An Experimental Analysis of the Latency of Linux Kernels Applicable for Real-Time Control Strategies

Ayoub Khodaparast

e-mail: khodaparast@mut-es.ac.ir

Faculty Naval of Aviation, Malek Ashtar University of Isfahan, Isfahan, Iran

Hassan Ghiti Sarand

e-mail: hgiti@aeoi.org.ir

Robotics and AI Research Institute of Atomic Energy Organization of Iran, Tehran, Iran

Abstract— Real-time control applications, crucial in robotics, industrial automation, and medical devices, demand precise and predictable timing for reliable operation. This paper presents an experimental investigation into the latency performance of various Linux kernels, including standard Linux, a low-latency kernel, Xenomai, and a real-time kernel patched with PREEMPT_RT. Our test setup utilizes a data acquisition card to measure the latency between sending and receiving a pulse signal through analog input-output channels, generated by a C++ code. This latency metric serves as an indicator of the responsiveness of the kernel and other control objects on a specific computer system. Our experiments were conducted under a wide range of conditions to comprehensively assess latency performance. This includes different versions of standard and real-time Linux kernels, varying numbers of CPU cores, program priority levels, data saving rates, a range of data acquisition cards, communication protocols, thread assignments to processor cores, and test durations. The results highlight the importance of long-term testing to accurately determine the maximum latency. Furthermore, the findings demonstrate significantly lower latency for the PREEMPT_RT patched kernel across various tests, indicating its suitability for demanding real-time control applications that require tight timing constraints.

Keywords: Latency measurement, Real-time control, Real-time Linux, Linux kernel

1. INTRODUCTION

In real-time systems, meeting the assigned sampling-time is critical, particularly in hard or firm real-time applications. In such systems, the controller must receive feedback signals and produce appropriate responses within the assigned sampling-time to ensure the stability and reliability of the controlled system. Therefore, it is necessary to use a low latency operating system for control purposes. Linux as an open source operating system has earned popularity in recent years, particularly in the field of real-time systems [1-3]. Linux real-time (RT) extensions have been widely employed for control applications [4-15] such as robotic systems.

Measuring latency in an operating system intended for real-time applications is challenging due to unpredictable time-behavior factors, such as hardware interrupts and scheduling delays, which can occur on the system. In the literature, some works focused on latency analysis of standard (vanilla) Linux kernel and its RT versions [2, 3 ,16-21]. In [16-22], some analysis tools such as LXRT, cyclictest, ftrace, … are employed to measure the latencies happen inside Linux kernel. In [23, 24], the automata-based model method was proposed to model and track events of a RT Linux patched with Preempt-RT. In [25], capabilities of the standard Linux and PREEMPT-RT patch for a real-time application were compared. The steps of patching a Linux kernel with PREEMPT_RT and effect of some parameters of the kernel configuration were discussed in [26]. Chen et al. [27] proposed an analysis method for examining the execution path that a task, such as a file read or write operation, may traverse under various test conditions. The scheduling latency of a multicore processor for Xenomai and Preempt-RT, two RT Linux versions, was discussed in [28].

In summary, the majority of the aforementioned studies focused on measuring the latency of functions running in the Linux kernel, such as scheduling, system management interrupts, and memory access. Moreover, the measurements in these studies are based on short-term tests which may result in missing the latency of critical events or tasks that can significantly impact overall latency. For control applications, in addition to the latency of processes running in the Linux kernel, delays arising from the control algorithm code, the data acquisition (DAQ) cards and communication protocol that may be significant, must be taken into account. On the other hand, several studies [17, 19, 20], [29] have assessed the performance of the Linux kernel by subjecting it to workloads that stress the CPU, memory, I/O and other components. However, in a real-time control strategy, many of these stressors that can increase latency are avoided.

2. HIGHLIGHTS

This paper proposes a method for measuring latency on a personal computer (PC) to determine an appropriate sampling frequency for real-time control applications. Our approach utilizes both standard Linux and real-time kernels, including the Preempt-RT patched kernel. The PC is equipped with a DAQ card and runs a C++ code to generate a periodic signal for latency measurement. Therefore, latency encompasses the latency of the Linux kernel and delays associated with the DAQ card, the card's communication protocol, and the C++ code. The main contributions of this paper are as follows:

1. This work presents a novel latency measurement methodology that surpasses existing approaches [2, 3, 16-22] by encompassing the entire control loop latency. This includes not only

kernel process execution times, but also delays introduced by application code, data acquisition (DAQ) card processing, and communication protocol overhead. This approach provides a more accurate representation of real-world latency in control systems.

2. Unlike the studies [17, 19, 20], [27] that have artificially stressed the CPU, memory, I/O, and other components, this research prioritizes measuring latency in practical control scenarios. This methodology focuses on factors directly impacting sampling time in real-time control strategies, providing a more relevant and actionable analysis.

3. The research investigates the impact of key system parameters and tasks on overall latency. This includes analyzing the influence of data saving, CPU core count, control algorithm complexity, and other relevant factors. This granular analysis provides valuable insights for optimizing real-time control system performance.

4. This study benchmarks the latency performance of PREEMPT_RT Linux against standard Linux, Low-latency kernel, Xenomai, and lower PREEMPT_RT versions. This comparative analysis highlights the relative advantages of PREEMPT_RT for real-time control applications.

5. Unlike prior studies [2, 3, 16-22] that relied on short-duration tests, this work incorporates long-term latency measurements. This approach provides a more accurate representation of the final sampling time in real-world control scenarios.

6. The study investigates the influence of different DAQ cards on latency, assessing the impact of hardware design and communication protocol characteristics. This analysis highlights the crucial role of hardware selection in achieving desired latency performance.

Our proposed latency evaluation methodology provides a comprehensive and applicable approach for real-time control applications, enabling system designers and developers to optimize performance and ensure reliable behavior. The rest of this paper is organized as follows.

Section 3 describes the test setup configuration and methodology used for latency measurement. In Section 4, the results of experiments under various conditions are presented. Finally, Section 5 concludes the paper.

Remark 1: This research investigates the latency associated with the Linux kernel, the DAQ card, its communication protocol, and the C++ code. This study focuses on these components as key factors influencing the maximum achievable latency for control applications. While the Linux kernel's intricate interplay of scheduling, system management interrupts, and memory access also contributes to overall latency, a comprehensive analysis of all contributing processes is beyond the scope of this study. This research provides a foundation for further investigation into the individual contributions of these processes, which is crucial for optimizing the performance of control applications.

