



# RT-AIDE: A RTOS-Agnostic and Interoperable Development Environment for Real-Time Systems

Raimarius Delgado , Member, IEEE, Yong Hwan Jo, Student Member, IEEE, and Byoung Wook Choi , Member, IEEE

**Abstract**—This article presents RT-AIDE, a real-time operating system (RTOS) Agnostic development environment considering performance evaluation and interoperability with nonreal-time tasks. Most real-time systems adopt RTOSes owing to their multitasking environment and priority-based scheduling. However, selecting for the appropriate RTOS according to application requirements is a complicated process as each RTOS has its own API and semantics. With open-source distributions real-time performance evaluation is also an open problem. RT-AIDE addresses this issue through the portable components to ensure execution across various RTOSes without modifying the source code, which has been proven to reduce development costs and effort as well. We also propose a new metric called real-time performance index to evaluate the behavior of the running application. To validate feasibility, theoretical analysis and practical experiment has been conducted on an actual control system based on Xenomai and RT-Preempt.

**Index Terms**—Interoperability, real-time development environment, real-time systems, RT-Preempt, Xenomai.

## I. INTRODUCTION

NOWADAYS, complex control systems are found everywhere: in automobiles and other forms of transportation, industrial automation, smart factories, and in the aviation, healthcare, and robotics industries [1]–[3]. As most of these

control systems typically share the same workspace with humans, they are considered safety-critical systems, meaning that any task failure can damage the infrastructure, or worse, can cause worker accidents. To guarantee the safety of control systems, real-time requirements should be satisfied [4]. In general, real-time operating systems (RTOS) are employed in developing real-time systems. In an RTOS, tasks support strict deadlines and are scheduled based on priorities.

In this article, selecting for the appropriate RTOS according to the application requirements is very important. Currently, RTOSes are available either as commercial or open-source distributions. In the industrial case, commercial RTOSes are mostly utilized owing to their availability, continuous technical support, and performance. However, they may often run only on vendor-specific hardware and are usually distributed in black box, which hinders integration to more complex systems [5]. Expensive licensing and royalties are also critical points to consider. Open-source solutions such as Xenomai [6] and RT-Preempt [7] to overcome these shortcomings and has been proven as valid alternatives to commercial RTOSes [8].

During the selection process, one of the most relevant criteria is the real-time performance of the RTOS. Whether the tasks can satisfy real-time requirements. To compare the performance and obtain credible data samples, a benchmarking application should be executed across various RTOSes. Due to the different API and semantics, running the same application requires a separate source code for each RTOS that increases development costs and level of effort required. To this end, a strategy to allow execution with minimal modification on the application source code is utmost necessary.

Furthermore, a method to analyze timing measurements is also needed to easily determine the RTOS with the best real-time performance. In this article, the most common method of evaluating the real-time performance is to use a benchmarking tool to assess scheduling latency. *cyclictest* [9], for example, is the most popular benchmarking tool for Linux systems. It measures the difference between the actual and calculated release times of a “dummy” real-time task. Then, the timing measurements are statistically analyzed. The RTOS with the relatively lowest standard deviation is determined to have the “best” performance.

Although this might be effective for applications with only a single task, this approach may not yield meaningful results

Manuscript received 28 December 2021; revised 12 May 2022; accepted 29 May 2022. Date of publication 13 June 2022; date of current version 3 March 2023. 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 Grant 2019R1F1A1063547 and Grant 2022R1F1A1064297. Paper no. TII-21-5796. (Corresponding author: Byoung Wook Choi.)

Raimarius Delgado is with the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, South Korea, and also with the R&D Center of Koh Young Technology, Yongin 16864, South Korea (e-mail: raim.delgado@kohyoung.com).

Yong Hwan Jo and Byoung Wook Choi are with the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, South Korea (e-mail: jyh159@seoultech.ac.kr; bwchoi@seoultech.ac.kr).

Color versions of one or more figures in this article are available at <https://doi.org/10.1109/TII.2022.3182790>.

Digital Object Identifier 10.1109/TII.2022.3182790

in a multitasking environment because each task may produce a different trend on each RTOS. Moreover, this method does not account for the actual computational load to determine response times. A performance evaluation method which assesses periodicity and response times in a multitasking environment is essential.

To address these issues, this article presents a development environment termed RT-AIDE, which stands for a real-time operating system-agnostic and interoperable development environment (RT-AIDE). RT-AIDE aims to simplify the design and evaluation of real-time systems. First, RT-AIDE has portable real-time components responsible to provide abstraction of applications via portable real-time components. Note that the term portable refers to the ability to develop real-time applications that can be executed across various RTOSes without the need to modify the application source code. Thus, developers may focus more on designing task functionalities as knowledge about the system internals is no longer required. It also offers inter-operation components to integrate nonreal-time (NRT) external libraries within real-time tasks, without violating real-time requirements.

To evaluate the schedulability and deterministic behavior of the real-time application, this article proposed a new metric called real-time performance index (RTPI). First, the proposed method categorizes timing measurements into three performance indicators: determinism, predictability, and timeliness. The average of these indicators formulates the RTPI. In a multitasking environment, the weight of each real-time task is calculated by the number of task activations within a hyperperiod [10]. Simulations and actual experiments have been performed to validate the feasibility of RT-AIDE. An Intel-based embedded platform has been implemented with the real-time Linux extensions of Xenomai and RT-Preempt. Two real-time applications with multiple tasks are designed using RT-AIDE, and they have been deployed on each of the real-time extension. Finally, timing measurements has been acquired and the results are evaluated using the proposed RTPI.

The rest of this article is organized as follows. Section II reviews the state-of-the-art and summarizes the main contributions of this article. Section III presents the detailed system model of RT-AIDE. In Section IV, formulation of the proposed RTPI is presented. Simulation and experimentation results are respectively shown in Sections V and VI. Section VII presents the limitations of RT-AIDE and the probable threats to the validity of experiment results. Finally, Section VIII concludes this article.

## II. RELATED WORKS AND CONTRIBUTIONS

This section thoroughly analyzes the state-of-the-art related to real-time software framework and their features. Then, the main contributions and advantages of RT-AIDE in comparison to the related studies are thoroughly discussed.

### A. Related Works

In the low-level controller of a control system, there are various software architectures that guarantees real-time response

TABLE I  
SUMMARY OF RELATED WORKS AND THEIR FEATURES

Development Environment	Real-time	RTOS-agnostic	RT/NRT Interface	Performance Evaluation
PODO [11]	Y	N	Y	N
XBotCore [12]	Y	N	Y	N
Mauve [13]	Y	N	N	Y
RTHybrid [14]	Y	Y	N	N
ExSched [15]	Y	Y	N	N
RT-ROS [16]	Y	N	Y	N
RT-AIDE	Y	Y	Y	Y

Y and N, respectively, refer to the feature being supported/unsupported.

of the entire system. For instance, PODO [11] is the framework used by KAIST within HUBO during the Darpa robotics challenge (DRC) finals. Its control system has real-time control capabilities based on Xenomai. A daemon process handles communication between hardware devices and user space tasks, which results to additional runtime overheads. A similar real-time software architecture is XBotCore [12], which is specifically developed to support various EtherCAT based robots.

