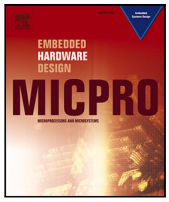
Microprocessors and Microsystems 109 (2024) 105085

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro



PyIgH : A unified architecture of IgH EtherCAT Master based on Python considering hard real-time constraints☆

Raimarius Delgado [a,b], Se Yeon Cho [c], Byoung Wook Choi [c,*]

[a] Department of Electronics Engineering, Myongji University, Yongin, 17058, Republic of Korea
[b] Center for Humanoid Research, Korea Institute of Science and Technology, Seoul, 02792, Republic of Korea
[c] Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, 01811, Republic of Korea

ARTICLE INFO

Keywords:
EtherCAT
Performance evaluation
PyIgH
Real-time systems
Robot control

ABSTRACT

The increasing demand for rapid application development tools, especially those employing high-level languages such as Python, has underscored the importance of utilizing a wide array of popular libraries while addressing real-time constraints in distributed hardware systems. This paper introduces PyIgH, a unified architecture of an IgH EtherCAT master based on Python, specifically designed to satisfy hard real-time requirements in an EtherCAT network. Implemented as a Python module, PyIgH exposes the functionalities and capabilities of an open-source EtherCAT master, facilitating seamless configuration and control of EtherCAT slave devices within the Python runtime environment. Real-time adaptation of the POSIX library, encapsulated within Python, is also utilized to satisfy the timing requirements of EtherCAT. The feasibility of the proposed approach is verified by analyzing the real-time performance in terms of periodicity and in-controller delay of the EtherCAT control task with a 1 kHz cycle. Experimental results demonstrate that PyIgH is suitable for hard real-time applications and serves as a valid alternative to conventional low-level EtherCAT masters. Additionally, a practical application involving motion control of a six-axis collaborative robot showcases consistent performance of PyIgH within a real-time multi-tasking environment.

1. Introduction

Python has emerged as a versatile programming language, experiencing a surge in popularity, as evidenced by its consistent ranking atop IEEE Spectrum's programming language statistics [1]. Its appeal lies in its straightforward syntax and minimal learning curve, rendering it accessible for diverse applications. Python's extensive library ecosystem and rapid development tools have made it indispensable for high-level algorithmic tasks, particularly in data analytics [2] and artificial intelligence domains like machine learning and deep learning [3,4]. However, in the realm of intelligent control systems, Python's utility is challenged by the necessity to process vast amounts of data from intricate sensor-actuator configurations within real-time communication networks, such as fieldbuses [5–7].

Among real-time Ethernet protocols, EtherCAT has emerged as a frontrunner due to its swift cycle times, minimal synchronization jitter, and adaptable topology. Notably, EtherCAT's utilization of standard Ethernet interfaces allows seamless integration of slave devices from various vendors without requiring additional peripherals or adapters. In an EtherCAT setup, effective communication between master and slaves is vital for synchronized control and optimal system performance. Traditionally, EtherCAT masters have been realized through programmable logic controllers (PLCs). PC-based solutions include TwinCAT by Beckhoff Automation [8], and viable open-source counterparts for Linux, namely Simple Open EtherCAT Master (SOEM) by RT-Labs [9,10] and IgH EtherCAT Master from the EtherLab group [11].

EtherCAT control applications traditionally rely on low-level languages such as C/C++ to optimize execution times. However, Python emerges as an appealing alternative, offering simplified software development and enhanced parameter experimentation capabilities. The most prevalent method for controlling EtherCAT devices with Python involves a server-client model. Shen et al. [12] established an Automation Device Specification (ADS)-based Ethernet connection between a Python client and a TwinCAT PLC to manage telescope operations. Despite achieving an average cycle time of 1 ms for 50,000 data points through parallel connections for data transmission and reception, performance limitations stemming from client-side bottlenecks resulted

☆ This work was supported by a grant from the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (MSIT) of the Korean government under Grants NRF-2022R1F1A1064297.

* Corresponding author.
E-mail address: bwchoi@seoultech.ac.kr (B.W. Choi).

https://doi.org/10.1016/j.micpro.2024.105085
Received 3 November 2023; Received in revised form 18 May 2024; Accepted 10 July 2024
Available online 19 July 2024

in maximum response times nearing 4 ms, thus failing to meet real-time requirements. This approach mirrors those of Liao et al. [13] in controlling a cryogenic permanent magnet undulator and Liang et al. [14] in transferring data between a physical robot and its virtual counterpart in a digital twin robot scenario, with average cycle times reaching 9 ms. Beyond real-time performance considerations, significant challenges include development costs and efforts, particularly given the necessity for multiple hardware platforms and expensive TwinCAT runtime licensing. Notably, a Python wrapper for the TwinCAT ADS library, PyADS, is available but exhibits a higher cycle time of 8 ms [15]. To the best of our knowledge, PySOEM [16] remains the sole open-source solution, albeit limited to basic system configuration and connection testing of EtherCAT devices, rendering it unsuitable for real-time control applications. Therefore, an open-source solution is imperative to address the performance issues and alleviate development costs associated with existing Python-based EtherCAT solutions.

This paper presents the development of a Python-based EtherCAT master designed to meet real-time constraints in practical control and automation systems. We devised Python modules to encapsulate the application programming interface (API) and libraries provided by IgH EtherCAT Master, enabling the implementation of an EtherCAT master responsible for configuring and controlling slave devices within a Linux environment. To address real-time requirements, the entire system was developed on a Linux kernel patched with RT-Preempt, following guidelines outlined in [17]. Real-time tasks were implemented in Python using a real-time variant of the POSIX library, also adapted for Python execution as described in a prior study [18]. A performance analysis was conducted to assess the feasibility of our approach, focusing on the periodicity of real-time tasks and in-controller delay [19]. Results were compared with those from traditional EtherCAT master application based on C/C++ and PySOEM. Additionally, we demonstrated the PyIgH's consistent performance through motion control of a six-axis collaborative robot in a practical application. Our findings enable researchers to develop real-time applications for EtherCAT device control, harnessing Python's extensive high-level control libraries to enhance the efficiency, safety, and performance of intelligent control systems.

The main contributions of this study are summarized as follows:

- We designed a unified architecture for an EtherCAT master, facilitating the seamless configuration and control of EtherCAT slave devices within the Python runtime environment, while meeting stringent real-time constraints.
- We developed Python modules to encapsulate the API and libraries of IgH EtherCAT master. This simplifies the EtherCAT master implementation in Python, smoothing out the learning curve typically associated with low-level languages like C/C++.
- To demonstrate feasibility of the proposed approach, a thorough performance analysis is conducted to assess the feasibility of the Python-based EtherCAT master. Results are compared with those obtained from conventional C/C++ applications and the PySOEM library, validating the efficacy of the proposed solution.
- Implementation of motion control for a six-axis collaborative robot, showcasing the consistent real-time performance of the Python-based EtherCAT master in practical multi-tasking applications.
- We provide an efficient means to implement real-time EtherCAT device control using Python to allow utilization of high-level control libraries, enhancing efficiency, safety, and performance in intelligent control systems.