Remark 2: At the time of the initial tests, the Linux driver for DAQ cards was only compatible with kernel 4.15.0, necessitating the use of Ubuntu 18.04. A driver for kernel 5.4.0 (Ubuntu 20.04) became available after our initial data collection. This allowed us to repeat specific tests and assess the impact of the newer driver and kernel version on latency.

3. MATERIALS AND METHODS

This section describes the methodology for measuring system latency. The main setup involves an Intel Core i3 dual-core computer with 4GB RAM running Ubuntu 18.04, equipped with Advantech PCI1716 and PCIe1816 DAQ cards. PCI1716 is the primary DAQ card in the dual-core system and in the quad-core system, PCIe1816 card is utilized.

A C++ code generates a pulse train signal and measures latencies. Data logging includes minimum, maximum, mean, and standard deviation of delays, with maximum latency being critical for hard real-time systems and mean delay for soft/firm real-time systems. A high-resolution timer is used, measuring latencies with a resolution equal to that of the system clock (303 ns for this specific computer).

The reported mean and maximum latencies were calculated from extended-duration runs on the original platform; sample standard deviations or confidence intervals are not provided here (see Remark 5). Such measures are commonly used in soft real-time analyses where occasional sampling deviations are acceptable.

Remark 3: For short, the Linux kernel patched with PREEMPT_RT is called PREEMPT_RT kernel in this paper.

Remark 4: In this paper, us denotes micro second and Td_max and Td_mean denote maximum latency and mean latency, respectively.

Remark 5: Mean and maximum latencies were rigorously determined through long-term testing on the original experimental platform. Detailed statistical error margins (e.g., sample standard deviations or confidence intervals) are not presented because the project concluded and the testbed is no longer available for re-running repeated trials. The experimental design prioritized extended-duration observation to capture representative central tendency and worst-case behavior; for real-time control applications the maximum (worst-case) latency is operationally more critical than short-term variability. We acknowledge that reporting formal error margins would increase statistical completeness and commit to including them in future work when the experiments can be repeated.

4. TEST RESULTS

In this section, the experimental results of the latency measurement under different condition are presented. The common conditions of the tests are as follows:

- The priority of the C++ code is set to 99 (highest real-time priority) unless otherwise specified in test conditions.
- If minimum and/or maximum value of latency is changed, the results will be saved and displayed in Terminal window unless the result of first run of every one million runs will be selected to be saved and displayed. Otherwise, is specified in test conditions.
- Ubuntu Linux kernel version 5.3.0-28 and PREEMPT_RT kernel with patch 5.2.21-rt15 are employed for experiments. Other variations of kernels are specified in test conditions.

To configure the RT-kernel, we initially investigated the impact of various kernel modules on latency by enabling and disabling them. The test results showed no significant changes in maximum and mean latency values. Therefore, this paper focuses on building the RT-kernel with the following configuration:

Enabled options:

- Fully preemptible kernel
- High-resolution timers

Disabled options:

- Check for stack overflows
- Frequency scaling mechanism
- Hyper-threading

This configuration ensures a more predictable and responsive kernel for real-time applications.

Test termination criteria were carefully considered given the inherent characteristics of our system. While fixed-duration tests are often preferred for consistency, the unpredictable nature of kernel behavior and the variability in optimal test durations across different system configurations made their direct application challenging for initial exploratory tests. For these early investigations (Subsections 4.1-4.4), where the impact of various parameters such as data saving, program priority, and core assignment on latency variations was studied, a latency threshold-based termination criterion was employed. This approach efficiently captured the latency dynamics under varying conditions. For the long-term tests (Subsection 4.5), once parameters were finalized, a trend-based termination criterion was adopted. Tests were concluded when the amplitude of changes in maximum latency became negligible, effectively resulting in durations akin to fixed-duration tests once system stability was achieved.

4.1. DATA SAVING AND KERNELS

This subsection discusses the effect of following parameters which may alter the latency:

1. Data Saving
2. Priority of the C++ code execution
3. Version of PREEMPT_RT kernel
4. Other RT kernels

The test results for the PREEMPT_RT kernel 5.2.21-rt15 (lines 1-4) are shown in Table 1, alongside results for other kernels. For all tests, the criterion for termination was defined as the first instance an observed latency (Td) value exceeded 75 µs. Column three of Table 1 presents the exact first observed latency value (Td_max) that triggered this termination. It is important to note that while the threshold for stopping is 75 µs, the specific latency value that first crosses this threshold, as seen in Td_max, can be considerably higher depending on the test condition (column 2). For example, Test 3 shows a Td_max of 95.22 µs, indicating that 95.22 µs was the first recorded latency value to exceed 75 µs, at which point the test was terminated. As stated in Remark 1, a detailed investigation into the underlying causes for these Td_max values is beyond the scope of this paper and will be addressed in future work; this study focuses on characterizing the observed delay distributions.

Table 1. The latency measurement under different conditions

	Test Condition	**Td_max**	**Td_mean**	**Test Duration (dd:hh:mm)***
1	Data Saving Disabled	73.88 us	52.50 us	03:00:00
2	Data Saving Enabled	89.60 us	52.83 us	03:05:00
3	Priority decreasing (97)	95.22 us	52.68 us	04:18:50
4	Data saving and displaying rate (for every run)	90.48 us	52.79 us	00:17:00
5	Xenomai kernel version 4.19.124	137.75 us	54.06 us	01:19:15
6	Standard Linux (Ubuntu 18.04)	264.43 us	52.80 us	01:19:30
7	PREEMPT_RT patch low version (4.14.170)	89.72 us	52.88 us	02:01:35
8	Low Latency kernel	97.59 us	53.08 us	01:21:30

* These abbreviations represent the test duration in days (dd), hours (hh) and minutes (mm). Moreover,

According to Table 1, we can conclude that:

- Data saving task increases the maximum latency, Td_max, as high as 16 us.
- Data saving and its display rate do not affect Td_max.
- The lower priority assigned to the code does not improve latency. However, according to some references [1, 2], setting the priority of a real-time application below 99 is recommended to allow kernel-critical tasks to run.
- Comparison results of different kernels show that PREEMPT_RT kernel has the least latency while that of latency of the standard Linux is the highest one.
- The mean latency, Td_mean, is not affected by varying test conditions and certain parameters.

4.2. COMMUNICATION BUS AND HARDWARE

This subsection discusses the effects of DAQ card parameters, such as communication protocol and hardware design characteristics, on latency variation. The study specifically aims to compare the bus effects (PCI vs. PCIe) and hardware design influences on latency. For this purpose, latency measurements are conducted using Advantech PCI1716, PCIe1816 cards and National Instrument (NI) PCI6281 DAQ card, which we refer to as NI6281 in this paper. The PCI1716 and NI6281 (both PCI cards from different vendors) are used to analyze hardware design effects, while the PCI1716 and PCIe1816 comparison focuses on bus protocol impact. The results are illustrated in Fig. 1 and Fig. 2. Fig. 1 shows Td_ max and Td_mean of the three DAQ cards when data saving is enabled. In Fig. 2, Td_max of the three cards is illustrated for both modes of data saving.