Gobillot *et al.* [13] presented an approach involving domain-specific languages of real-time software architectures for robotic applications. This approach is highly commendable from a software perspective; however, the aim of this article is to provide a real-time development environment which can accommodate both open source and commercial RTOSes. An effort to generalized real-time software in neuroscience research, Amaducci *et al.* [14] proposed RTHybrid. It can be executed across the real-time Linux extensions of Xenomai and RT-Preempt, which is similar to the goals of this article. However, it was developed to run a specific application, and does not offer a complete development toolchain. ExSched [15], is a framework to implement real-time schedulers without modifying the kernel code. The schedulers are implemented as external plugins, which results to high runtime overheads which render this unfit for practical real-time applications.

### B. Research Gap and Contributions

With the aim to simplify and reduce the development effort to design real-time applications, RT-AIDE has been developed with the following major features: RTOS-agnostic; RT/NRT interface; and performance evaluation method.

RTOS-agnostic refers to a real-time development environment with the ability to run real-time applications across various RTOSes with minimal user intervention; without the need to modify the source code. RT/NRT interface refers to the ability to access external NRT libraries within real-time tasks for rapid development, without jeopardizing hard real-time requirements. Finally, performance evaluation denotes the integration of a schedulability and periodicity tests to analyze the real-time behavior of the running application. Although this section has presented several related works, RT-AIDE is the only solution that covers all these objectives, as shown in the summary of related studies and their features in Table I.

RTHybrid and ExSched are the only other solutions with RTOS-agnostic features. In case of the RT/NRT interface, Wei

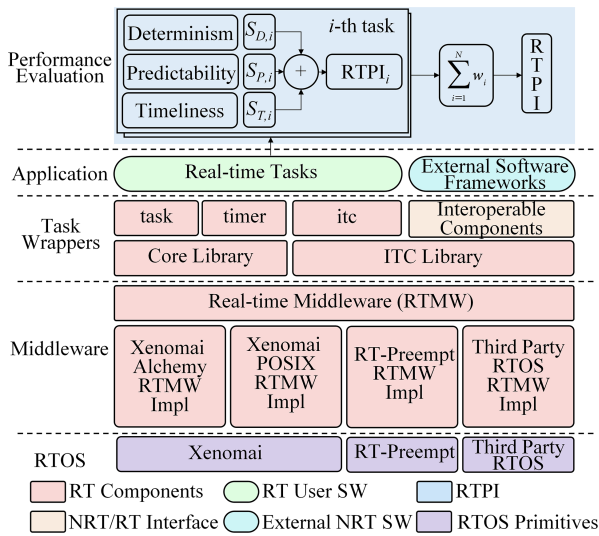


Fig. 1. Software architecture of the proposed real-time development environment called RT-AIDE.

*et al.* [16] presented a single hardware solution called RT-ROS, however, flexibility and extensibility of this approach is questionable because it only available for Intel CPU. The researchers of PODO have also presented a motion generation interface between robot operating system (ROS) and their Daemon process. XBotCore employs cross-domain datagram (XDDP) protocol of Xenomai to satisfy this requirement. For the performance evaluation, Mauve employs rate-monotonic analysis [17] to evaluate the worst-case response time (WCRT) of real-time tasks, which has also been adhered in the proposed RTPI.

The main contributions and features of RT-AIDE can be summarized as the following.

- 1) Develop real-time portable components to abstract applications and execute them on various RTOSes without modifying the source code.
- 2) Provide a communication interface between NRT and real-time tasks specifically for Xenomai through the XDDP as implemented and evaluated in [18].
- 3) Propose a performance index to evaluate the deterministic behavior and schedulability of real-time tasks. The RTPI has been formulated to provide simplified interpretation of timing measurements, considering multitasking.

### III. SYSTEM MODEL

This section describes the system model of the proposed real-time development environment, RT-AIDE. RT-AIDE consists of five major layers: the RTOS; middleware; task wrappers; application; and performance evaluation layers.

Fig. 1 describes the software architecture of RT-AIDE. From the bottom, the RTOS layer denotes the installed RTOS on a selected hardware. In this article, we focus on open-source real-time Linux extensions: Xenomai and RT-Preempt. In theory, RT-AIDE can support any RTOSes, even commercial distributions, if the required toolchain and real-time middleware (RTMW) implementation is included. The middleware layer consists of the RTMW and RTMW implementations (RTMW Impl) for each

available RTOS. The RTMW consists of a Python-based RTOS kernel detector module and compiler utility which automatically selects the required RTMW implementation when compiling applications. On the other hand, the RTMW implementation refers to the necessary libraries and headers of the corresponding RTOS, and RT-AIDE requires the features related to task creation, timer management, and inter-task communication mechanisms (e.g., mutex, semaphores, mailbox, etc.).

The task wrappers layer consists of APIs and shared libraries which are called within the real-time application. The task wrappers are generalized for all RTOSes; thus, users can develop real-time tasks without the knowledge about system internals and the application can be executed across RTOSes without modifying the source code. Task generation and timer management are included in the Core library, while the ITC mechanisms are included in the ITC library.

The available ITC mechanisms within RT-AIDE are mutex, semaphore, and message queue. The application layer consists of the periodic real-time tasks and external libraries (e.g., ROS). For the performance evaluation of each mechanism, readers can refer to a related study in [19]. The interoperable components represent the pipe-like interface between NRT and real-time tasks specifically for Xenomai. The performance evaluation layer evaluates the behavior of the real-time application through the proposed RTPI (refer to Section IV).

#### A. Portable Real-Time Components

The portable real-time components of RT-AIDE consist of the RTMW and real-time task wrappers. The RTMW is mainly responsible to ensure that a real-time application is compiled and can be executed across various RTOSes with minimal user-intervention. The different elements of the RTMW include the RTMW implementations, the kernel detector module, and the compiler utility.

The RTMW implementation contains the primitives of the respective RTOS, which are the headers and libraries for features related to task creation, timer management, and ITC mechanisms. In the case of Xenomai, this refers either to the native API (Alchemy) or the POSIX skin. On the other hand, RT-Preempt uses the standard POSIX library. Aside from the Linux-based real-time extensions, third-party RTOSes can be easily integrated to RT-AIDE if their respective task creation, timer management, and ITC libraries are available. These are necessary to compile the core and ITC libraries in the task wrappers layer.

As shown in In Fig. 1, the task wrappers layer is the one directly exposed to the user-space application layer. Its main responsibility is to wrap each RTMW implementation into general user-space APIs such that users can develop real-time applications without knowledge of the system internals. Some examples of the RT-AIDE task and timer wrappers necessary to create periodic real-time tasks are the following.

- 1) `aide_create_rt_task(*task, name, prio)` creates a real-time task. `*task` is a pointer to the task descriptor that they should be initialized to create a user-space real-time task. Here, `prio` denotes the priority of the task with a minimum



value of 0 (lowest priority) and a maximum value of 99 (highest priority).