The remainder of this paper is structured as follows. Section 2 provides an in-depth discussion on the software architecture and methodology underlying PyIgH. Experimental findings, including a comparative analysis with existing open-source solutions, are elucidated in Section 3. Additionally, this section presents our reference implementation of a multi-tasking application for the motion control of a six-axis collaborative robot. Finally, Section 4 offers a summary and conclusion of the study.

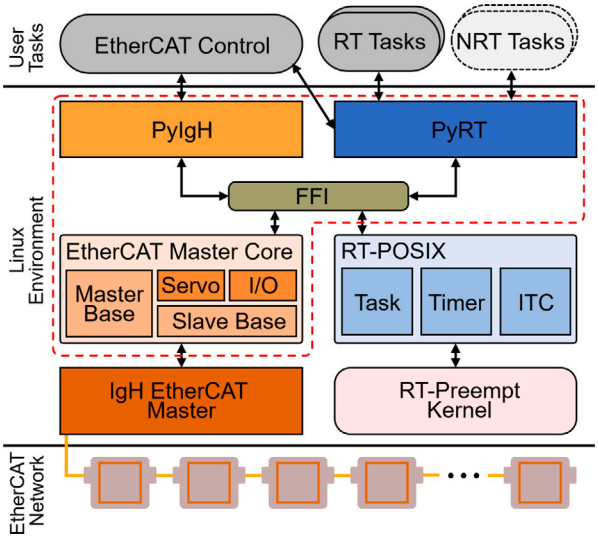


Fig. 1. Software architecture of the Python-based EtherCAT master.

2. PyIgH function implementation

2.1. Software architecture

The software architecture of PyIgH, a comprehensive framework integrating IgH EtherCAT Master with Python for AI integration, is depicted in Fig. 1. The entire system has been developed within a Linux kernel patched with RT-Preempt, following methodologies outlined in prior research [17]. This earlier study demonstrated RT-Preempt's real-time performance comparable to Xenomai. It is also noteworthy that when utilizing Xenomai, the real-time performance of open-source EtherCAT masters may vary depending on the employed EtherCAT device driver, as reported in [9,10]. This variation arises from Xenomai's co-kernel architecture, which necessitates native real-time device drivers for proper processing of EtherCAT dataframes within real-time tasks. Thus, our decision to utilize RT-Preempt as the real-time Linux kernel for PyIgH is justified.

The software architecture of PyIgH, a unified framework integrating IgH EtherCAT Master with Python, is depicted in Fig. 1. The entire system was developed within a Linux kernel patched with RT-Preempt, following procedures outlined in previous research [17]. This earlier study demonstrated RT-Preempt's real-time performance comparable to Xenomai. It is also worth to note that with Xenomai, the real-time performance of open-source EtherCAT masters may vary depending on the utilized EtherCAT device driver as reported in [9,10]. This is due to the co-kernel architecture of Xenomai, where it requires native real-time device drivers to properly process EtherCAT dataframes within the real-time tasks. Thus, justifying our decision to utilize RT-Preempt as the real-time Linux kernel for PyIgH.

In the figure, the left-hand sides of the orange-shaded blocks represent the EtherCAT-related components, while the blue blocks denote those required to create and manage real-time tasks. The real-time kernel is presented here as the pink rectangle with round corners, with the topmost layer indicating the user-space control tasks, in this case RT Tasks and NRT Tasks. The essential parts to develop a real-time application for EtherCAT control are encircled by the red dashed line, consisting of the EtherCAT master core library, the foreign function interface (FFI) to enable the use of dynamic libraries in Python, and the Python modules extended for real-time applications (i.e., PyIgH and PyRT).

The EtherCAT master core was developed to abstract EtherCAT-related system-level routines based on the IgH EtherCAT Master API and library, which do not require further attention from the user.

Table 1
PyIgH function implementation list.

Python functions	Used C functions
ecat_init_slave()	*ecrt_request_master()* *ecrt_master_create_domain()* *ecrt_master_get_slave()* *ecrt_master_slave_config()* *ecrt_slave_config_pdos()* *ecrt_slave_config_sdo8()* *ecrt_slave_config_dc()* *ecrt_domain_reg_pdo_entry_list()*
ecat_activate_domain()	*ecrt_domain_data()*
ecat_master_receive()	*ecrt_master_receive()*
ecat_domain_process()	*ecrt_domain_process()*
ecat_domain_queue()	*ecrt_domain_queue()*
ecat_master_send()	*ecrt_master_send()*
ecat_activate_domain()	*ecrt_domain_data()*
ecat_write_apptime()	*ecrt_master_application_time()* *ecrt_master_sync_reference_clock()*
ecat_get_slave_info()	*ecrt_master_get_slave()*
ecat_master_status()	*ecrt_master_state()*
ecat_slave_config_state()	*ecrt_slave_config_state()*

The Master Base handles the configuration of the EtherCAT master in accordance with the connected slaves. Data acquisition and control of each slave is performed should be according to its respective Slave Base. In its current form, the EtherCAT master core support servo drivers running on the CiA402 profile and digital/analog input–output devices.

On Linux kernels patched with RT-Preempt, real-time tasks are created using POSIX threads which is also used by the Python threading module. However, this setup does not have the functionality to set the priority levels and/or deadlines of real-time tasks. Thus, we employed a Python real-time module presented in a previous work [18] to ensure that the EtherCAT master meets real-time requirements. It is worth to note that the real-time tasks created by this module is governed by a fixed-priority pre-emptive scheduler, SCHED_FIFO policy, implemented by the real-time Linux approach of RT-Preempt. To this end, schedulability of the real-time tasks can be evaluated in accordance to rate-monotonic analysis (RMA) [15]. This means that a set of real-time tasks, including the EtherCAT control tasks of PyIgH is schedulable only of the worst-case response time for every task is less than their deadline.

Moreover, the Foreign Function Interface (FFI) can be implemented using various methods such as CTypes [20], CFFI [21], or the Simplified Wrapper and Interface Generator (SWIG) [22]. Despite concerns raised by the community regarding the performance of CTypes in comparison to SWIG and CFFI for C++ integration, we opted for CTypes due to its inclusion in standard libraries. This choice mitigates potential dependency issues during software distribution or version updates. Additionally, function calls through CTypes release the Global Interpreter Lock (GIL), thereby preventing unwanted delays or system crashes. In contrast, SWIG does not disable the GIL, making it unsuitable for real-time applications. Leveraging CTypes, we developed a module to load the necessary dynamic libraries (e.g., EtherCAT Master Core, RT-POSIX) into Python.

