((L_a s + R_a)(Js + c_D) + k_\tau k_\omega)} \tag{12}$$

Assuming that $L_a << R_a$, we get $L_a s + R_a \approx R_a$. A common assumption in DC motors modeling. The transfer function of the simplified model is

$$G(s) = \frac{b}{s(s + a)} \tag{13}$$

where

$$a = \frac{R_a c_D + k_\tau k_\omega}{R_a J} \quad \text{and} \quad b = \frac{k_\tau}{R_a J}$$

There are two ways to obtain the values of a and b: Using the parameter values provided in the Qube-servo 2 platform manual, or using a parameter identification method. The next sections compare both approaches.

3.2 Parameters identification

The parameter identification problem involves finding parameters a and b such that the behavior (13) approximates the behavior of the real DC motor system. The Least Squares method (Isermann, 2005) is a classical and powerful way to identify the parameters of a dynamic system. This method minimizes the quadratic error criterion of a linear problem of the form:

$$A\Theta = B \tag{14}$$

when A and B are given. The quadratic error criterion is:

$$J = \sum e^\top e = \sum (B - A\Theta)^\top (B - A\Theta)$$

and its minimum is achieved when the partial derivative of J w.r.t Θ vanishes, that is:

$$\frac{\partial J}{\partial \Theta} = -2B^\top A + 2\Theta^\top A^\top A = 0 \tag{15}$$

Therefore the value Θ that minimize the criterion J are given by

$$\Theta = (A^\top A)^{-1} A^\top B \tag{16}$$

The time-domain representation of (13) is

$$\ddot{y}(t) = -a\dot{y}(t) + bu(t) \ , \tag{17}$$

It can be written in the form (14). In practice, only y and u are available from measurements, and one must obtain approximations $y_1 \approx \dot{y}$ and $y_2 \approx \ddot{y}$ using filters:

$$\frac{Y_1(s)}{Y(s)} = \frac{f_2 s}{s^2 + f_1 s + f_2} \tag{18}$$

$$\frac{Y_2(s)}{Y(s)} = \frac{f_2 s^2}{s^2 + f_1 s + f_2} \tag{19}$$

where f_1 and f_2 are positive constants, while Y_1 and Y_2 denote the Laplace transforms of y_1 and y_2, respectively. Therefore, (17) is written as:

$$y_2(t) = -ay_1(t) + bu(t) \tag{20}$$

The identification problem consist of sampling u and y with a fixed period $T \in \mathbb{R}_+$ and filtering y to build the following system of equations:

$$\begin{bmatrix} -y_1(T) & u(T) \\ -y_1(2T) & u(2T) \\ \vdots & \vdots \\ -y_1(kT) & u(kT) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_2(T) \\ y_2(2T) \\ \vdots \\ y_2(kT) \end{bmatrix} \tag{21}$$

with $k \in \mathbb{N}$. Thus, the parameter $\Theta = [a, b]^\top$ are obtained using (16). The identification process requires that the reference fulfills the *persistency of excitation* (PE) condition (Sastry and Bodson, 1989). The filtered white noise signal fulfills such a condition because the system is excited with the same amplitude at all frequencies. In practice, there is no ideal white noise, nevertheless, an approximation can be computed as follows. Consider a set of q distinct frequencies defined as

$$0 < \omega_0 < \omega_1 < \cdots < \omega_q \tag{22}$$

thus, the excitation signal is computed by

$$N(t) = \sum_{i=0}^{q} \sin(2\pi\omega_i t + \phi_i) \tag{23}$$

where ω_i and ϕ_i are the frequency and phase, respectively. In this way, a flat power spectrum is approximated at the frequency interval $[\omega_0, \ \omega_q]$.

4. EXPERIMENTAL RESULTS

We use the software introduced in this paper to implement the parameter identification method detailed in Section 3.2. We compute (23) with $\omega_0 = 0$, $\omega_q = 200$, $q = 1024$ and set random numbers for the phases ϕ_i. The excitation signal obtained is the red dashed line and the output signal is the blue solid line of Fig. 5. Our identification process yields the parameter estimates of $a = 8.05$ and $b = 171.7$. These values closely align with those reported in the literature; for instance Reck (2018) reports parameter estimates of $a = 8$ and $b = 195$.