- 2) *aide\_set\_task\_period*(\*task, period) configures the real-time task to run cyclically every period in nanoseconds.
- 3) *aide\_start\_task*(\*task, \*entry(\*arg), \*arg) starts the execution of the real-time task to run the function pointer represented by \*entry with the argument \*arg. If the function does not require an argument, NULL should be passed as an argument to \*arg, or *aide\_start\_task\_noarg*() should be used.
- 4) *aide\_wait\_period*(\*overruns\_cnt): This should be called within the function of a periodic task to wait for the next scheduled entry point after the current iteration has expired.

To design real-time tasks with RT-AIDE, the Task Wrappers should be included in the application source code. This ensures that the real-time application can be executed across RTOSes without any modifications.

It also worth to note that, all the wrappers are implemented as inline functions, thus overheads are negligible in the actual application. Moreover, most of the APIs are only called before the task enters the real-time context, thus these overheads shall have minimal effect on the overall real-time performance of the system.

The *kernel\_detector* is a Python-based tool which identifies the underlying RTOS and stores the RTOS information in a JSON-based configuration file (*config.json*). The configuration file consists of two keys: “rt\_linux” and “rt\_skin,” which refers to the real-time Linux extension and the real-time skin, respectively. If the kernel detector module detects RT-Preempt, the value of “rt\_linux” is “rt\_preempt” and “rt\_skin” will be “null”. In case of Xenomai, “rt\_linux” is equal to “xenomai,” and the user should select the desired Xenomai skin on the terminal. The user can select between the native or POSIX skin.

In case that the kernel detector module does not recognize any RTOS, both of keys would have a null value. It is advisable to invoke the kernel detector module upon reboot or before compilation to ensure that RT-AIDE will select the proper dependencies according to the running RTOS. The *Compiler utility* is a *Makefile* with Node.js-based function to extract the values from the configuration file. The compilation process of a real-time application, *sample*, with RT-AIDE is shown in the sequence diagram in Fig. 2.

To implement the task creation and timer management APIs, the source code of the application should include the header file “core.h.” Before compilation, it is advisable that kernel detector has been invoked to detect the kernel being used and to generate is the configuration file. When the compiler is invoked, the compiler utility reads the configuration file and interprets the values of the JSON keys as environmental variables of the compiler. After the source code has been compiled and assembled as object binaries, the linker combines it with the respective RTMW implementation. After the process has been completed, an executable binary file is generated.

To highlight the advantages of RT-AIDE, the development effort and costs are calculated using the constructive cost model (COCOMO) [20] for an application with one periodic real-time

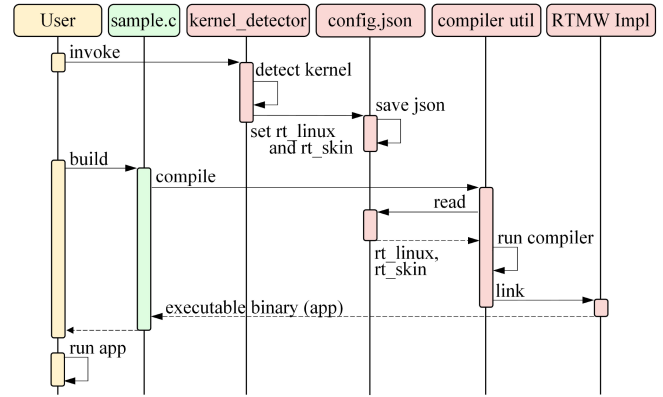


Fig. 2. Sequence diagram of the compilation process in RT-AIDE of an example real-time application, termed sample.

TABLE II  
COCOMO RESULTS FOR A SINGLE REAL-TIME TASK

RTOS	LOC	Effort (Person-Month)	Time (Month)	Costs (KRW)
Xenomai (Alchemy)	30	0.04	0.76	125,403
Xenomai (POSIX)	39	0.06	0.85	188,105
RT-Preempt	39	0.06	0.85	188,105
RT-AIDE	25	0.03	0.69	94,053

task, which prints a string every second. COCOMO is a regression model that estimates the programming effort and costs based on the lines of code (LOC). Table II gives the results focusing on development effort, time, and cost.

The development costs are calculated based on the GNI per capita of South Korea for the year 2020, which is 37621000 KRW. It is clear that RT-AIDE has fewer LOC, which eventually results to lower development costs and less effort. Especially, in a scenario where the same application should be executed in various RTOSes (e.g., comparison of the real-time performance), the source code for each RTOS should be written due to their different APIs and semantics.

In this article, the development effort of Xenomai (Alchemy) should be added to each of the POSIX-based RTOSes. Thus, the calculated overall development effort for the sample application is 0.16 person-month, which is five time greater than when it has been developed under RT-AIDE. The development costs show a similar trend with the total value of 501613 KRW compared to 94053 KRW.

## B. Interoperable Components

Traditionally, task functionalities are developed by hand to execute, specifically in the real-time context. External NRT libraries, such as ROS for rapid development are gaining popularity, and their integration with real-time software is in great demand. However, their direct implementation in the real-time context, particularly Xenomai, would result in an event called mode switching [18]. Where, the real-time task loses its real-time capabilities making it vulnerable to stability problems such as priority inversion, or worse, a kernel panic.

In this article, RT-AIDE is equipped with a pipe mechanism called the XDDP protocol, which serves as the medium between real-time and NRT tasks. Instead of directly accessing NRT resources, real-time tasks are connected to NRT tasks that perform the required functionalities through the XDDP interface. With this solution, external NRT libraries can be integrated with real-time tasks, without jeopardizing hard real-time requirements. The required libraries and headers for this mechanism is wrapped inside the RT-AIDE ITC library. For the performance evaluation and a detailed example of its stand-alone implementation on an actual control system, readers can refer to the work of Delgado *et al.* [18].

#### IV. REAL-TIME PERFORMANCE EVALUATION CONSIDERING A MULTITASKING ENVIRONMENT

Aside from the algorithmic performance of control systems [21], [22], in practice, it is especially important to determine the RTOS with the best real-time performance according to application requirements. For example, the RTOS with the best periodicity is often determined, whether it has the relatively lowest standard deviation [19]. However, in a multitasking application, the trends of these differences may not be consistent across all the real-time tasks.

To address this issue, this article provides a performance evaluation method which interprets timing measurements and categorizes them into three qualitative criteria based on the determinism ( $S_D$ ), predictability ( $S_P$ ), and timeliness ( $S_T$ ) of the real-time tasks. These qualitative criteria are associated with the suitable quantitative performance indicators, such as the periodicity, the difference between the actual response times and the WCRT, and the ratio of deadline misses.

These performance indicators can describe the real-time behavior of a real-time task on a scale of 0 to 10, with 10 being the highest. A higher score means that the real-time task can satisfy the respective qualitative criterion. The RTPI for the  $i$ th task in a set of  $N$  number of real-time tasks,  $\tau = \{\tau_1, \tau_2, \dots, \tau_N\}$ , is determined according to the averages of the three performance indicators, owing to their importance being equally preferred

$$\text{RTPI}_i = \frac{1}{3}(S_{D,i} + S_{P,i} + S_{T,i}). \quad (1)$$

For an application with multiple tasks, averaging the RTPI of each task is not viable due to the different task priorities. In other words, the calculation of the overall RTPI should also consider priority levels. This article, for the first time, provides a solution that assigns a weight proportional to the number of iterations of a task within a hyperperiod [10].