2.2. Function implementation

The Python IgH EtherCAT master (PyIgH) module undertakes the domain activation, slave initialization, distributed clock synchronization, data exchange, and status check functions using the libethercat shared library. The function wrappers start with the name "ecat", detailed in Table 1.

The function *ecat_activate_master()* is responsible for activating the EtherCAT master. This is called after all of the slaves are configured, directly before the task enters the real-time loop. In the configuration phase, process data objects (PDO), the protocol used to transfer data between the slaves in real time, are configured for all connected slave devices. Note that once the master is activated, PDO configurations are not allowed. To initialize an EtherCAT slave, the ecat init slave() function is called. This function performs several important steps.

First, it requests an EtherCAT master instance for the real-time user space operation using the *ecrt_request_master()* function. This allows the application to use the EtherCAT system. Next, a process data domain is created for process data exchanges using the *ecrt_master_create_domain()* function. This domain is crucial for registering PDOs and exchanging them during a cyclic operation. The *ecrt_master_slave_config()* function is then used to create a slave configuration object. This function takes an alias and position as input and matches the vendor ID and product code of the slave with the given address against the given values. If a mismatch occurs, the slave is not configured, and an error occurs. Note that any error that occurs before the PyIgH enters the real-time domain would result to an exit code depending on the error code of the calling function.

The *ecrt_slave_config_pdos()* function is used to specify a complete PDO mapping configuration for the master. PDO entries for the created domain are registered using the *ecrt_domain_reg_pdo_entry_list()* function. This is an important step during the process of initializing the EtherCAT slave. Finally, the usage of distributed clocks is enabled and configured using the *ecrt_slave_config_dc()* function. This ensures synchronous communication between the EtherCAT slave device. The data receive an important sequence to follow during a cyclic task; that is, the stored datagram from the Ethernet device buffer is fetched and the state or the working counters of the EtherCAT frame are determined.

After the initialization process, the following functions should be called sequentially within the real-time loop to ensure data exchange and control of the initialized EtherCAT slave devices. *ecat_master_receive()* and *ecat_domain_ process()* receives EtherCAT datagrams from the slaves and encapsulates it into domains for processing and calculation of the next set of commands. The calculated commands are copied back to the network buffer and sent back to the slaves using the function *ecat_domain_queue()*, which queues the EtherCAT datagrams so that they can be sent to the EtherCAT slaves at the next call of *ecat_master_send()*. Distributed clock synchronization is implemented to match the DC synchronization of the slave devices using the *ecat_write_apptime()* function. Status checks are used to determine specific information and status, such as the status of the master, the product code of the slave device, and the vendor ID. These are the *ecat_get_slave_info()*, *get_master_status()*, *check_slave_state()* functions.

The implemented class hierarchy is structured according to the class diagram in Fig. 2. From the user perspective, the registration of PDO mapping to an instance of the MasterBase class is imperative to enable efficient EtherCAT communication. To streamline this process, we have introduced a SlaveBase class, designed to facilitate PDO mapping registration and enhance user convenience in managing and debugging EtherCAT domains. The SlaveBase class is configured to execute pivotal functions essential for establishing the master–slave relationship, including PDO mapping, enabling DC, and monitoring slave statuses. Furthermore, our hierarchical design includes specialized child classes to cater to specific device functionalities within the EtherCAT network. The SlaveServo class, inheriting from the SlaveBase class, serves as a specialized class for servo drivers employing CANOpen-over-EtherCAT (CoE), and the CiA402 device profile [23] which is commonly utilized in servo control applications. As shown in the figure, this class extends the functionality of the SlaveBase class with additional methods designed specifically for motor control, such as setting the target position, velocity, torque, or operation mode. Conversely, the SlaveIO class, another child class within the hierarchy, is responsible for an interface to digital or analog input–output devices. In the next section, we will provide comprehensive implementation examples of the proposed PyIgH EtherCAT master. Through these examples, we aim to demonstrate the practical feasibility and robust hard real-time performance of our proposed method within actual control systems, thus further validating its efficacy and applicability.

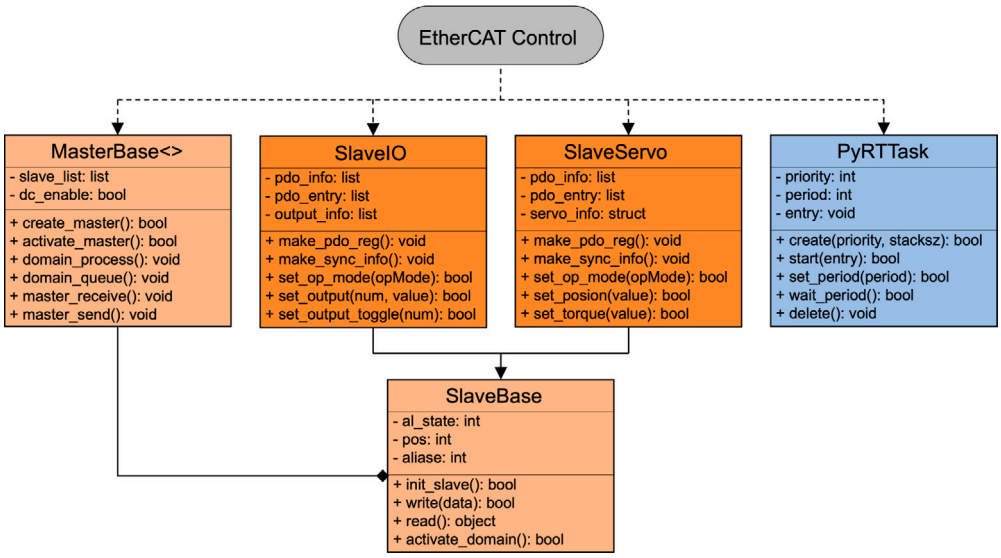


Fig. 2. Class diagram of the Python-based EtherCAT master.

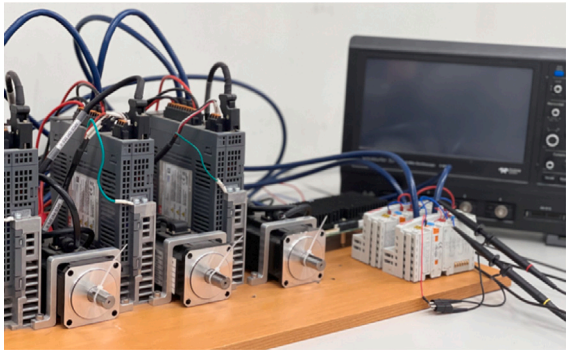


Fig. 3. Experimental setup.