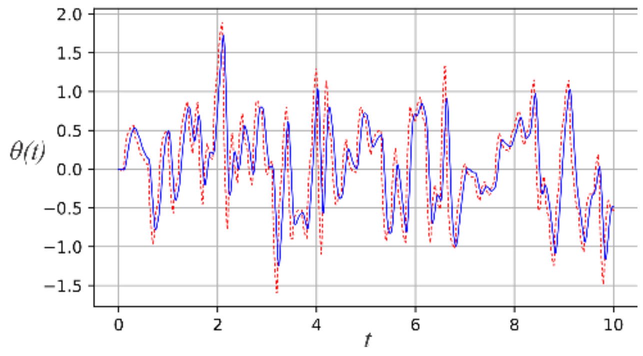
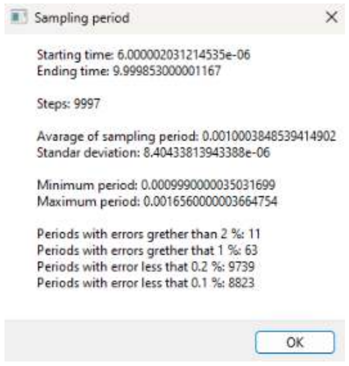


Fig. 5. The parameter identification process of the Qube-Servo 2 platform, where the reference (red dashed line) provided by (23), and the resulting output (blue solid line) are displayed.



```
Sampling period

Starting time: 6.000002031214535e-06
Ending time: 9.999853000001167

Steps: 9997

Avarage of sampling period: 0.0010003848539414902
Standar deviation: 8.40433813943388e-06

Minimum period: 0.0009990000035031699
Maximum period: 0.0016560000003664754

Periods with errors grether than 2 %: 11
Periods with errors grether that 1 %: 63
Periods with error less that 0.2 %: 9739
Periods with error less that 0.1 %: 8823
```

Fig. 6. Sampling period information.

Additionally, as illustrated in Fig. 6, the software stores and displays the sampling period statistics via an additional function button. Notably, the software maintains a high precision in a sampled period of 1 ms, with no errors greater than $\pm 2\mu s$, for 97.39% of the running time.

Consider the Proportional Derivative (PD) controller:

$$C(s) = \frac{1}{b}(k_d s + k_p) \tag{24}$$

The closed-loop system (13, 24) is

$$G_{c.l.}(s) = \frac{k_d s + k_p}{s^2 + (a + k_d)s + k_p} \tag{25}$$

The Ruth-Hurwitz criterion (Nise, 2019) gives $k_d > -a$ and $k_p > 0$ as stability conditions, but note that every $k_d < 0$ introduces a non-minimum phase zero. Therefore, we state $k_d > 0$ and $k_p > 0$ as tuning rules. The estimated parameters a and b are used to calculate the desired closed-loop characteristic function coefficients. To assign a double real root at $s = -20$ we set $k_d = 31.95$ and $k_p = 400$, which satisfies the stated tuning rules. Fig. 7 depicts a square wave as reference θ_d (black dashed line), and the angular position θ (blue solid line) whereas Fig. 8 displays the control signal u (red solid line).

5. CONCLUSION AND INSTALLATION GUIDE

The proposed software serves as a starting point for self-development or as an alternative for implementing real-time control systems using Python. The software achieves

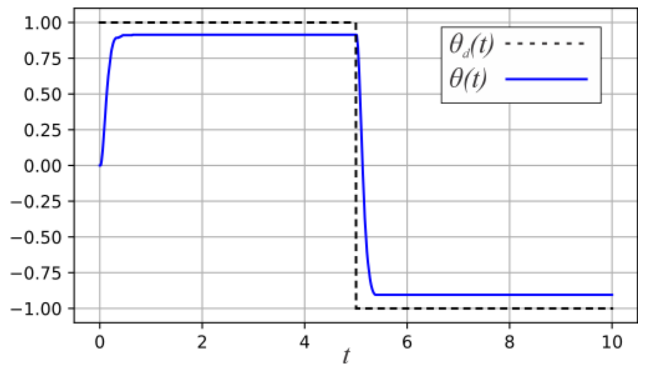


Fig. 7. Motor position output (blue solid line) of Qube-Servo 2 platform with a PD controller and the corresponding reference signal (black dashed line).

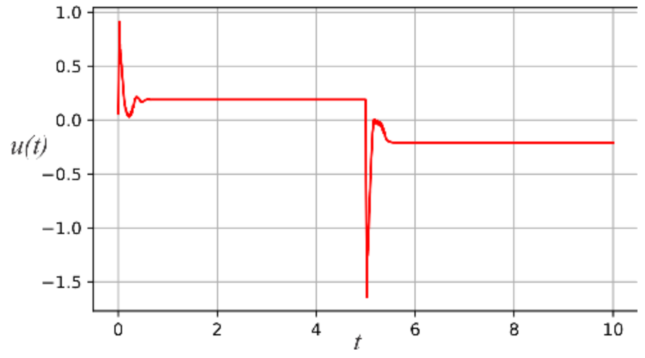


Fig. 8. The voltages supplied to the DC motor (input signal).

a high sampling time performance despite the limitations of GPOS, and it provides a Python resource of numerical methods for real-time applications. The parameter identification and the control law are available in the code, therefore, these results can be easily reproduced. It is important to underline that the proposed low-cost software can be modified to make it compatible with platforms such as Arduino and Raspberry Pi. This feature expands the scope of the Python code, making it a versatile tool for implementing experimental platforms for real-time control systems.