The following results are notable:

- For two data saving modes, The Td-max of NI6281 is at least 25 us higher than that of the PCI1716. However, among the three tested cards, the NI6281 has the lowest Td_mean (48.81 *us*).
- PCIe1816 has the lowest Td_max compared to the PCI cards.
- For Advantech cards, i.e. PCI1716 and PCIe1816, enabling the data saving mode leads to a higher Td_max compared to the disabled mode of data saving. But for the NI NI6281 card the results are reversed.

Fig. 1 and Fig. 2 demonstrate that the communication protocol speed and design characteristics of a DAQ card can significantly affect its latency value. For instance, both PCI cards PCI1716 and NI6281 exhibit different latencies, primarily due to variations in the speed of their analog input (AI) and analog output (AO) channels, as well as differences in their respective board designs. The same conclusion holds for the PCIe1816 card.

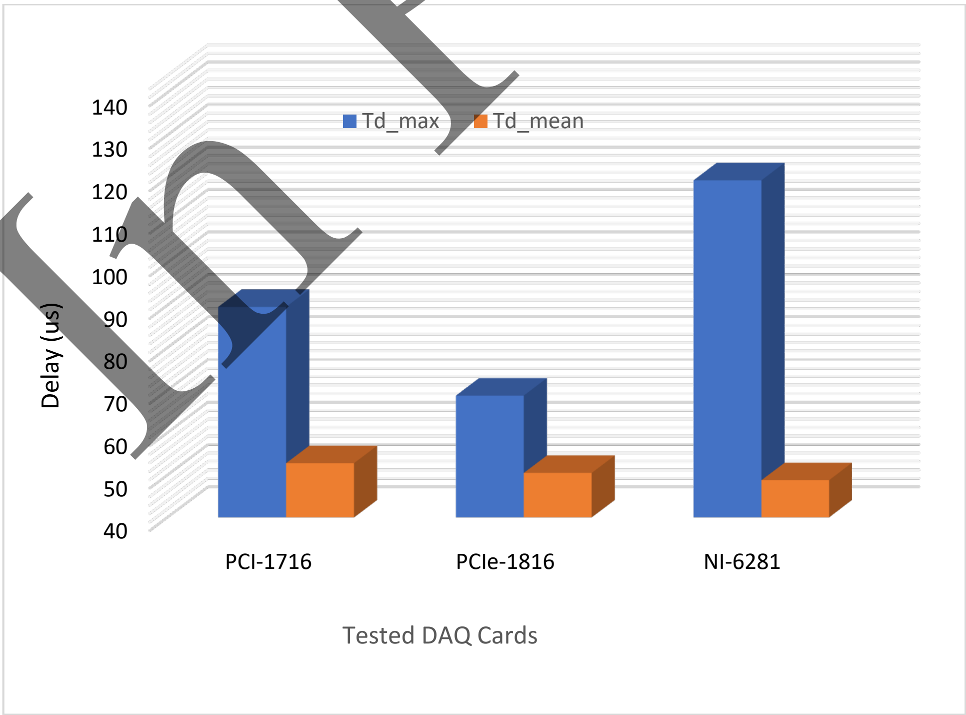


Fig. 1. Latency comparisons of the tested three DAQ cards when data saving task is enabled.

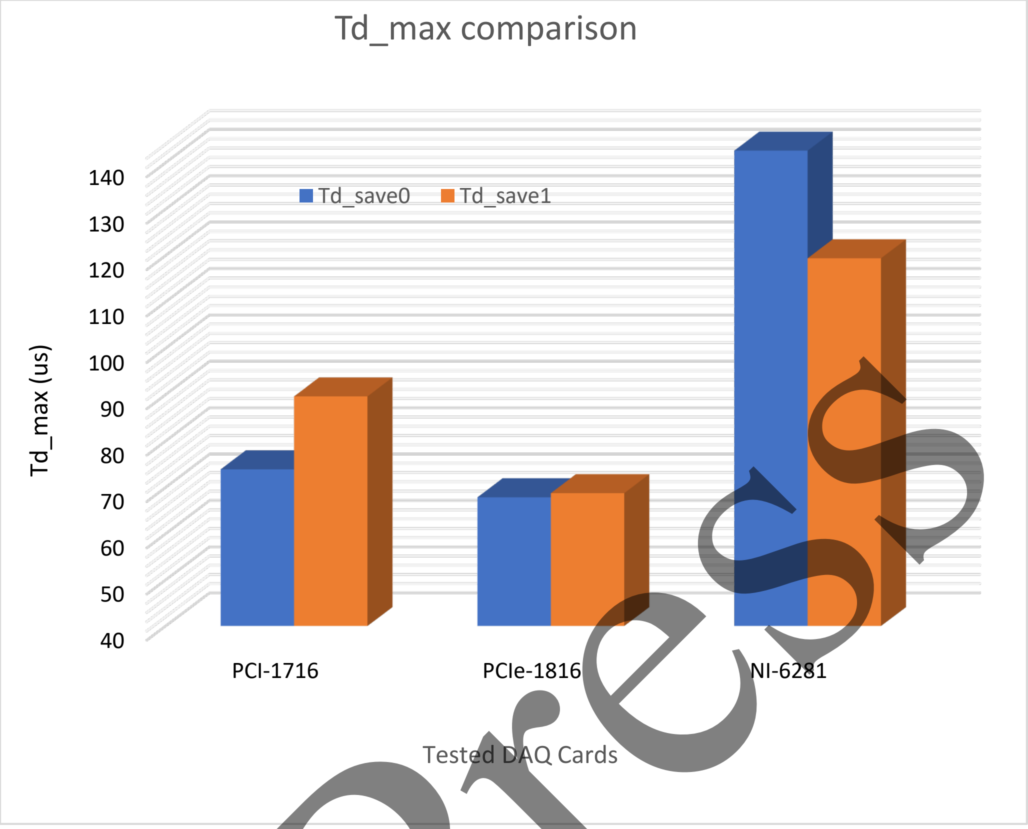


Fig. 2. Latency comparisons of the tested three DAQ cards under two data saving modes.

4.3. CONTROL ALGORITHM

In this section, we introduce an adaptive control algorithm to the C++ code to investigate its impact on latency. The complexity of a control law increases when it includes the following characteristics [29-31]:

1. Call-by-reference: This involves indirect memory access (through a reference), which can be slightly slower than directly accessing the data itself.
2. Recursive relations: Recursive function calls introduce overhead, increase memory consumption, and make it difficult to predict execution time.