##### A. Performance Indicators

This section describes the derivation of the performance indicators based on the three qualitative criteria of the determinism, predictability, and timeliness of real-time tasks based on their timing measurements. The dataset is represented as a collection of tuples for the  $i$ th real-time task

$$\Pi_i = \{ \langle p_i^k, r_i^k \rangle \}_{k=1}^K \quad (2)$$

where each element in  $\Pi_i$  corresponds to a timing measurement;  $p_i^k$  which is the measured period in the  $k$ th iteration, and  $r_i^k$  denotes the measured response times.  $K$  denotes the total number of samples within the dataset.

1) *Determinism*: The deterministic behavior of a real-time task is associated with its periodicity. This means that the measured period should approximately be a normal distribution with its center equal to the expected period. The standard deviation should also be kept minimal. With these assumptions, the measured period should satisfy the following conditions to be considered deterministic as follows.

- 1) The center,  $\mu$ , should be equal to the scheduled period.
- 2) The distribution should be symmetrical, defined by its skewness and kurtosis.
- 3) If the first two requirements are not met, the sample farthest from the center is omitted, and the analysis is performed for the remaining samples.
- 4) Such omission has a tradeoff with accuracy which is calculated as accuracy =  $1 - n/K$ , where  $n$  denotes the number of omissions from the  $K$  total number of samples.

From these conditions, the distribution of  $p_i$  is analyzed to determine the determinism performance indicator,  $S_D$ . This performance indicator is calculated as the probability that  $p_i$  is inside the interval of a user-defined three-sigma limit  $\hat{\sigma}$  multiplied by the accuracy factor, or

$$S_{D,i} = \text{Prob}(\mu - \hat{\sigma} \leq p_i \leq \mu + \hat{\sigma}) \times \text{accuracy}. \quad (3)$$

This means that the periodicity will have a value between 0 and 1. To ensure that the center of the distribution is approximately equal to the expected period, the statistical mean ( $\mu$ ) is calculated. For flexibility, a user-defined tolerance,  $\alpha$ , is considered to allow a small margin of error, or  $\mu = \mu \pm \alpha$ . To determine if the distribution is symmetric, the skewness,  $\mu_3$ , should be within twice of its standard error. If the skewness is within the allowable range, the standard deviation  $\sigma$  is compared as to whether it is less than  $\hat{\sigma}$ . If so, the analysis ends, and  $S_D$  is equal to the accuracy factor. Otherwise, the kurtosis and its standard of error are calculated as a test of normality [23].

If the distribution of  $p_i$  does not satisfy all these requirements, the dataset is adjusted by omitting the sample farthest from the center. This comes with a tradeoff in accuracy. The process will end with  $S_D = 0$  when all the available samples are omitted. Otherwise,  $S_D$  is calculated as the probability of  $p_i$  in the interval of  $\mu - \hat{\sigma} \leq p_i \leq \mu + \hat{\sigma}$ .

Given the stipulations above,  $p_i$  is determined as a center-focused symmetric distribution which is approximately normal, with the center  $\mu$  approximately equal to the expected period. To determine the probability, the corresponding standard score, or  $z$ -score, is calculated

$$z^+ = \frac{(\mu + \hat{\sigma}) - \mu}{\sigma} = \frac{\hat{\sigma}}{\sigma}, \quad z^- = \frac{(\mu - \hat{\sigma}) - \mu}{\sigma} = -\frac{\hat{\sigma}}{\sigma}. \quad (4)$$

These equations measure the distance from the center depending on the measured standard deviation,  $\sigma$ . To calculate the determinism performance indicator,  $S_D$ , for the  $i$ th real-time task, (3) becomes

$$S_{D,i} = (Z(z^+) - Z(z^-)) \times \text{accuracy}. \quad (5)$$

Herein,  $Z(\cdot)$  converts the  $z$ -scores into standard normal probabilities ( $p$ -values). The corresponding values are found in a  $z$ -score table [24]. The calculated periodicity has an inverse relationship with the standard deviation. It can be concluded that the distribution of the measured period satisfies the standard deviation limits, which eventually determines the deterministic behavior of the real-time task.

**2) Predictability and Timeliness:** The predictability and timeliness of a real-time task greatly rely on the measured response times. For the predictability, the maximum actual response time is compared to the calculated WCRT. Note that both Xenomai and RT-Preempt implements a fixed-priority first-in, first-out scheduler; thus, the WCRT is calculated via RMA [17].

In RMA, the WCRT is affected by several factors, such as the blocking time, calculation time, and interference. In an RTOS, the blocking time should be zero assuming that the real-time tasks do not share any resource locking mechanism.

On the other hand, the interference is affected by the scheduling latencies (e.g., context switching, pre-emption latencies, interrupt latencies).

To account for the effects of these latencies, the predictability of a real-time task is defined as the closeness of the measured WCRT,  $r_i^{\max}$  to the expected value calculated using RMA  $R_i$  while satisfying the stringent deadline  $D_i$ . From this definition, the predictability performance indicator, and  $S_P$  is formulated to acquire values representing the following behavior:

- 1)  $S_P = 1$ , when  $r_i^{\max}$  is equal to the expected WCRT,  $R_i$ .
- 2)  $S_P = 0$ , if  $r_i^{\max}$  either exceeds  $D_i$ , or if  $r_i^{\max} = 0$ .
- 3)  $S_P$  increases as it approaches  $R_i$  and decreases if it approaches either  $D_i$  or 0.

From these conditions,  $S_P$  is calculated using the following piecewise equation, which consists of two variations of linear interpolation, and which resembles a triangular function

$$S_{P,i} = \begin{cases} \frac{r_i^{\max}}{R_i}, & 0 < r_i^{\max} < R_i \\ 1 - \frac{r_i^{\max} - R_i}{D_i - R_i}, & R_i < r_i^{\max} < D_i \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

Despite the difficulty in identifying the exact sources of latencies,  $S_{P,i}$  ensures an accurate interpretation of their effects through (6). On the other hand, we define timeliness of a periodic real-time task as the ability to meet timing constraints during its execution. The timeliness performance indicator,  $S_{T,i}$  is simply calculated as the ratio of samples exceeding the deadlines against the total number of samples. Finally, the RTPI, which describes the deterministic behavior and schedulability of real-time tasks, is determined with the averages of the three performance indicators in (1).

Note that each performance index is multiplied by a user defined scaling factor of 10. Real-time tasks with an RTPI ranging from  $0 \leq \text{RTPI} \leq 3$  can be classified as nonreal-time. This is the case for timing measurements, which shows deterministic (periodic) behavior, at best, with  $S_D = 10$ , but half of all samples miss the deadline ( $S_P = 0$ ,  $S_T = 5$ ). On the other hand, soft or firm real-time systems have RTPI values in the range of  $3 < \text{RTPI} \leq 6.7$ . This occurs when a task misses a single deadline, or  $S_D = 10$ ,  $S_P = 0$ , and  $S_T = 9.99$ . RTPI greater than 6.7 can

be classified as hard real-time. It can be concluded that RTPI can describe both the deterministic behavior and schedulability of real-time tasks by analyzing the periodicity, predictability, and timeliness from actual timing measurements for a single task. The following section explains the strategy to consider the priority of each task in a multitasking environment.