3. Experimental result

This section describes the experimental environment and methodology used to verify the performance of PyIgH in comparison with existing open-source solutions and discusses the experimental results. Motion control a six-axis collaborative robot is also presented to validate its feasibility in a practical control application.

3.1. Experimental environment

The development environment used to perform the experiments is shown in Fig. 3. We utilized an Advantech MIO-5272 board with an Intel i7-6600U processor to control three servo motors manufactured by LS Mecapion. Lubuntu 20.04 served as the Linux distribution, and the Linux kernel was patched with RT-Preempt v.5.15.79rt54 following procedures used in earlier work [15].

For the IgH EtherCAT master, we have installed the latest version available in the stable branch of its Gitlab repository available in [24]. Originally, IgH EtherCAT Master provides native device drivers for both Intel igb and e1000e, the network drivers for MIO-5272. However, native drivers compatible with the selected kernel version are still not available at the time of this writing. Therefore, we utilized generic drivers for all experiments. The data objects and sizes communicated through PDO communication for the servo drivers are shown in Table 2. TxPDO and RxPDO represent transmit and receive PDO, respectively.

Table 2
PDO configuration of the EtherCAT network.

PDO	Index	Object	Size
TxPDO	0 × 6040	Control word	16 bits
	0 × 6060	Drive mode	8 bits
	0 × 60FF	Target velocity	32 bits
	0 × 607A	Target position	32 bits
	0 × 6071	Target torque	16 bits
RxPDO	0 × 6041	Status word	16 bits
	0 × 6061	Actual drive mode	8 bits
	0 × 606C	Actual velocity	32 bits
	0 × 6064	Actual position	32 bits
	0 × 6077	Actual torque	16 bits

3.2. Comparative analysis

The real-time performance of PyIgH has been analyzed in terms of the periodicity of the EtherCAT control task running at a rate of 1 kHz, and the in-controller delay [19]. Experiments were conducted to compare the proposed Python-based approach with IgH EtherCAT master written in C++ and PySOEM to validate whether PyIgH is suitable for hard real-time applications and can serve as a viable alternative to its low-level counterpart.

Considering the rate-monotonic scheduling policy of real-time Linux, the goal of this analysis is to prove whether the real-time task responsible for controlling EtherCAT slave devices can execute periodically without violating temporal deadlines in conformity with rate-monotonic analysis [25].

For the EtherCAT control task, we assume that the execution time is equal to the in-controller delay, defined as the time interval from the master receiving data, processing the data to generate commands, and sending the calculated commands back to the slaves, as shown in the timeline in Fig. 4.

In the figure, R represents the duration for receiving available data from the buffer of the EtherCAT generic driver, P denotes the time during which the next command to be packetized into EtherCAT telegrams is processed, and S is the time required to encapsulate these telegrams into Ethernet frames and copied to the send buffer of the generic driver via direct memory access. The in-controller delay is equal to the sum of R, P, and S, and these can vary depending on the number of slaves, the size of the PDO, and the data processing method implemented by the user. Assuming that the in-controller delay is the execution time, we can easily calculate the worst-case response time

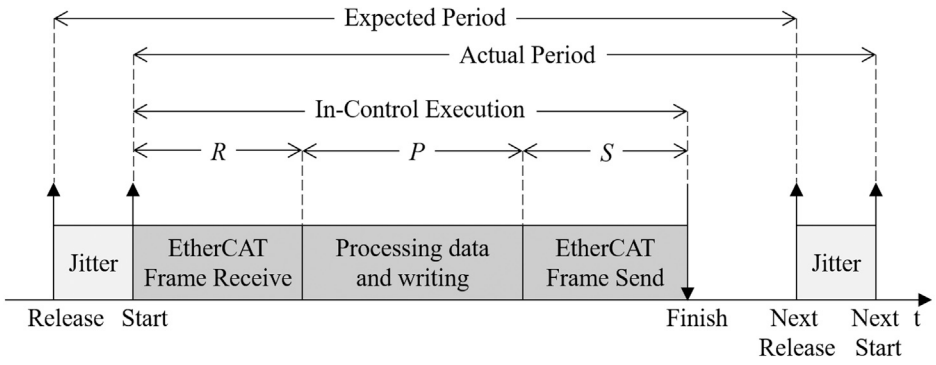


Fig. 4. EtherCAT in-controller delay timing analysis diagram.

(WCRT) and determine whether the real-time task is schedulable and eventually meets real-time requirements.

Algorithm 1 Real-time loop of the EtherCAT control task

Require:
MB = MasterBase instance
SS[n] = SlaveServo instance (n = 3)
while True **do**
  wait_next_period()
  currentTime = read_time()
  MB.read_process_data()
  recvTime = read_time()

  dT = dT+0.001 ▷ 1 kHz period
  posCmd = π sin(πdT)
  procTime = read_time()

  for (i=0; i<n; i++) **do**
    SS[i].set_position(posCmd)
  end for
  MB.send_process_data()
  sendTime = read_time()

  Period = currentTime - prevTime
  prevTime = currentTime
  Receive = recvTime - currTime
  Process = procTime - recvTime
  Send = sendTime - procTime
end while

The pseudo code for the measurement is shown in Algorithm 1. Position commands are generated to track a sine wave with a frequency of 0.5 Hz to minimize the processing time. The application is executed for ten minutes, producing 600,000 data samples.

Table 3 summarizes the results of the performance analysis of each EtherCAT master. The timing measurements are statistically analyzed and are presented as the average (μ), maximum (max), minimum (min), and standard deviation (σ). Note that the in-controller delay ($D_{in\_control}$) is the sum of all time intervals and is not a distribution.

From these experiment results, it is clear that the measured period for all EtherCAT master applications is in good agreement with the expected value of one millisecond with minimal deviation from the statistical mean. As expected, IgH C++ shows the lowest deviation, followed by PyIgH and PySOEM. With regard to the in-controller delay, IgH C++ performs best with the lowest delay across all metrics in comparison with the Python-based EtherCAT masters. This is expected and likely stems from the overhead that the FFI may introduce during its execution. The maximum value, however, shows a difference of only 24 µs between IgH C++ and PyIgH. The in-controller delay is also visualized in Fig. 5. It should also be noted that even with the difference in the average values, real-time deadlines are not violated as maximum delay times are less than one millisecond.

Table 3
Summary of the performance evaluation results. Period is expressed in milliseconds, all others are in microseconds.