3. Nonlinear functions: These require more complex calculations and processing time compared to linear functions.

To address these complications, a neuro-adaptive control law inspired by [30] is proposed for unknown SISO nonlinear affine systems described by

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= f(x) + g(x)u + d \end{aligned} \tag{1}$$

where $x = (x_1, x_2)^{\mathrm{T}}$, u and d are the states, input and disturbance of the system, respectively. The system dynamics $f(x)$ and input gain $g(x)$ are unknown. In addition, the input gain $g(x)$ is assumed to be nonzero for all time. Since the system (1) is in canonical form, it can be represented as

$$\ddot{x}_1 = f(x) + g(x)u + d \tag{2}$$

Let x_r be the desired signal, then the tracking error and its derivative are defined as

$$\begin{aligned} e &= x_1 - x_r \\ \dot{e} &= \dot{x_1} - \dot{x_r} \end{aligned} \tag{3}$$

Taking the second derivative of the tracking error (3) and substituting (2) into that yields to

$$\ddot{e}_1 = f(x) + g(x)u + w \tag{4}$$

where $w = d - \ddot{x}_r$. Similar to [32, 33], two RBF neural networks (NNRBF) are employed to approximate $f(x)$ and $g(x)$ as

$$\hat{f}(x) = \widehat{W}_1^T \emptyset_1(x) \tag{5}$$

$$\hat{g}(x) = \widehat{W}_2^T \emptyset_2(x) \tag{6}$$

where $\widehat{W}_i \in \boldsymbol{R}^{\boldsymbol{n_{ri}}}$ and $\emptyset_i(x)$, i =1,2 are the weight vector and Gaussian activation function of NN, respectively. n_{ri} denotes the number of neurons. By introducing $E = (e, \dot{e})^{\mathrm{T}}$, error dynamics is defined as

$$\dot{E} = AC + B\ddot{e} \tag{7}$$

where $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ and $B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. The control law, u, and the update rules of NN parameters , $\widehat{W}_1$ and $\widehat{W}_2$, are given by equations (8) -(10)

$$u = \frac{v_1}{1 + |\hat{g}|} \tag{8}$$

$$\dot{\widehat{W}}_1 = \alpha_1(\emptyset_1 E^T B - \beta\widehat{W}_1) = \alpha_1(\emptyset_1\dot{e} - \beta\widehat{W}_1) \tag{9}$$

$$\dot{\widehat{W}}_2 = \alpha_2(\frac{v_1\emptyset_2\dot{e}}{1 + |\hat{g}|} - \gamma\widehat{W}_1) \tag{10}$$

where K= [k_1 k_2], α_1, α_2, β and γ are positive real constants and $v_1 = -\hat{f}(x) - KE + \ddot{x}_r$. The recursive equations (9)-(10) involve a call-by-reference property. Moreover, (8)-(10) include nonlinear functions, such as Gaussian and absolute functions, as well as dividing operation all of which make solving these equations time consuming. This control strategy with x_r = 1 is implemented between the code of sending and receiving the pulse train. The latency measurement results are presented in Table 2.

Compared to test 2 of Table 1, it can be seen that the complexity of the control code does not affect the latency value. The main reason of this phenomenon is related to the high speed computation of the CPU in the computer.

In our experiments we used a neuro-adaptive controller - a relatively complex nonlinear algorithm - together with the reported hardware (DAQ, bus, CPU) and drivers. We observed that mean and maximum latencies were dominated by hardware and I/O/driver interactions rather than by controller execution time on this platform. This does not imply that controller complexity cannot affect latency in other setups (for example, at higher loop rates, on less capable CPUs, or with more CPU-bound implementations); therefore, future work will evaluate multiple controller classes (e.g., PID, MPC, and computationally intensive nonlinear controllers) and report per-loop execution times and CPU utilization to quantify algorithm-induced latency.

Remark 6: Stability analysis of the proposed control law (8) and update rules (9)-(10) are out of scope of this paper. Readers are referred to [30] and references therein to explore the stability requirements of the control law (8).

Table 2. The latency measurement for a quad-core processor

	Test Condition	**Td_max**	**Td_mean**	**Test Duration (dd:hh:mm)**
1	Data Saving Disabled	95.67 us	52.81 us	11:17:10

4.4. MULTI-CORE SYSTEMS

Developing CPUs with more cores has provided simultaneous multiple task execution and sped up processes, but according to some references, e.g. [1] and references therein, the increase in the number of cores leads to more contention among processes accessing shared resources, which may increase the latency. In order to investigate the effect of number of cores on the latency, we chose a quad-core Intel® Core i7-4790 processor with base frequency of 3.6 GHz and 4GB memory. The cores are labeled Core 0 to Core 3. The PCIe1816 card was mounted on the motherboard and the Linux kernel and PREEMPT_RT kernel were used the same as the dual-core system. The test results are shown in Table 3 for different core assignment.

Table 3. The latency measurement for the quad-core system

	Test Condition	**Td_max***	**Td_mean**	**Test Duration (dd:hh:mm)**
1	• Four cores enabled • Core 1 assigned to data acquisition • Core 0 assigned to data saving • Display results of the 1st run of every 2 million runs of the code	94.70 us	50.01 us	06:23:43
2	• Four cores enabled • Cores 0 & 1 assigned to data acquisition • Cores 2 & 3 assigned to data saving • Display results of the 1st run of every 2 million runs	102.72 us	50.01 us	07:00:10
3	• Four cores enabled • Cores 0 assigned to data acquisition • Cores 1 to 3 assigned to data saving	100.35 us	50.01 us	07:00:25

	• Display results of the 1st run of every 2 million runs			
4	• Four cores enabled • Cores 3 assigned to data acquisition • Cores 0 to 2 assigned to data saving • Display results of the 1st run of every 5 million runs	94.73 us	50.01 us	07:00:00
5	• Cores 2 & 3 disabled • Cores 0 assigned to data acquisition • Cores 1 assigned to data saving • Display results of the 1st run of million runs	94.61 us	50.01 us	00:22:00
6	• Cores 1 to 3 disabled • Cores 0 assigned to data acquisition and data saving • Display results of the 1st run of every 5 million runs	26.33 ms	51.12 us	00:00:15
7	• Cores 1 to 3 disabled • Cores 0 assigned to data acquisition • Data saving disabled • Display results of the 1st run of every 5 million runs	25.65 ms	51.15 us	00:00:02

When Td_max > 94.0 us test is stop

According to Table 3, the order of core assignment, data saving rate and the number of data displaying do not affect the latency value. While, assigning only one core to the two threads increases Td_max considerably. Generally, none of the above conditions alter the Td_mean.