## B. RTPI in a Multitasking Environment

As mentioned in a technical report by Hillary [25], there are no metrics to measure the performance of multitasking applications. For a real-time application with multiple tasks, averaging the RTPIs is not viable due to the different task priorities.

In other words, calculation of the overall RTPI should consider these priorities, and real-time tasks should be weighed depending on their contribution to the overall performance of the entire system. This article, for the first time, provides a solution that weighs task contributions proportional to the number of iterations of a given task within a hyperperiod. A hyperperiod is the smallest interval in time after which the execution pattern of a periodic task is repeated. It is also defined as the least common multiple of the periods of all real-time tasks in a taskset. In this article, the analysis can be restricted into one hyperperiod, and it can be concluded that the same results can be expected for the succeeding hyperperiods.

The real-time tasks should be rate-monotonic, which means that the task with the highest priority should have the highest frequency. In other words, in one hyperperiod ( $P_{\text{hyper}}$ ), the highest priority task will have the greatest number of iterations. Therefore, the number of iterations of the  $i$ th task is calculated as the ratio of the hyperperiod to its period, or  $m_i = P_{\text{hyper}}/P_i$ . Herein, the contribution of each task to the total operation can therefore be calculated, as follows:

$$w_i = \frac{m_i}{\sum_{i=1}^N m_i} \quad (7)$$

Here,  $N$  is the total number of real-time tasks, and  $w_i$  stands for the weight of the  $i$ th real-time task. The final RTPI in a multitasking application is calculated using the following equation:

$$\text{RTPI} = \frac{\sum_{i=1}^N w_i \cdot \text{RTPI}_i}{\sum_{i=1}^N w_i} \quad (8)$$

## V. THEORETICAL ANALYSIS WITH A SIMULATED COMPUTATIONAL LOAD

This section demonstrates the design of an experimental taskset with simulated loads using RT-AIDE. The experiment has been conducted on a quad-core Intel embedded platform, Advantech MIO-5272. It has been installed with the graphics minimal Linux distribution, Ubuntu 18.04. To accommodate the real-time extensions of Xenomai 3.1 and RT-Preempt 4.14.134-rt63, we have chosen the Linux kernel version 4.14.134. The complete details for the kernel configuration and installation of each real-time extension can be found in [26].

The experimental taskset contains four periodic real-time tasks. As both real-time extensions of Linux employ a fixed-priority-based pre-emptive real-time scheduler, the periodic



**TABLE III**  
XENOMAI TIMING MEASUREMENTS (IN MILLISECONDS)

Xenomai	$\tau_1 (P=20, C=5, pr=99)$		$\tau_2 (P=40, C=10, pr=89)$	
	Period	Response	Period	Response
$\mu$	20.000	5.015	40.000	15.006
Max	20.066	5.125	40.110	15.083
Min	19.954	5.004	37.085	15.006
$\sigma$	0.001	0.014	0.001	0.001
Metric	$\tau_3 (P=80, C=10, pr=79)$		$\tau_4 (P=160, C=20, pr=69)$	
	Period	Response	Period	Response
$\mu$	80.000	30.059	160.000	70.000
Max	80.077	30.109	160.019	70.047
Min	79.925	29.063	159.987	69.924
$\sigma$	0.005	0.015	0.005	0.007

**TABLE IV**  
RT-PREEMPT TIMING MEASUREMENTS (IN MILLISECONDS)

RT-Preempt	$\tau_1 (P=20, C=5, pr=99)$		$\tau_2 (P=40, C=10, pr=89)$	
	Period	Response	Period	Response
$\mu$	20.000	5.022	40.000	15.007
Max	20.061	5.105	41.349	15.063
Min	19.942	5.003	39.901	15.005
$\sigma$	0.002	0.019	0.013	0.003
Metric	$\tau_3 (P=80, C=10, pr=79)$		$\tau_4 (P=160, C=20, pr=69)$	
	Period	Response	Period	Response
$\mu$	80.000	31.043	160.000	70.043
Max	81.422	31.092	160.972	71.078
Min	79.929	29.015	159.929	70.015
$\sigma$	0.010	0.052	0.012	0.061

tasks are configured with their own period ( $P$ ), worst-case execution time (WCET) ( $C$ ), and priority ( $pr$ ). In RT-AIDE, the highest priority is 99, while the lowest possible priority for a real-time task is 1. The taskset are described by the following set of tuples with  $P$ ,  $C$ , and  $pr$ :  $\{(20, 5, 99); (40, 10, 89); (80, 10, 79); \text{ and } (160, 20, 69)\}$ . Each task has been designed to burn CPU cycles until the given execution time expires [26].

During the experiment, all timing measurements were stored in a buffer to prevent any overheads that can affect data integrity and all data analyses are conducted offline. The statistical summary is given in Tables III and IV for Xenomai and RT-Preempt, respectively. The experiments were performed for ten minutes. The results were summarized in terms of the average ( $\mu$ ), maximum (Max), minimum (Min), and standard deviation ( $\sigma$ ).

From the actual period, it could easily be identified that RT-Preempt has shown better standard deviation for the highest priority task. On the other hand, Xenomai is superior across all metrics for the remaining tasks. The measured response times has shown that RT-Preempt has produced results closer to the calculated WCRT. As expected, these measurements alone are inconclusive to compare both RTOS and determine the one with the better real-time performance. Tables V and VI show the results of employing the proposed performance metrics and RTPI. To analyze  $S_D$ , the three-sigma limit ( $\hat{\sigma}$ ) and tolerance ( $\alpha$ ) are defined as 0.005 ms and 0.0001 ms, respectively. To better describe the behavior of the system with multiple tasks, the hyperperiod of the taskset has been calculated to be 160 ms. Respectively, the numbers of iterations for each task are 8, 4,

**TABLE V**  
REAL-TIME EVALUATION USING RTPI (XENOMAI)

$\tau_i$	$S_{D_i}$ (Accuracy)	$S_{P_i}$	$S_{T_i}$	RTPI <sub><math>i</math></sub>	$w_i$	RTPI <sub><math>i</math></sub> $\times w_i$
$\tau_1$	10.000 (1)	9.750	10.000	9.917	0.533	5.286
$\tau_2$	10.000 (1)	9.945	10.000	9.982	0.267	2.665
$\tau_3$	6.826 (0.998)	9.964	10.000	8.930	0.133	1.188
$\tau_4$	6.572 (0.963)	9.993	10.000	8.855	0.067	0.593
Final RTPI						9.732

**TABLE VI**  
REAL-TIME EVALUATION USING RTPI (RT-PREEMPT)

$\tau_i$	$S_{D_i}$ (Accuracy)	$S_{P_i}$	$S_{T_i}$	RTPI <sub><math>i</math></sub>	$w_i$	RTPI <sub><math>i</math></sub> $\times w_i$
$\tau_1$	10.000 (1)	9.790	10.000	9.889	0.533	5.271
$\tau_2$	2.043 (1)	9.958	10.000	7.651	0.267	2.043
$\tau_3$	3.783 (0.988)	9.636	10.000	7.806	0.133	1.038
$\tau_4$	3.231 (1)	9.846	10.000	7.692	0.067	0.515
Final RTPI						8.867

2, and 1. From these results, the task weight and final RTPI are determined using (7) and (8).