Metric	Period	Execution time			$D_{in\_control}$
		Receive	Process	Send	
IgH C++					
μ	1.000000	8.849	0.001	8.963	17.812
max	1.027000	44.000	5.000	64.000	113.000
min	0.972000	7.000	0.000	5.000	12.000
σ	0.000297	0.446	0.031	0.287	–
PyIgH					
μ	1.000000	46.772	2.998	31.341	81.110
max	1.017000	61.000	19.000	57.000	137.000
min	0.984000	43.000	2.000	29.000	74.000
σ	0.000518	0.817	0.158	0.541	–
PySOEM					
μ	1.000000	49.325	2.988	36.904	89.217
max	1.063000	120.000	17.000	102.000	239.000
min	0.936000	48.000	2.000	35.000	85.000
σ	0.000717	0.886	0.189	0.910	–

A comparison of the averages and extreme values above may yield pessimistic results regarding the Python-based EtherCAT master controllers. This is due to the fact that IgH EtherCAT Master was developed natively for C/C++. In this context, we performed a single comparison of the Python-based EtherCAT master to determine the relationships with each C++ counterpart, i.e., whether there is a significant difference in performance. We conducted a statistical analysis using the χ^2 goodness-of-fit test of the measured execution times, where the results from IgH C++ serve as the expected values and the timing measurements from each Python-based EtherCAT master are the observed values. The null hypothesis here is that there is no significant difference between the Python and C++ EtherCAT masters.

We selected a stringent significance level (α) of 0.01 to set a threshold of 99% accuracy for the results with 29 degrees of freedom. This is done to ensure that the results are not determined by coincidence. The calculated critical value is 49.59, indicating that there is a significant difference between IgH C++ and the corresponding Python-based EtherCAT master if its calculated test statistic, denoted by χ^2_α, is greater than the critical value.

The results of the χ^2 test are also shown in the figure. The calculated test statistic of PySOEM is $\chi^2_T = 55.75$, which exceeds the critical limit. This clearly means that PySOEM rejects the null hypothesis and proves a statistical difference with IgH C++. With regard to PyIgH, the test statistic of 48.94 means that the null hypothesis can be rejected, although there were differences measured between the averages and extreme values in this case. From these results, it can be concluded that the proposed Python-based EtherCAT master has the potential to be a feasible alternative to the conventional C/C++ EtherCAT master.

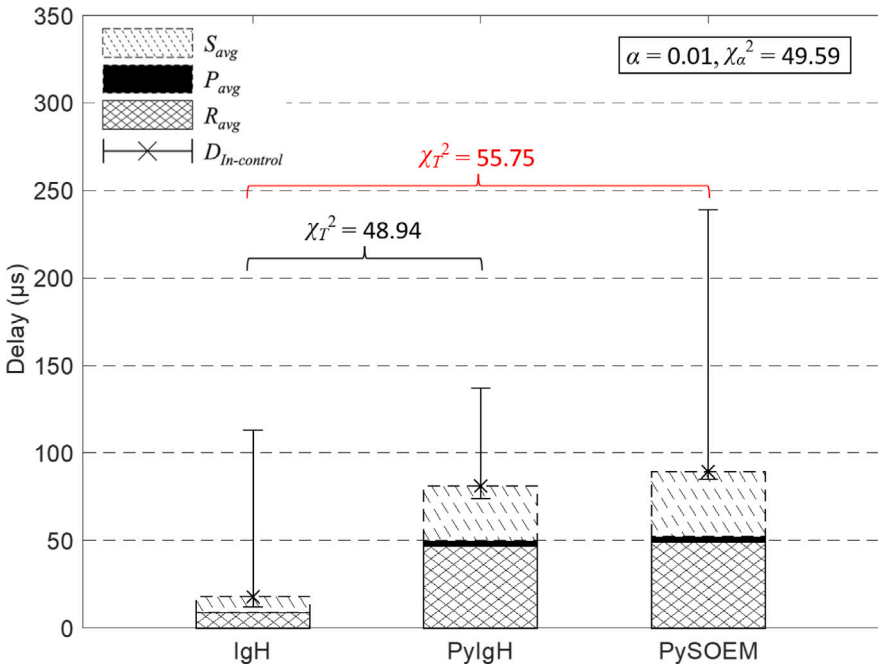


Fig. 5. Visualization of the in-controller delay measurements.

3.3. Motion control of a collaborative robot

To demonstrate feasibility in practical real-time control applications, the proposed Python-based EtherCAT master is employed in a multi-tasking environment for motion control of a six-axis collaborative robot (Indy7, manufactured by Neuromeka [26]). As in the preceding subsections, the period of the control task and the in-controller delay were measured to ensure the schedulability of real-time tasks through RMA in a multi-tasking environment. Also, actual kinematic data were acquired through the encoders of each joint for motion correctness verification. An inverse dynamic was implemented for gravity compensation utilizing the kinematics and dynamics library (KDL) distributed by the Orocos project [27].

3.3.1. Joint space control

To implement joint-space motion of the collaborative robot, real-time periodic tasks are designed to handle the EtherCAT control task employing PyIgH, trajectory generation, dynamics calculation, log control, and ROS2 communication for the user-interface. A single sporadic tasks has been generated to perform an emergency stop of all servo drives in case an error occur on a single joint to prevent accidents. The real-time tasks are deployed on the sample hardware (MIO-5272) and software environment as in the previous section. Fig. 6 shows the task deployment setup of the real-time application with the respective periods and priorities for each task. To ensure synchronization and data correctness shared between the real-time tasks, Python's native queue module was utilized as the inter-task communication (ITC) mechanism.

The Error task was designed with the highest priority to ensure safe operation of the robot. The task will signal each joint (motors) to perform an emergency stop and prevent an accident in the event of abnormal behavior. The Error task is in the sleep state by default, waiting for a signal from the Main Control task through a Python queue. This ensures that it will not affect the cyclic operation of other real-time tasks but will preempt tasks with lower priority levels when the need for an emergency stop is detected.

To ensure that the required joint-space torque throughout the robot's range of motion is compensated for, the Dynamics task has the second highest priority (95), running at a periodic cycle of one millisecond. This task updates the mass matrix information and calculates the gravity while also undertaking Coriolis Effect compensation of the robot with the kinematics and dynamics library (KDL). State information from each joint of the robot is acquired from the Main Control task through shared memory represented by the green block in Fig. 6. In return, the calculated external forces are stored in shared memory to be accessed by the Main Control task.

The Main Control task handles communication with the EtherCAT slaves through an instance of the proposed PyIgH module. Running at a cyclic period of one millisecond, the task acquires joint data from each EtherCAT slave and stores the data in shared memory for state estimation by the Dynamics task. As the Indy7 robot only allows the cyclic torque control mode, a cascaded position-velocity PID controller serves to calculate the torque commands to ensure that each joint moves to the desired position. This task is also responsible for communication with the ROS2 task through a Python queue.

The ROS2 task receives the robot's current state via a Python queue and transmits it to a remote desktop PC on the same local network using ROS2 topic communication for monitoring and visualization purposes. The Log Control task functions to save real-time performance metrics, such as the measured period, jitter, and response times to log data files when a predetermined program runtime is reached.