4.5. LONG-TERM TEST

Latency measurements mentioned in the previous sections were obtained through short-duration tests. But the delay values may increase if the test duration is increased. On the other hand, the final values of maximum and mean latencies, Td_max and Td_mean, are necessary to determine the sampling time of a control application. Therefore, long-term tests are considered for latency measurements. The results of long-term tests for the PCIe1816 card are shown in Table 4 for both the dual-core and the quad-core systems. Also, for better comparison, short-term test values of the card are presented. In Fig. 3, the trend of variation of the maximum latency versus the number of test days for test 2 of Table 4 is illustrated. As seen in Fig. 3, during the first week, Td_max has increased considerably. After the fifteenth day, Td_max converged to its final value and increased by at most 1 us over the following days. Duplicate data points on the x-axis for several days of the test indicate the number of changes in latency that occurred on those days. The Td_mean variations for the two tests under the quad-core system are shown in Fig. 4. As can be seen, T_mean reached its final value quickly (below 200 us), which implies that at the start of a test, the high variation of latency and the low number of samples lead to T_mean being high. In contrast, as the test progresses, the variation of T_mean decreases.

Fig. 5 shows the long-term trend of Td_max in both systems, as well as a short-term test of the dual-core system. Taking into account the results in Table 4 and Fig. 5, conducting a long-term test of a specified system is necessary to accurately determine its maximum latency.

Additionally, the quad-core system experiences approximately 25 us higher latency than the dual-core system.

Table 4. The latency measurement of the quad-core system

	Test Condition*	**Td_max**	**Td_mean**	**Test Duration (dd:hh:mm)**
1	• Dual-core system • Core 0 assigned to AI/AO • Core 1 assigned to data saving	76.76 us	50.01 us	31:01:35
2	• Quad-core system • Cores 0 assigned to AI/AO • Cores 1 to 3 assigned to data saving	101.06 us	50.01 us	35:05:30
3	• Quad-core system • Cores 0 assigned to AI/AO • Cores 1 assigned data saving	101.32 us	50.01 us	31:15:15
4	• Dual-core system • Core 0 assigned to AI/AO • Core 1 assigned to data saving	68.74 us	50.01 us	06:02:30

* For every 1 million runs, results of the 1st run are displayed.

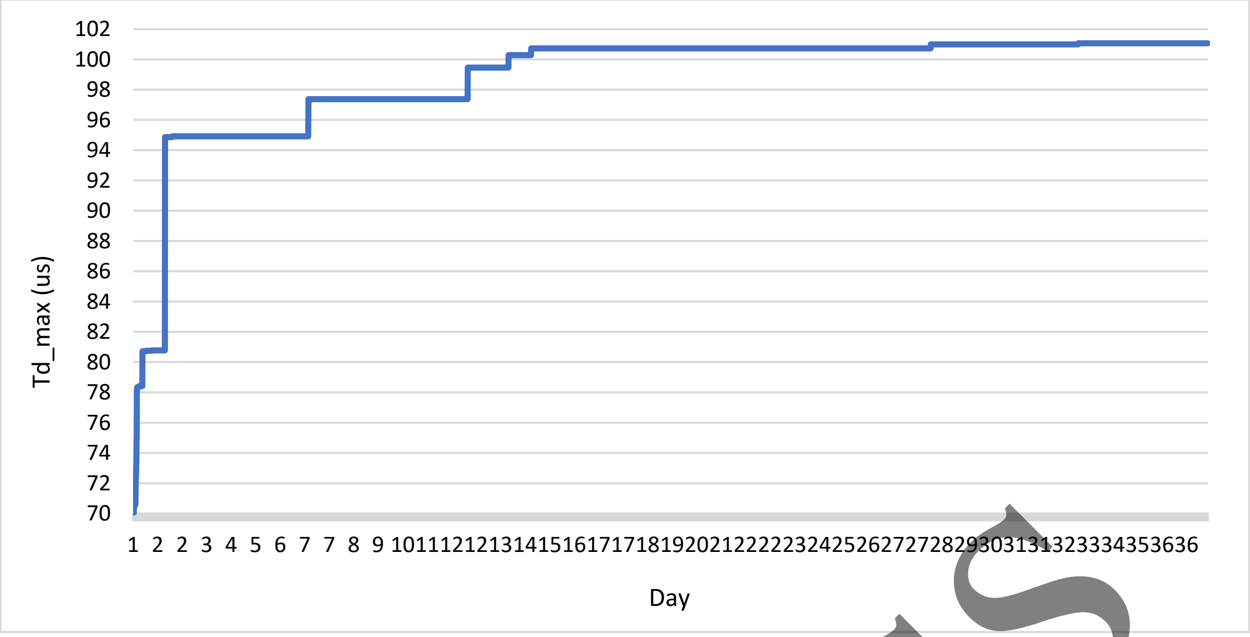


Fig. 3. The trend of maximum latency variation for the quad-core system.

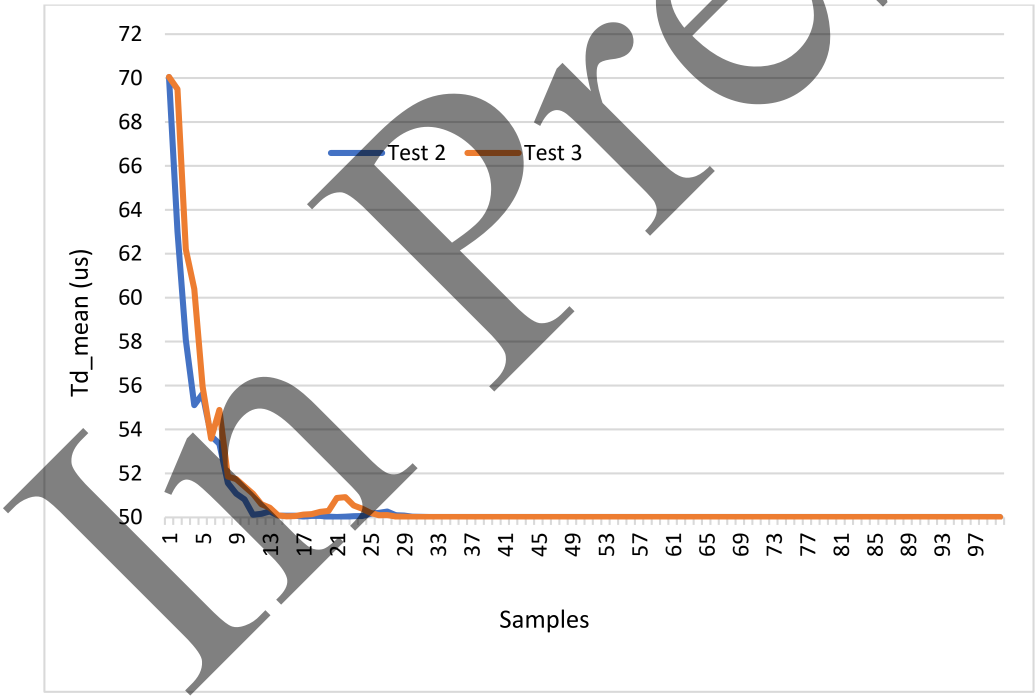


Fig. 4. The trend of mean latency variation for the quad-core system.