It can be concluded that both RTOSes have met hard real-time requirements as their final RTPIs are greater than 6.7 as described in Section IV-A. Their difference is since tasks  $\tau_3$ ,  $\tau_3$ , and  $\tau_4$  have lower individual RTPI on RT-Preempt. It can also be observed that for these tasks, the determinism indicator has shown low values of 2.043, 3.771, and 3.171, respectively.

Thus, it can be easily concluded that the periodicity of RT-Preempt produces higher standard deviations than Xenomai, with respect to the user-defined limit and tolerance values. Given these results, RTPI has clearly been able to describe the behavior of the system with multiple real-time tasks. It also has simplified the interpretation of the timing measurements by classifying the results into a single value which is essential to present a straightforward comparison between various RTOSes.

## VI. EXPERIMENTAL VALIDATION

This section covers the design of a real-time application to control an actual network-based motion control system developed with RT-AIDE. The application is deployed on the same real-time environment of Xenomai and RT-Preempt as described in the previous section. Finally, the real-time performance of the real-time extensions are compared using the proposed RTPI.

### A. Network-Based Servo Control System

The experimental testbed consists of four servo drives and ac motors manufactured by LS Mecapion. The servo control system is compliant with the CANopen-over-EtherCAT (CoE) protocol using the CiA 402 profile. The motors are rated to run with a maximum speed of 3000 r/min and come with a built-in 19-bit absolute encoder. All the servo drives are connected in a daisy-chain configuration with Cat7 Ethernet cables.

### B. Real-Time Task Design Using RT-AIDE

For motion control of the four EtherCAT servo drives, real-time tasks are designed using RT-AIDE to handle EtherCAT

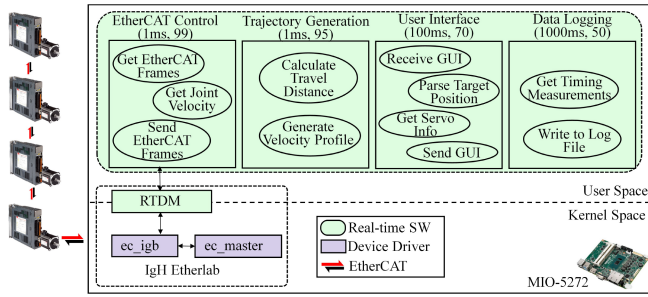


Fig. 3. Real-time controller for an EtherCAT-based servo control system based on RT-AIDE.

communication, trajectory generation, data logging, and communication with a user interface. The real-time tasks are deployed on both Xenomai and RT-Preempt. Four real-time tasks are developed using RT-AIDE, referred to as EtherCAT control, trajectory generation, data logging, and user interface tasks. The real-time application is deployed in the embedded platform as described in the previous section. Fig. 3 shows the task deployment setup of the real-time application designed using RT-AIDE.

The EtherCAT Control task has the highest priority of 99, running at a period of 1 ms. This task is responsible for the communication with the EtherCAT slaves. On the other hand, the Trajectory Generator task runs at the same cyclic period of 1 ms, but with a lower priority of 88. A convolution-based trajectory planning algorithm is utilized considering the maximum velocity, acceleration, and jerk [27].

The user interface task has a priority of 70 and cyclically runs at a period of 100 ms. It receives the position and velocity commands from the EtherCAT Control task and sends these to a user interface through a UDP socket. The user interface is developed using Python. The data Logging task has the lowest priority of 50 and runs at a cyclic period of 1 s. Its responsibility is to acquire the timing measurements of each task and store these values in a local log file.

Due to the dependencies on the IgH EtherCAT, the paths to the IgH EtherCAT API and libraries should be added in CFLAGS and LDFLAGS, respectively. The default installation path of the IgH EtherCAT master is/opt/ethercat. The APIs are in the ethercat/include directory, while the libraries are in ethercat/lib.

### C. Experimental Results

To evaluate the timing measurements with RTPi, the analysis needs to observe the WCET of each real-time task. To obtain the observed WCET, the real-time tasks are deployed individually on each RTOS as a single task application with the highest priority. It can be assumed that the WCET is approximately equal to the WCRT when it has the highest priority on a single application.

It is highly preferable to implement estimation strategies to obtain a statistically sound evaluation of the actual WCET. The choice of this empirical approach of the observed WCET is made as we are more focused on trend analysis and to prove that RTPi can simplify interpretation of timing measurements. For the

TABLE VII  
REAL-TIME EVALUATION USING RTPi (XENOMAI)

$\tau_i$	$S_{D_i}$ (Accuracy)	$S_{P_i}$	$S_{T_i}$	RTPi <sub>i</sub>	$w_i$	RTPi <sub>i</sub> × $w_i$
$\tau_1$	10.000 (1)	9.727	10.000	9.909	0.497	4.927
$\tau_2$	7.271 (1)	9.647	10.000	8.973	0.497	4.462
$\tau_3$	10.000 (1)	9.963	10.000	9.988	0.005	0.050
$\tau_4$	6.864 (0.985)	9.803	10.000	8.889	0.001	0.009
Final RTPi						9.448

TABLE VIII  
REAL-TIME EVALUATION USING RTPi (RT-PREEMPT)

$\tau_i$	$S_{D_i}$ (Accuracy)	$S_{P_i}$	$S_{T_i}$	RTPi <sub>i</sub>	$w_i$	RTPi <sub>i</sub> × $w_i$
$\tau_1$	9.542 (1)	9.975	10.000	9.839	0.497	4.893
$\tau_2$	7.688 (0.999)	7.999	10.000	8.562	0.497	4.258
$\tau_3$	6.614 (0.999)	9.515	10.000	8.710	0.005	0.043
$\tau_4$	1.441 (0.950)	9.355	10.000	6.932	0.001	0.003
Final RTPi						9.197

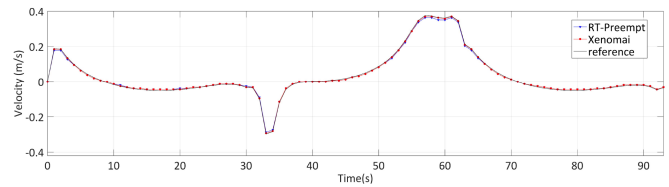


Fig. 4. Reference and feedback velocity profile.

EtherCAT control ( $\tau_1$ ), trajectory generation ( $\tau_2$ ), user interface ( $\tau_3$ ), and data logging ( $\tau_4$ ) tasks, the measured execution times are 0.05, 0.08, 15, and 5 ms, respectively. After calculating for the WCRT using RMA, the timing measurements were evaluated through RTPi with the three-sigma limit ( $\hat{\sigma}$ ) and tolerance ( $\alpha$ ) defined as 0.005 ms and 0.0001 ms for ten minutes.

The final scores in Tables VII and VIII show that both RTOSes can meet hard real-time requirements. Even with a small difference of 0.251, Xenomai shows superior performance in this specific application.