The modules and Python code are available at

`https://github.com/BryanRojasRicca/ReTime_CA`

where it is possible to download, clone, or collaborate with the repository. The software execution requires the following Python modules installed: `multiprocessing`, `time`, `numpy`, `matplotlib`, `PyQt5` and `cffi`. To interact with the Qube-Servo 2 platform, the *Quanser SDK*, an application programming interface (API) for C and Python hardware interfacing and communication libraries are required. The *Quanser SDK* download from:

`https://github.com/quanser/quanser_sdk_win64`
`https://github.com/quanser/quanser_sdk_linux`

for GPOS Windows and Linux, respectively. Also, installing QUARC© (including trial version) the *Quanser SDK* is downloaded. Finally, installing the Python API from *Quanser SDK* as in (Quanser Inc, 2024), choosing `"%QSDK_DIR\%` or `"\%QUARC_DIR\%` appropriately.

REFERENCES

An, B., Liu, G., and Senchun, C. (2012). Design and implementation of real-time control system using rtai and matlab/rtw. In *Proceedings of 2012 UKACC International Conference on Control*, 834–839. doi: 10.1109/CONTROL.2012.6334740.

Fadali, M.S. and Visioli, A. (2009). *Digital control engineering, analysis and design.* Academic Press, Boston. doi:10.1016/B978-0-12-374498-2.X0001-X.

Hernández-Gallardo, J.A., Guel-Cortez, A.J., González-Galván, E.J., Cárdenas-Galindo, J.A., Félix, L., and Méndez-Barrios, C.F. (2024). Synergistic design of optimal pi controllers for linear time-delayed systems. In M.N. Cardona, J. Baca, C. Garcia, I.G. Carrera, and C. Martinez (eds.), *Advances in Automation and Robotics Research*, 77–88. Springer Nature Switzerland, Cham.

Holmes, M.H. (2007). *Introduction to Numerical Methods in Differential Equations.* Springer New York, New York. doi:10.1007/978-0-387-68121-4.

Isermann, R. (2005). *Identification of Dynamic Systems*, 293–332. Springer London, London. doi:10.1007/1-84628-259-4_7.

Li, Y., Potkonjak, M., and Wolf, W. (1997). Real-time operating systems for embedded computing. In *Proceedings International Conference on Computer Design VLSI in Computers and Processors*, 388–392. doi: 10.1109/ICCD.1997.628899.

Maldonado, J., Lopez, K., Garrido, R., and Mondié, S. (2018). Implementing time-delay controllers on an educational motion control platform. In *2018 XX Congreso Mexicano de Robótica (COMRob)*, 1–6. doi: 10.1109/COMROB.2018.8689425.

Morales, O.J., Maldonado, J., and Garrido, R. (2022). Robust adaptive control of servo systems. In *2022 19th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, 1–6. doi:10.1109/CCE56709.2022.9975932.

Nise, N.S. (2019). *Control Systems Engineering, 8th Edition.* Jhon Wiley & Sons, New Jersey.

Python Software Foundation (2001-2024). Python documentation. `https://docs.python.org/3/`.

Quanser Inc (2024). Quanser Python API Documentation. `https://docs.quanser.com/quarc/documentation/python/installation.html`.

Reck, R. (2018). Validating dc motor models on the quanser qube servo. In *Dynamic Systems and Control Conference, DSCC 2018, Sep 30 –Oct 3, Atlanta, Georgia, USA.* doi:10.1115/DSCC2018-9158.

Sastry, S. and Bodson, M. (1989). *Adaptive control: Stability, convergence, and robustness.* Prentice-Hall, Inc., New Jersey. doi:10.1016/B978-0-12-374498-2.X0001-X.

Seuret, A. (2012). A novel stability analysis of linear systems under asynchronous samplings. *Automatica*, 48(1), 177–182. doi:10.1016/j.automatica.2011.09.033.

Sigarev, V., Kuzmina, T., and Krasilnikov, A. (2016). Real-time control system for a dc motor. In *2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW)*, 689–690. doi:10.1109/EIConRusNW.2016.7448276.