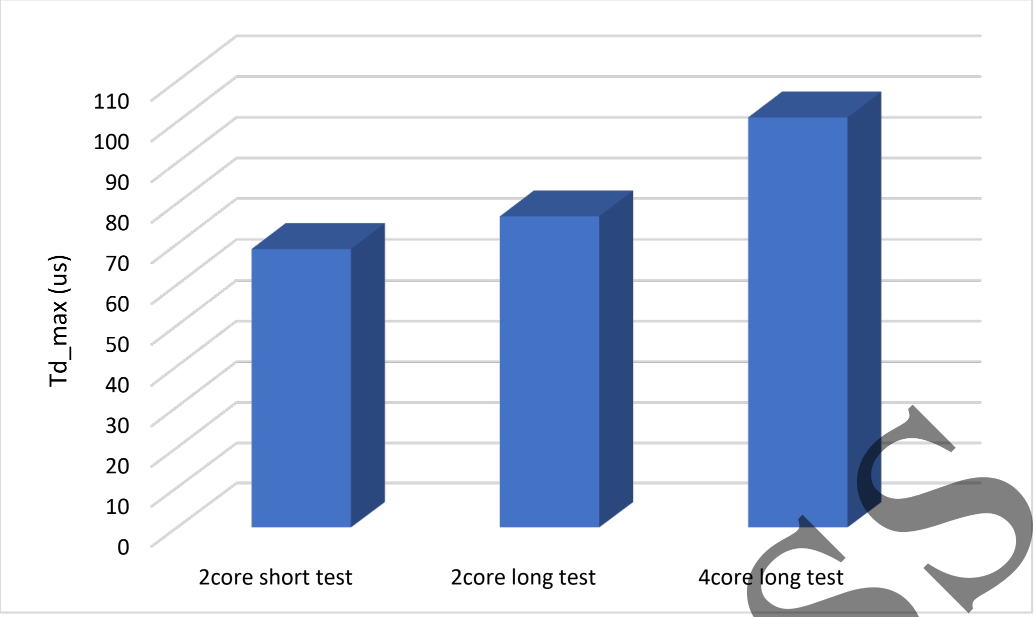


Fig. 5. Td_max comparison for short-term and long-term tests.

4.6. Ubuntu 20.04

To investigate the latency of new releases of Linux, measurement tests are conducted on Ubuntu 20.04.1 with kernel version 5.8.0-43. The results are then compared with those from Ubuntu 18.04 with kernel version 5.3.0-28, as well as with a PREEMPT_RT kernel built on Ubuntu 20.04.1 using patch 5.2.21-rt15. All measurements are made using PCIe1816 card in the quad-core system where Core 0 and Core 1 are assigned to AI/AO and data saving threads, respectively. The measurement results are summarized in Table 5. The results indicate that the latency value does not improve in Ubuntu's new release 20.04 and RT kernel made on it.

Table 5. The latency measurement results of the standard Ubuntu 20.04 in comparison to other kernels

	Test Condition	**Td_max**	**Td_mean**	**Test Duration (dd:hh:mm)***
1	Standard Linux (Ubuntu 18.04)	110.46 us	52.74 us	40:00:00
2	Standard Linux (Ubuntu 20.04)	133.21 us	53.04 us	06:23:00
3	RT Kernel (Patch 5.2.2)	100.24 us	52.68 us	12:15:00

5. RESULTS AND DISCUSSION

Standard Linux is designed for general-purpose computing, prioritizing high throughput and fair resource sharing with a fair scheduling algorithm, which can result in unpredictable latency due to less aggressive kernel preemption and potentially longer interrupt disabling periods. In contrast, Real-Time Linux (e.g., PREEMPT-RT) is built for predictable and consistent response times in time-critical applications, utilizing a priority-based, fully preemptive scheduler that allows high-priority tasks to immediately preempt lower-priority ones, including those in kernel space. It achieves this by minimizing interrupt disabling times through converting handlers to high-priority kernel threads and employing priority inheritance for resource locking to prevent priority inversions.

Building upon these architectural differences, this study investigated the latency of various real-time Linux kernels, concluding that the PREEMPT_RT kernel consistently offers the lowest latency, making it ideal for demanding real-time control applications.

Key factors influencing system latency include:

- **Core Allocation and Threading:** Separating data acquisition and saving tasks onto different CPU cores significantly minimizes latency; running them on a single core increases contention.
- **Hardware Architecture:** The design and port speed of the DAQ card impact latency, with some cards performing better than others. Dual-core systems can even outperform quad-core systems in certain configurations due to reduced resource contention.
- **Concurrent Operations:** Surprisingly, running a concurrent data saving task alongside data acquisition had minimal impact on latency (maximum 16 us increase).
- **Kernel Choice:** The PREEMPT_RT kernel consistently outperformed other kernels in latency.

The study also found that certain modifications did not significantly improve latency, such as parameter tuning (decreasing priority, increasing algorithm complexity, decreasing data saving rate, or upgrading Linux versions) and that mean latency remained consistent across test conditions, suggesting that core allocation and hardware are the primary latency drivers.

Important considerations for real-time system design include:

- **Hard Real-Time Systems:** Require long-term latency measurements to identify maximum latency spikes, which are critical for reliability.
- **Soft and Firm Real-Time Systems:** Short-term tests are sufficient to determine mean latency, as occasional spikes are tolerable.

The reported mean and maximum latencies were computed from extended-duration runs on the original experimental platform. Due to the decommissioning of this platform, we were unable to re-run experiments to obtain additional repeated-trial summaries, such as sample standard deviations or formal confidence intervals. Consequently, formal error margins are not reported in this manuscript. We acknowledge that such measures would improve statistical completeness, particularly for soft real-time analyses where occasional deviations from the nominal sampling period may be acceptable. Future research will address this by exploring individual contributions of scheduling, interrupts, and memory access, along with different hardware and software optimization strategies to further reduce latency in real-time control systems, and we commit to reporting sample variances, confidence intervals, and additional percentile statistics in follow-up work if the testbed can be reassembled.