The difference between both RTOSes can also be determined by analyzing each performance metric. In terms of the deterministic behavior, Xenomai has shown superior results with the EtherCAT Control and user interface tasks, while RT-Preempt have better results for the trajectory generation and data logging tasks. For the predictability and timeliness, both RTOSes has shown remarkably similar results. The small difference on the final RTPi can be accounted to the lower value of the predictability indicator of the trajectory generation task in RT-Preempt. This means that the measured WCRT has a significant difference with the one calculated with RMA. To analyze whether this difference has an effect in the actual application, the velocity profile used in the experiments and the actual feedback from each RTOS for a single joint are shown in Fig. 4.

In the figure, it can be easily observed that both RTOSes have managed to track the reference velocity profile, with exceedingly minor differences, as expected. We have also attempted to execute the same application on the standard Linux kernel.



**TABLE IX**  
REAL-TIME EVALUATION OF THE APPLICATION UTILIZING THE INTEROPERABLE COMPONENTS

$\tau_i$	$S_{D_i}$ (Accuracy)	$S_{P_i}$	$S_{T_i}$	RTPI <sub><i>i</i></sub>	$w_i$	RTPI <sub><i>i</i></sub> × $w_i$
$\tau_1$	9.438 (1)	8.258	10.000	9.232	0.497	4.591
$\tau_2$	7.288 (0.987)	4.742	10.000	7.343	0.497	3.649
$\tau_3$	6.178 (0.965)	5.065	10.000	7.081	0.005	0.035
$\tau_4$	6.427 (0.999)	8.664	10.000	8.364	0.001	0.008
Final RTPI						8.283

Due to the fact that the general-purpose kernel is NRT, the real-time constraints of EtherCAT could not be satisfied. The RTPI in this configuration has been calculated to be 2.132. This behavior leads to an unstable connection between the embedded board and EtherCAT slaves, thus neither motion control nor data acquisition has been impossible.

Also, we have followed the same configuration as in [18] to evaluate the real-time performance of the interoperable components discussed in Section III-B. The Trajectory Generation task has been modified to utilize an XDDP mechanism. The task receives velocity commands from a NRT task which subscribes to “/cmd\_vel” topic of the ROS navigation stack [18]. Compilation of this application requires both the Core and ITC headers and libraries.

The same conditions have been followed as in the previous experiments and the results are given in Table IX. We can observe that the RTPI has deteriorated in comparison to the previous real-time application (9.4422). In particular, the determinism performance indicator has shown reduced values for all real-time tasks. This difference can be accounted to the implementation of XDDP. It can be observed from the values of  $S_{P_i}$  that the response times of all real-time tasks are farther from the expected WCRT due to the overhead introduced by the XDDP mechanism in a multitasking environment [18]. Note that the RTPI of 8.283 is still included in the range of hard real-time (see Section IV-A-2). Thus, the interoperable components can guarantee real-time performance of real-time tasks even with an NRT external library (ROS).

From these results it can be concluded that the RTPI has been a useful measure to evaluate RTOSes and their ability to satisfy hard real-time requirements of real-time applications. Aside from the comparison between RTOSes, RTPI can also be used to compare the real-time performance between embedded boards. For example, performance comparison of Xenomai on an Intel PC with various ARM-based embedded systems [19]. Overall, RT-AIDE has been proven to simplify development of real-time applications, guarantee real-time performance of real-time tasks with external NRT libraries, and is also essential to easily compare the performance of various RTOSes using the proposed RTPI.

## VII. LIMITATIONS OF THE STUDY

The empirical results of the experiments reported herein should be considered in the light of some limitations that could be addressed in future research. The first limitation concerns the

results of COCOMO to estimate the development costs and effort to develop real-time applications using RT-AIDE. The results are based on a real-time application running a single periodic task with the available RTMW implementations. Therefore, the comparison focuses on the LOC necessary to create real-time tasks and to manage real-time timers. In case the application is required to execute across various RTOSes, RT-AIDE has shown the lowest development effort since the source code for a specific application should only be developed once, and no further modifications are necessary. However, COCOMO is highly dependent only on LOC to estimate costs and effort, this may underestimate the actual effort related to code complexity, project size, and the learning curve to be familiar with the development environment. It is also worth to note that development of the RTMW implementation for a specific RTOS, in case it is not available, eventually increases development effort.

Other limitation surrounds the proposed RTPI. First, in calculating the deterministic performance indicator, we assume that the measured periodicity of real-time tasks will produce an approximately normal distribution if the sample size is large enough. However, actual results may result to leptokurtic distributions with most of the samples having small deviations from the mean. In this case, the calculated deterministic indicator would be pessimistic probabilities. Although this may be enough to calculate the lower bounds of the real-time performance, calculating the actual probability density function is necessary to achieve more accurate estimations.

Another issue is regarding the calculation of the WCET. In this article, as we are more focused on trend analysis and to prove that RTPI can simplify interpretation of timing measurements, we utilize observed WCET in the experiments. These are measured using software timing probes within the real-time task itself. For more accurate results, it is highly preferable to acquire WCET using probabilistic estimation strategies [28], [29], or by profiling/tracing. Aside from software approaches, using data acquisition systems, or probing digital output pulses are other methods to measure actual WCET.

## VIII. CONCLUSION

This article has presented RT-AIDE, a development environment for real-time systems considering NRT task interoperations and performance evaluations with the aim to reduce the development effort in the design and provide feasible performance evaluations of real-time systems. From a study of the literature and comparison with the state-of-the-art, it was proven that RT-AIDE is the first of its kind.

Theoretical analysis and experimental validation were conducted on an Advantech MIO-5272 with real-time Linux extensions of Xenomai and RT-Preempt. From all the results presented in this article, it was proven that RT-AIDE has successfully met the objectives of an RTOS-agnostic real-time development environment to execute application across various RTOSes, without modifying the source code. Moreover, RTPI, has been proven to simplify analysis of timing measurements, essential in the comparison of the real-time performance of various RTOSes.

There were several directions for future works. First, improving the package management of RT-AIDE by leveraging build process managers was being considered. Also, more testing and improvement of the proposed RTPi was conducted. Hypothesis testing or machine learning models can be employed to consider the actual shape of the measured timing distribution. Calculation of the weights, or a guideline for the developers to determine the weights of the performance indicators was considered. Moreover, strategies to estimate the WCET was proposed [28], [29]. We will study how to utilize them to further optimize the calculation of the response times. We are also considering application of a more complex system, such as multi-axis collaborative robots, to highlight the importance of real-time systems on real-world applications. Finally, RT-AIDE can support different hardware architectures and distributions of RTOS, we will integrate other RTOSes, especially those that implement other scheduling policies, even in multicore processors.