3.3.2. Result of motion control

In the experiment, the Trajectory task generates a smooth trajectory (velocity profile) from 0 to 90 degrees which are given to all of the joints of the collaborative robot. With a runtime of five minutes, timing measurements were acquired following the same method from the previous sections, as shown in Table 4.

The experimental results demonstrate that all six tasks have consistently achieved their respective periods (mean) with minimal deviation. Furthermore, both the maximum and minimum values exhibit acceptable deviations, remaining within a 5% range of the mean. Regarding response times, the maximum value signifies the worst-case response time (WCRT). It is evident that all real-time tasks have successfully met their respective deadlines, indicating deterministic behavior consistent with Rate Monotonic Analysis (RMA).

For Dynamics and Main control tasks with a period of 1 ms, the measured times were 165.733 µs and 339.378 µs at the corresponding

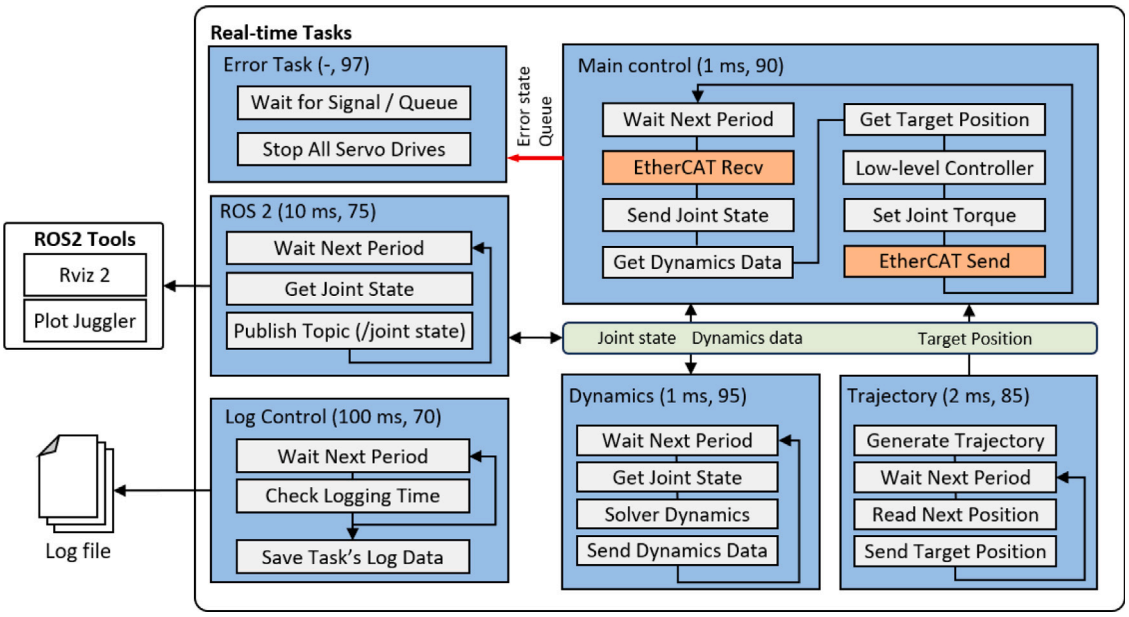


Fig. 6. Real-time task deployment for the motion control of Indy7 employing PyIgH architecture. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 4
Summary of the performance evaluation results. Period is expressed in milliseconds, the rest are in microseconds.

Metric	Period	Jitter	Response	Metric	Period	Jitter	Response
Dynamics (P: 1 ms, pr: 95)				Main control (P: 1 ms, pr: 90)			
Min	0.959	1.086	39.624	Min	0.975	1.028	133.508
Max	1.041	48.328	165.733	Max	1.025	22.405	339.378
Mean	1	16.811	97.555	Mean	1	1.686	225.162
Std.	0.007	4.346	17.978	Std.	0.001	0.795	19.356
Metric	Period	Jitter	Response	Metric	Period	Jitter	Response
Trajectory (P: 2.0 ms, pr: 85)				ROS2 (P: 10 ms, pr: 80)			
Min	1.965	1.171	4.732	Min	9.883	6.309	9.272
Max	2.035	32.46	455.232	Max	10.116	198.234	836.576
Mean	2	2.064	24.338	Mean	10	97.859	326.594
Std.	0.002	1.216	46.769	Std.	0.026	19.451	82.731
Metric	Period	Jitter	Response	Metric	Period	Jitter	Response
Log control (P: 100 ms, pr: 75)				Error (P: 1000 ms, pr: 70)			
Min	99.554	1.454	3.832	Min	999.292	1.766	4.735
Max	100.502	504.362	513.557	Max	1000.642	352.268	358.615
Mean	100	56.924	61.772	Mean	1000	8.338	12.633
Std.	0.082	60.639	61.281	Std.	0.044	28.426	28.872

maximum values. The Trajectory task with a 2 ms period was measured at 455.232 µs at its maximum value. For the ROS2, Log control, and Error tasks with corresponding periods of 10 ms, 100 ms, and 1000 ms, the maximum measured response times were 836.576 µs, 513.557 µs, and 358.615 µs, respectively. The calculated CPU utilization for the real-time application is 73.7%, which indicates schedulability and deterministic response times.

To analyze the effects of the real-time performance has an effect in the physical domain, actual motion of the robot are shown in Fig. 7, and the velocity profile and position values from each joint during the motion of the robot are illustrated in Fig. 8.

Fig. 7 shows a visual representation of the actual movements of the motions from 0 to 90 degrees during motion control of the Indy7 robot. The reference trajectory is generated considering the physical limits (maximimum velocity, acceleration, and jerk) of each joint of the robot in accordance to manufacturer specifications. Fig. 8 illustrates the reference and actual trajectories acquired from each joint of the Indy7 robot during the motion depicted in Fig. 7. The red and pink solid lines represent the target position value input into the controller and the corresponding target velocity value, respectively. The blue and sky-blue dash-dotted lines correspondingly represent the current position and velocity values obtained through encoder sensor feedback data. We have observed that each joint of the robot accurately follows the reference trajectory, a pivotal aspect in assessing the precision and effectiveness of the robotic control system. This achievement underscores the EtherCAT control task's ability to transmit motion target commands while satisfying real-time requirements. This corroborates the reliability of the timing measurements presented in Table 4. In terms of velocity control, certain oscillations were noted. These oscillations potentially suggest that additional gain-tuning and optimization are necessary for the velocity controller to enhance performance and attain smoother motions. Nonetheless, it is pertinent to highlight that the primary objective of this study is to validate the applicability of Python for a pragmatic real-time control system featuring multi-axis EtherCAT slaves, rather than to evaluate the efficacy of low-level controllers.