CONFLICT OF INTEREST

The authors declare that they have no conflicts of interest.

APENDIX

PCI1716 includes 16 single-ended AI channels, a 16-bit A/D converter with up to 250 kS/s sampling rate, and 2 AO channels with a 1 MS/s update rate. PCIe1718 has 16 single-ended AI channels with up to 1 MS/s sampling rate and 16-bit resolution, plus 2 AO channels with a 3 MS/s update rate. NI-6281 (PCI) includes 16 single-ended AI channels, an 18-bit A/D converter with up to 625 kS/s sampling rate, and 2 AO channels with a maximum 2.86 MS/s update rate for a single channel.

In the main latency-measurement setup, each DAQ card's analog output (AO0) is connected to its analog input (AI0) to send and receive a pulse wave. DAQNavi_SDK version 4.0.0.0 was used as the Linux driver for the Advantech PCI1716 and PCIe1718 cards, and NI Linux Device Drivers 2020 was used for the NI-6281 card. Both drivers are compatible with the 4.15.0 kernel of Ubuntu 18.04; therefore, Ubuntu 18.04 was used as the testbed distribution.

Also, a quad-core system is used to test the effect of increasing the number of CPU cores on latency, as explained in subsection 4.4. Note that the DAQ cards are tested separately.

Since data logging and display are necessary for post-analysis in practice, the updated data of latency is displayed in the Linux terminal window and saved in the memory. Minimum, maximum, mean and standard deviation (STD) values of measured delays are considered for data saving and display. The maximum latency is a critical factor in determining the sampling time for a hard real-time control application whereas the mean delay is more relevant to soft and firm real-time systems.

To measure the latency, the C++ code generates a pulse train and measures the time between sending and receiving each edge of a pulse through AO0 and AI0 channels. This time difference determines the latency. If maximum or minimum latency changes, the relevant data is saved and displayed. As stated in Section 3, the high-resolution timer with a resolution of 303 ns measures the latencies. The latency measurement code is shown as follows:

```
void AIO (void* p_device_info)
{
    ErrorCode ret =Success;
    DeviceInformation devInfo = *( DeviceInformation*) p_device_info;
    InstantAiCtrl* instantAiCtrl = instantAiCtrl::Create();
    InstantAoCtrl* instantAoCtrl = instantAoCtrl::Create();
    do{
        double AI_Data[channelCount_AI] = {0};
        double AO_Data[channelCount_AO] = {0};
```

```
u64 scntr = 0;              // u64 = unsigned long long
double LastSentData = 0.0;
timespec LastSentTime ;
timespec Wait ;
double Signal_Level = 5;
Ao_Data[0] = Signal_Level;
if (clock_gettime(CLOCK_REALTIME, & LastSentTime) != 0)
{
        // error in getting time
}
ret = instantAoCtrl->Write(startChannel_AO, channel_Count_AO, AO_Data);
CHK_RESULT(ret);
LastSentData = Signal_Level;
Diffwithtime dwt;
timespec Receive_Time;

do{
        ret = instantAiCtrl->Read(startChannel_AI, channel_Count_AI, AI_Data);
        CHK_RESULT(ret);
        If (AI_Data[0] >= 1.0)
           Signal_Level = 5;
        else if (AI_Data[0] <= 1.0)
           Signal_Level = 0;
        if (Signal_Level == LastSentData)
        {
            if (clock_gettime(CLOCK_REALTIME, & LastSentTime) != 0)
```

```
                {
                    // error in getting time
                }
                Dwt.t1 = LastSentTime;
                Dwt.t2 = RecieveTime;
                Buffer ->WriteValue(dwt);
                LastSentData = Signal_Level;
                LastSentTime = RecieveTime;
                AO_Data[0] = Signal_Level;
                Wait.tv_sec = 0;
                Wait.tv_nsec = 1000;
               Clock_nanosleep(CLOCK_REALTIME, 0, &Wait, NULL)
            }
            ret = instantAoCtrl->Write(startChannel_AO, channel_Count_AO, AO_Data);

            CHK_RESULT(ret);
    } while (ThreadEnable);

} while (false);
instantAiCtrl -> Dispose();
instantAoCtrl -> Dispose();
if (BioFailed(ret)){
    //  declare error code
}
```

```
}
```

Moreover, the C++ code uses two threads to handle the latency measurement and data saving tasks independently. Each thread is assigned to one core of the dual-core CPU to prevent possible migration between the cores.

REFERENCES

1. F. Reghenzani, G. Massari, W. Fornaciari, “The real-time Linux kernel: A Survey on PREEMPT_RT,” ACM Computing Surveys, Vol. 52, No. 1, pp. 1-36, May 2019.

2. T. Knutsson, Performance evaluation of Gnu/Linux for real-time applications, Technical Report, Uppsala University, 2008.

3. N. Asadi, Enhancing the monitoring of real-time performance in Linux, Master of Science Thesis, Mälardalen University, Västerås and Eskilstuna, 2014.

4. R. V. Aroca, D. M. Tavares, G. Caurin, “Scara robot controller using real time Linux,” in IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Zürich, 2007.

5. A. S. Conceição, A. Scolari, A. P. Moreira, P. J. Costa, “A nonlinear model predictive control strategy for trajectory tracking of a four-wheeled omnidirectional mobile robot,” Optimal Control Application and Methods, Vol. 29, No. 5, pp. 335-352, Sep.2007.

6. M. Dobšoviè, M. Kratmüller, “Control an electromechanical system in the real-time Linux environment,” Acta Polytechnica Hungary, Vol. 5, No. 2, pp. 105-112, 2008.

7. B. W. Choi, D. G. Shin, J. H. Park, S. Y. Yi, S. Gerald, “Real-time control architecture using Xenomai for intelligent service robots in USN environments,” Intelligent Service Robotics, Vol. 2, pp. 139–151, Jul. 2009.

8. A. Al Moshi, E. Islam, S. S. Cynthia, A multitasking PC based robotic arm manipulator control system in RT-Linux environment, Ph. D Thesis, BRAC University, Dhaka, 2010.

9. D. Liming, Z. Xianchao, Q. Chenkun, G. Feng, “A real-time walking robot control system based on Linux RTAI,” in Proc. The 2013 International Conference on Advanced Mechatronic Systems, China, 2013.

10. W. Shuai, Z. Heng, T. Han-Qing, J. Lin-Ying, “Implementation of step motor control under embedded Linux based on S3C2440,” in 2012 International Conference on Future Energy, Environment and Materials, Hong Kong, 2012.