## REFERENCES

- [1] R. Hussain and S. Zeadally, "Autonomous cars: Research results, issues, and future challenges," *IEEE Commun. Surv. Tut.*, vol. 21, no. 2, pp. 1275–1313, Sep. 2018.
- [2] X. Wang, L. Liu, T. Tang, and W. Sun, "Enhancing communication-based train control systems through train-to-train communications," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 4, pp. 1544–1561, Aug. 2018.
- [3] Y. Yang, X. Liu, and R. H. Deng, "Lightweight break-glass access control system for healthcare internet-of-things," *IEEE Trans. Ind. Informat.*, vol. 14, no. 8, pp. 3610–3617, Sep. 2018.
- [4] C. Zhou *et al.*, "Risk-based scheduling of security tasks in industrial control systems with consideration of safety," *IEEE Trans. Ind. Informat.*, vol. 16, no. 5, pp. 3112–3123, Mar. 2019.
- [5] M. N. Omidvar, M. Yang, Y. Mei, X. Li, and X. Yao, "DG2: A faster and more accurate differential grouping for large-scale black-box optimization," *IEEE Trans. Evol. Comput.*, vol. 21, no. 6, pp. 929–942, Dec. 2017.
- [6] Xenomai, Gröbenzell, Xenomai Organization, Bavaria, Germany, 2019. [Online]. Available: <https://source.denx.de/Xenomai/xenomai>
- [7] RT-Preempt, Linux Foundation, San Francisco, CA, USA, 2019. [Online]. Available: <https://rt.wiki.kernel.org/index.php>
- [8] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of vxworks, linux, RTAI, and xenomai in a hard real-time application," *IEEE Trans. Nucl. Sci.*, vol. 55, no. 1, pp. 435–439, Feb. 2008.
- [9] rt-tests, Linux Foundation, San Francisco, CA, USA, 2019. [Online]. Available: <https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/rt-tests>
- [10] M. Hasanloo, M. Kargahi, and S. Jalilian, "Dynamic harvesting-and energy-aware real-time task scheduling," *Sustain. Comput., Inform. Syst.*, vol. 28, Dec. 2020, Art. no. 100413.
- [11] M. Lee, Y. Heo, S. Cho, H. Park, and J.-H. Oh, "Motion generation interface of ROS to PODO software framework for wheeled humanoid robot," in *Proc. 19th Int. Conf. Adv. Robot.*, 2019, pp. 456–461.
- [12] L. Muratore, A. Laurenzi, E. M. Hoffman, A. Rocchi, D. G. Caldwell, and N. G. Tsagarakis, "XBotCore: A real-time cross-robot software platform," in *Proc. 1st IEEE Int. Conf. Robotic Comput.*, 2017, pp. 77–80.
- [13] N. Gobillot, C. Lesire, and D. Doose, "A design and analysis methodology for component-based real-time architectures of autonomous systems," *J. Intell. Robot. Syst.*, vol. 96, no. 1, pp. 123–138, Oct. 2019.
- [14] R. Amaducci, M. Reyes-Sanchez, I. Elices, F. B. Rodriguez, and P. Varona, "RTHybrid: A standardized and open-source real-time software model library for experimental neuroscience," *Front. Neuroinf.*, vol. 13, no. 11, pp. 1–14, Mar. 2019.
- [15] M. Åsberg, T. Nolte, S. Kato, and R. Rajkumar, "ExSched: An external CPU scheduler framework for real-time systems," in *Proc. 18th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2012, pp. 240–249.
- [16] H. Wei *et al.*, "RT-ROS: A real-time ROS architecture on multi-core processors," *Future Gener. Comput. Syst.*, vol. 56, pp. 171–178, Mar. 2016.
- [17] L. Sha, M. H. Klein, and J. B. Goodenough, "Rate monotonic analysis for real-time systems," in *Foundations of Real-Time Computing: Scheduling and Resource Management*. Boston, MA, USA: Springer, 1991, pp. 129–155.
- [18] R. Delgado, B.-J. You, and B. W. Choi, "Real-time control architecture based on xenomai using ROS packages for a service robot," *J. Syst. Softw.*, vol. 151, pp. 8–19, May 2019.
- [19] R. Delgado, J. Park, and B. W. Choi, "Open embedded real-time controllers for industrial distributed control systems," *Electronics*, vol. 8, no. 2, pp. 223, Feb. 2019.
- [20] J. A. Khan, S. U. R. Khan, T. A. Khan, and I. U. R. Khan, "An amplified COCOMO-II based cost estimation model in global software development context," *IEEE Access*, vol. 9, pp. 88602–88620, Jun. 2021.
- [21] M. Pizzoli, C. Forster, and D. Scaramuzza, "REMODE: Probabilistic, monocular dense reconstruction in real time," in *Proc. Int. Conf. Robot. Automat.*, 2014, pp. 2609–2616.
- [22] F. Steinbrucker, J. Sturm, and D. Cremers, "Volumetric 3D mapping in real-time on a CPU," in *Proc. Int. Conf. Robot. Automat.*, 2014, pp. 2021–2028.
- [23] T. Górecki, L. Horváth, and P. Kokoszka, "Tests of normality of functional data," *Int. Stat. Rev.*, vol. 88, no. 3, pp. 677–697, Feb. 2020.
- [24] z-score table, "Z Table," Z-Score Table, Ontario, Canada, 2015. Accessed: Apr. 27, 2022. [Online]. Available: <http://www.z-table.com/>
- [25] N. Hillary, "Measuring performance for real-time systems," 2005. [Online]. Available: <https://www.nxp.com/docs/en/white-paper/CWPERFORMWVP.pdf>
- [26] R. Delgado and B. W. Choi, "New insights into the real-time performance of a multicore processor," *IEEE Access*, vol. 8, pp. 186199–186211, Oct. 2020.
- [27] G. J. Yang, R. Delgado, and B. W. Choi, "A practical joint-space trajectory generation method based on convolution in real-time control," *Int. J. Adv. Robot. Syst.*, vol. 13, no. 2, May 2017, Art. no. 56.
- [28] S. A. B. Shah, M. Rashid, and M. Arif, "Estimating WCET using prediction models to compute fitness function of a genetic algorithm," *Real-Time Syst.*, vol. 56, no. 1, pp. 28–63, Feb. 2020.
- [29] F. Reghenzani, G. Massari, and W. Fornaciari, "Probabilistic-WCET reliability: Statistical testing of EVT hypotheses," *Microprocessors Microsyst.*, vol. 77, pp. 1–12, Sep. 2020.



**Raimarius Delgado** (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical and information engineering from the Seoul National University of Science and Technology, Seoul, South Korea, in 2014 and 2016, and 2021, respectively.

From February 2021, he has been with Koh Young Technology, Yongin, South Korea, where he is currently a Medical Control Software Engineer and a Researcher. His current research interests include real-time systems, medical and service robotics, embedded systems, systems and software architecture, and industrial automation.



**Yong Hwan Jo** (Student Member, IEEE) received the B.S. degree in electrical and information engineering in 2021 from the Seoul National University of Science and Technology, Seoul, South Korea, where he is currently working toward the M.S. degree in electrical and information engineering.

His research interests include real-time systems, industrial control, and automation and embedded systems.



**Byoung Wook Choi** (Member, IEEE) received the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Seoul, South Korea in 1988 and 1992, respectively.

He is currently the Dean of the College of Creativity and Convergence Studies and a Professor with the Department of Electrical and Information Engineering at Seoul National University of Science and Technology. Previously, he was a Principal Research Engineer in 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. Also, he was a Senior Fellow with Nanyang Technological University, Singapore, during 2007–2008. He has published textbooks on Embedded Linux. His current research interests include real-time systems design, embedded systems, and intelligent robot software.