These results indicate that the proposed unified architecture, PyIgH, effectively meets real-time demands within practical applications, particularly in a multitasking context controlling a dynamic system like

Fig. 7. Motion control of the Indy7 collaborative robot.

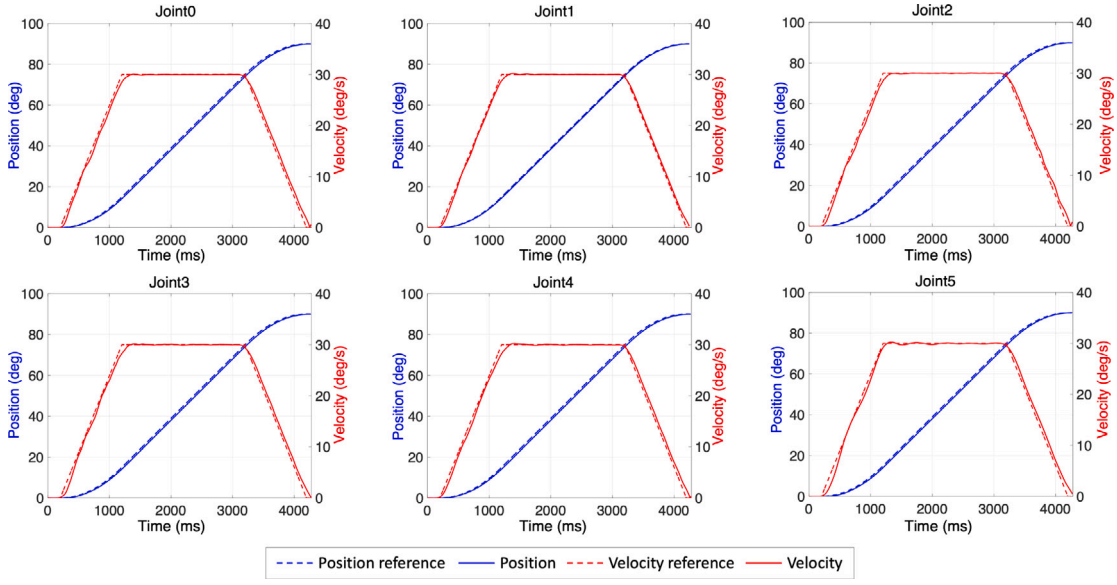


Fig. 8. Reference and actual trajectories of Indy7 in joint-space. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the six-axis collaborative robot. Additionally, when compared to a C/C++-based EtherCAT master, PyIgH exhibits minimal disparities, suggesting its potential as a viable alternative to conventional systems. Furthermore, the incorporation of real-time tasks and PyIgH within the Python runtime environment holds promise for seamless integration with diverse libraries, facilitating advanced applications in artificial intelligence, including machine learning and deep learning.

4. Conclusion and future works

In this paper, we have proposed PyIgH, a Python-based EtherCAT master designed to facilitate and control EtherCAT devices in Python runtime environment. To ensure compliance with the real-time requirements of EtherCAT we have employed integrated a real-time POSIX library for Python [17] to create and manage real-time tasks.

Our assessment of the proposed approach involved a comprehensive comparison with two alternative implementations, IgH C++ and PySOEM, focusing on metrics such as periodicity and in-controller delay. Statistical analysis revealed negligible differences between PyIgH and IgH C++, which means that PyIgH is a viable alternative to traditional EtherCAT masters. Moreover, we validated PyIgH's efficacy by deploying it in joint-space motion control of the commercial collaborative robot, Indy7. Our experiments demonstrated consistent control attainment based on feedback sensor data, affirming PyIgH's reliability in practical control scenarios.

There are several directions for future works. First, we will prioritize the deployment of PyIgH in increasingly complex control systems necessitating coordination among multiple agents, such as mobile robots and manipulators. Additionally, we plan to conduct rigorous assessments to evaluate the schedulability and performance of real-time applications comprising of multiple tasks considering unexpected effects of interfering loads, such as CPU, memory, and network operations. Lastly, we aim to explore the integration of intelligent control systems empowered by deep learning and machine learning techniques for enhanced human-object interaction.

Through these ongoing endeavors, we aspire to advance the field of real-time control systems, fostering innovation and progress in diverse scientific and technological domains.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Byoung Wook Choi reports financial support was provided by National Research Foundation of Korea. If there are other authors, they declare

that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

Raimarius Delgado and Se Yeon Cho contributed equally to this paper and are co-first authors.