11. B. Choi, W. Lee, G. Park, Y. Lee, J. Min, S. Hong, “Development and control of a military rescue robot for casualty extraction task, Journal of Field Robotics, Vol. 36, No. 4, pp. 656-676, Jun. 2019.

12. Z. M. Huang, B. Li, G.Y. Zheng, Q. P. Yuan, X. Q. Ji, B. J. Xiao, J. Zhou, J. J. Huang, R. R. Zhang, T. C. Lu, D. Humphreys, “A new scheme of plasma control system based on real-time Linux cluster for HL-2M,” Fusion Engineering and Design, Vol. 192, 113763, Jul. 2023.

13. R. Delgado, B. J. You, B. W. Choi, “Real-time control architecture based on Xenomai using ROS packages for a service robot,” Journal of Systems and Software, Vol. 151, pp. 8–19, May 2019.

14. P. Orkisz, B. Sapinski, “Hybrid vibration reduction system for a vehicle suspension under deterministic and random excitations,” Energies, Vol. 16, No. 5, pp. 2202, Feb. 2023.

15. J. Ahn, S. Park, J. Sim, J. Park, “Dual-channel Ethercat control system for 33-dof humanoid robot TOCABI,” IEEE Access, Vol. 11, pp. 44278-44286, May 2023.

16. J. Arm, Z. Bradac, V. Kaczmarczyk, “Real-time capabilities of Linux RTAI,” IFAC-Papers on Line, Vol. 49, No. 25, pp. 401–406, Jan. 2016.

17. C. Wang, F. Yang, H. Wang, P. Guo, J. Hou, “Improving real time performance of Linux system using RT-Linux,” Journal of Physics: Conference Series, Vol. 1237, No. 5, p. 052017, Jun 2019.

18 .G. K. Adam, “Real-time performance and response latency measurements of Linux kernels on single-board computers,” Computers, Vol. 10, No. 5, pp. 1-18, May 2021.

19. P. Karachatzis, J. Ruh, S. S. Craciunas, “An evaluation of time-triggered scheduling in the Linux kernel,” in 31st International. Conference on Real-Time Networks and Systems, Germany, 2023.

20. Y. Li, Y. Matsubara, H. Takada, K. Suzuki, H. Murata, “A performance evaluation of embedded multi-core mixed-criticality system based on PREEMPT RT Linux,” Journal of Information Processing, Vol. 31, pp. 78–87, 2023.

21. Q. Shi, L. J. Chen, Z. Qiao, “A measurement tool for interrupt latency based on Linux,” Journal of Physics: Conference Series, Vol. 2476, No. 1, 2023.

22. R. Beamonte, M. R. Dagenais, “Linux low-latency tracing for multicore hard real-time systems,” Advances in Computer Engineering, Vol. 2015, No. 1, pp. 1-8, 2015.

23. D. B. De Oliveira, R. S. De Oliveira, “Timing analysis of the PREEMPT RT Linux kernel,” Software: Practice and Experience, Vol. 46, No. 6, pp. 789–819, Jun. 2016.

24. D.B. De Oliveira, R.S. De Oliveira, T. Cucinotta, “A thread synchronization model for the PREEMPT_RT Linux kernel,” Software: Practice and Experience, Vol. 107, No. 6, pp. 789-819, Jun. 2020.

25. C. Wang, F. Yang, H. Wang, P. Guo, J. Hou, “Improving Real Time Performance of Linux System Using RT-Linux,” Journal of Physics: Conference Series, Vol. 1237, No. 5, pp. 1-6, Jun. 2019.

26. L. Perneel, F. Guan, L. Peng, H. Fayyad-Kazan, M. Timmerman, “Real-time capabilities in the standard Linux kernel: How to enable and use them,” International Journal on Recent and Innovation Trends in Computing and Communication, Vol. 3, No. 1, pp. 131–135, Jan. 2015.

27. Y. Chen, X. Tang, S. Xu, F. Zhu, Q. Zhou, T. H. Weng, “Analyzing execution path non-determinism of the Linux kernel in different scenarios,” Connection Science., Vol. 35, No. 1, pp. 1-21, Dec. 2023.

28. R. Delgado B. W. Choi, “New insights into the real-time performance of a multicore processor,” IEEE Access, Vol. 8, pp. 186199-186211, Oct. 2020.

29. P. A. Laplante, S. G. Ovaska, *Real-Time Systems Design and Analysis; Tools for The Practitioner*. New Jersey, John Wiley & Sons, 4th Edition, 2012.

30. H. Ghiti Sarand, B. Karimi, “Observer based robust neuro-adaptive control of non-square MIMO nonlinear systems with unknown dynamics,” International Journal of Computational Intelligence Systems, Vol. 10, No. 1, pp. 23–33, Jan. 2017.

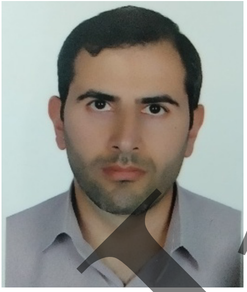
31. S. De, “Uncertainty quantification of locally nonlinear dynamical systems using neural networks,” Journal of Computing in Civil Engineering, Vol. 35, No. 4, Jul. 2021.

32. S. Labioda, M. S. Boucheritb, T. M. Guerra, “Adaptive fuzzy control of a class of MIMO nonlinear systems,” Fuzzy sets and systems, Vol. 151, No. 1, pp. 59–77, Apr. 2005.

33. H. F. Ho, Y. K. Wong, A. B. Rad, W. L. Lo, "State observer based indirect adaptive fuzzy tracking control," Simulation Modelling Practice and Theory, Vol. 13, No. 7, pp. 646–663, Oct. 2005.

BIOGRAPHY

received his M.Sc. degree in Mechatronics Engineering in 2014 and his Ph.D. degree in Electrical Engineering (Control Systems) in 2019, both from Malek-Ashtar University of Technology, Isfahan, Iran. He is currently an Assistant Professor of Electrical Engineering (Control and Mechatronics) at the Faculty of Naval Aviation, Malek-Ashtar University of Technology, Iran. His research interests include nonlinear control systems, multi-agent systems, modern control theory, robotics, and mechatronics.



Hassan Ghiti Sarand received his Master of Science in Electrical Engineering (Control) in 2006 from Amir Kabir University of Technology and his Ph.D. in Electrical Engineering (Control) from Malek Ashtar University of Technology in 2017. He is an R&D Engineer at AEOI, specializing in real-time control systems. His current research focuses on implementing machine learning algorithms in real-time systems.