References

[1] S. Cass, Top programming languages 2021, 2023, https://spectrum.ieee.org/top-programming-languages-2021, (Accessed 10 February 2023).
[2] A. Nagpal, G. Gabrani, Python for data analytics, scientific and technical applications, in: 2019 Amity Int. Conf. Artif. Intell., AICAI, Dubai, United Arab Emirates, 2019, pp. 140–145.
[3] S.K. Bhoi, K.K. Jena, S.K. Panda, H.V. Long, R. Kumar, P. Subbulakshmi, H.B. Jebreen, An Internet of Things assisted Unmanned Aerial Vehicle based artificial intelligence model for rice pest detection, Microprocess. Microsyst. 80 (2021) 103607, http://dx.doi.org/10.1016/j.micpro.2020.103607, URL https://www.sciencedirect.com/science/article/pii/S0141933120307560.
[4] B.-S. Kim, K.-I. Kim, B. Shah, BANSIM: A new discrete-event simulator for wireless body area networks with deep reinforcement learning in Python, J. Syst. Archit. 126 (2022) 102489, http://dx.doi.org/10.1016/j.sysarc.2022.102489, URL https://www.sciencedirect.com/science/article/pii/S1383762122000662.
[5] C.-H. Chen, M.-Y. Lin, Y.-C. Shih, C.-C. Chen, High-precision time synchronization chip design for industrial sensor and actuator network, Microprocess. Microsyst. 91 (2022) 104507, http://dx.doi.org/10.1016/j.micpro.2022.104507, URL https://www.sciencedirect.com/science/article/pii/S0141933122000679.
[6] R. Delgado, J. Park, B. Choi, Open embedded real-time controllers for industrial distributed control systems, Electronics 8 (2) (2019) 223, http://dx.doi.org/10.3390/electronics8020223.
[7] M. Mareš, O. Horejš, L. Havlík, Thermal error compensation of a 5-axis machine tool using indigenous temperature sensors and CNC integrated Python code validated with a machined test piece, Precis. Eng. 66 (2020) 21–30, http://dx.doi.org/10.1016/j.precisioneng.2020.06.010, URL https://www.sciencedirect.com/science/article/pii/S0141635920301653.
[8] K. Langlois, T. van der Hoeven, D. Rodriguez Cianca, T. Verstraten, T. Bacek, B. Convens, C. Rodriguez-Guerrero, V. Grosu, D. Lefeber, B. Vanderborght, EtherCAT tutorial: An introduction for real-time hardware communication on windows [tutorial], IEEE Robot. Autom. Mag. 25 (1) (2018) 22–122, http://dx.doi.org/10.1109/MRA.2017.2787224.
[9] T. Asfour, M. Waechter, L. Kaul, S. Rader, P. Weiner, S. Ottenhaus, R. Grimm, Y. Zhou, M. Grotz, F. Paus, ARMAR-6: A high-performance humanoid for human-robot collaboration in real-world scenarios, IEEE Robot. Autom. Mag. 26 (4) (2019) 108–121, http://dx.doi.org/10.1109/MRA.2019.2941246.
[10] J. Ahn, S. Park, J. Sim, J. Park, Dual-channel EtherCAT control system for 33-DOF humanoid robot TOCABI, IEEE Access 11 (2023) 44278–44286, http://dx.doi.org/10.1109/ACCESS.2023.3272045.
[11] R. Delgado, B.W. Choi, Network-oriented real-time embedded system considering synchronous joint space motion for an omnidirectional mobile robot, Electronics 8 (3) (2019) http://dx.doi.org/10.3390/electronics8030317, URL https://www.mdpi.com/2079-9292/8/3/317.
[12] T.-C. Shen, P. Galeas, S. Carrasco, J. Seplúveda, F. Huenupan, R. Seguel, R. Augsburger, EtherCAT as an alternative of the next generation real-time control system for telescopes, in: Proc. Software and Cyberinfrastructure for Astronomy VII, Vol. 12189, 2022, pp. 163–175, http://dx.doi.org/10.1117/12.2629232.
[13] C.Y. Liao, C.Y. Wu, J. Chen, D. Lee, H.Z. Chen, Y.S. Cheng, K.H. Hu, K.T. Hsu, Control system for a cryogenic permanent magnet undulator at the Taiwan photon source, IEEE Trans. Appl. Supercond. 30 (4) (2020) 1–5, http://dx.doi.org/10.1109/TASC.2019.2959729.
[14] C.-J. Liang, W. McGee, C.C. Menassa, V.R. Kamat, Real-time state synchronization between physical construction robots and process-level digital twins, Constr. Robot. 6 (1) (2022) 57–73, http://dx.doi.org/10.1007/s41693-022-00068-1.
[15] A. Weber, H. Eichelberger, P. Schreiber, S. Wienrich, Performance comparison of TwinCat ADS for Python and Java, in: Softwaretechnik-Trends Band 43, Heft 4, Gesellschaft für Informatik e.V., 2023.
[16] bnjmnp, PySOEM, 2023, https://github.com/bnjmnp/pysoem, (Accessed 3 April 2023).
[17] R. Delgado, B.W. Choi, New insights into the real-time performance of a multicore processor, IEEE Access 8 (2020) 186199–186211, http://dx.doi.org/10.1109/access.2020.3029858.
[18] S.Y. Cho, R. Delgado, B.W. Choi, Feasibility study for a Python-based embedded real-time control system, Electronics 12 (6) (2023) 1426, http://dx.doi.org/10.3390/electronics12061426.
[19] X. Wu, L. Xie, Performance evaluation of industrial Ethernet protocols for networked control application, Control Eng. Pract. 84 (2019) 208–217, http://dx.doi.org/10.1016/j.conengprac.2018.11.022.
[20] Python Software Foundation, ctypes - a foreign function library for Python, 2022, https://docs.python.org/3/library/ctypes.html.
[21] R. Armin, F. Maciej, C foreign function interface, 2018, https://cffi.readthedocs.io/ (Accessed 20 April 2023).
[22] D.M. Beazley, SWIG: an easy to use tool for integrating scripting languages with C and C++, in: Proc. 4th Conf. USENIX Tcl/Tk Workshop, Monterey, California, 1996, p. 15.
[23] CiA 402 series: CANopen device profile for drives and motion control, 2016, https://www.can-cia.org/can-knowledge/cia-402-series-canopen-device-profile-for-drives-and-motion-control (Accessed 24 March 2023).
[24] IgH EtherCAT master for Linux, 2023, https://gitlab.com/etherlab.org/ethercat.
[25] S.-Y. Lee, M. Sung, Design and implementation of an ethernet-based linear motor drive for industrial transport systems, IEEE Access 9 (2021) 33061–33074, http://dx.doi.org/10.1109/access.2021.3060856.
[26] Collaborative robot, 2023, https://en.neuromeka.com/cobot (Accessed 4 April 2023).
[27] H. Bruyninckx, Open robot control software: the OROCOS project, in: Proc. 2001 ICRA. IEEE Int. Conf. Robot. Automat. (Cat. No.01CH37164), Seoul, South Korea, 2001, pp. 2523–2528.

Raimarius Delgado received B.S., M.S., and Ph.D. degrees in Electrical and Information Engineering from Seoul National University of Science and Technology, Seoul, South Korea, in 2014, 2016, and 2021, respectively. He was a medical control software research engineer with Koh Young Technology, Yongin, South Korea, from 2021 to 2022. He was a postdoctoral researcher with the Center for Humanoid Research (formerly Center for Intelligent & Interactive Robotics), Korea Institute of Science and Technology (KIST), Seoul, South Korea from 2022 to 2024. In the middle of 2024, he joined the Department of Electronics Engineering, Myongji University, Yongin, South Korea, where he is currently an Assistant Professor. His current research interests include mixed-critical systems, intelligent embedded systems, and software architecture for intelligent robotics.

Se Yeon Cho received the B.S. degree in Electronic Engineering from Shinhan University, Uijeongbu, Korea, in 2020, and the M.S. degree in Electrical and Information Engineering from Seoul National University of Science and Technology in 2023. From June 2019 to June 2021, he worked as a software engineer at Onpoom Co., Ltd., Seoul, South Korea. He is currently a Medical Control Software Engineer at Koh Young Technology, Yongin, South Korea. His research interests include real-time systems, industrial control, automation, and embedded systems.

Byoung Wook Choi received M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Seoul, South Korea, in 1988 and 1992, respectively. He was a Principal Research Engineer with LG Industrial Systems from 1992 to 2000 and a Professor with Sun Moon University from 2000 to 2005. He was the CEO of Embedded Web Company Ltd., from 2001 to 2003. He was a Senior Fellow at Nanyang Technological University, Singapore, from 2007 to 2008. He is currently a Professor at the Department of Electrical and Information Engineering and was the Dean of Graduate School, Seoul National University of Science and Technology. He has published textbooks on embedded Linux. His current research interests include real-time system design, embedded systems, and intelligent robot software.