國立臺灣大學電機資訊學院電機工程學系

碩士論文

Department of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master's Thesis

藉由乙太網控制自動化技術的即時運動控制與基於信號量的即時排程

Real-time Motion Control through EtherCAT and Semaphore-based Real-time Scheduling

陳昱彣

Yu-Wen Chen

指導教授: 連豊力 博士

Advisor: Feng-Li Lian, Ph.D.

中華民國　113 年 7 月

July 2024

doi:10.6342/NTU202403297

國立臺灣大學碩士學位論文
口試委員會審定書

MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

藉由乙太網控制自動化技術的即時運動控制與基於信號量的即時排程

Real-time Motion Control through EtherCAT and Semaphore-Based Real-Time Scheduling

本論文係陳昱彣（學號 R11921079）在國立臺灣大學電機工程學系完成之碩士學位論文，於民國一百一十三年七月二十二日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Electrical Engineering on 22 July 2024 have examined a Master's thesis entitled above presented by Yu-Wen Chen (ID R11921079) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

連豊力（指導教授 Advisor）

李後燦

黃正民

江明理

系主任/所長 Director: 李建模

誌謝

在本論文的完成過程中，我得到了許多人的支持與幫助。在此，我謹向所有給予我幫助和支持的人表示誠摯的感謝。

首先，要特別感謝我的指導教授**連豐力**教授，他在整個研究過程中給了我無私的指導和寶貴的建議。在排程的議題上，老師常常用生動有趣的例子來說明，讓人一聽就明白一些抽象的概念，學道將抽象的事物具體說明的方法，在日常的報告和期刊報告裡，也給了許多實用的意見。教授的專業知識和深刻見解不僅幫助我克服了許多困難，也使我在研究領域取得了顯著的進步。我還要感謝我的論文口試委員**李後燦**教授、**黃正民**教授和**江明理**教授，感謝您們提供的寶貴回饋與建議，這些專業意見大大提升了我的研究品質。

謝謝 NCS Lab 的同學們，雖然我的研究方向與大部分同學都不相同，但同學們仍然給予我許多支持。首先，要先謝謝**暄坐**，在進入實驗室前的碩 0 時，花了很長一段時間讓我跟著學習機械手臂的相關技術，雖然最後我的論文主題與機械手臂不相關，但那段時間學習到許多研究的態度與方法。謝謝**芳緯**和**旅青**，常常給我一些實用的意見，可以一起討論一些研究中遇到的困難並互相鼓勵。謝謝**雁丞**幾乎每天進實驗室都看能見到，完全就是研究生表率，也謝謝你提供的 latex 論文模板，讓論文撰寫方便很多。謝謝**芳源**、**昀誠**、**敬祥**、**凱正**，提供一些梗圖、晚上的宵夜、聖誕節的雨傘聖誕樹、元宵節的湯圓..... 為實驗室生活增添樂趣。

此外，還要感謝合作的公司夥伴，在剛接觸這個領域時有非常多東西都不太了解，友通資訊的 **Harris** 和 **Waterball**、Intel 的 **Sherman**、**Sam** 和馬來西亞 **Siew Wen** 的團隊都對本研究的幫助匪淺助，友通提供的硬體設備和 Intel 提供的軟體與技術支持缺一不可，每週的開會討論使的研究進度能走在正確的道路上。另外，要特別感謝 **Harris** 時常能夠與我一起討論與研究技術細節，在下班時間仍撥冗提供視訊討論機會。沒有你們的支持，這項研究無法順利進行。

最後，我要感謝我的朋友**羽彤**，人在英國但仍然和我通訊聊天，陪我排解情緒，也要謝謝家人，感謝你們在我完成這篇論文期間給予的理解和支持。你們的鼓勵和陪伴使我能夠在面對挑戰時保持積極的心態，並順利完成這項研究。

陳昱彣 謹誌

中華民國一百一十三年七月三十一日

doi:10.6342/NTU202403297

藉由乙太網控制自動化技術的即時運動控制與基於信號量的即時排程

研究生: 陳昱彣　　指導教授: 連豊力 博士

國立臺灣大學　電機工程學系

摘要

由於網路系統的複雜性不斷增加，多個任務必須即時協調對共享資源的存取，延遲、同步和資源爭用的挑戰變得至關重要，但仍須保持精準的運動表現。為了應對這些挑戰，該研究透過即時作業系統 Xenomai 將 EtherCAT 與基於訊號量的即時調度方法整合，使用 PID 方法控制網絡上的馬達，提出了一種綜合解決方案進行多軸開源控制。這種方法增強了運動控制任務的反應能力，同時管理多任務環境中的資源分配。實驗與模擬結果表明，單軸控制在 30 rev/s 時的最小延遲為 1 微秒，相對速度誤差為 1.47%。在多軸場景中，即使軸數量增加，系統也能以最小的偏移量保持可預測的同步。在 3 個設備時，基於信號量的調度方法使較低的優先級的設備任務執行次數提高 32%，使多設備時，較高優先級的設備能夠維持良好的資源實用效率，同時較低的優先級的設備相比傳統排程方式也可以獲得不錯的資源分配。這一研究成果為工業自動化和機器人技術提供了新的解決思路，有助於推動相關領域的進一步發展。

關鍵字：

EtherCAT、即時控制系統、即時排程、信號量、PID 控制、軟硬體整合

doi:10.6342/NTU202403297

Real-time Motion Control through EtherCAT and Semaphore-based Real-time Scheduling

Student: Yu-Wen Chen Advisor: Feng-Li Lian, Ph.D.

Department of Electrical Engineering

National Taiwan University

ABSTRACT

As the complexity of networked systems increases, multiple tasks need to coordinate access to shared resources in real time. The challenges of latency, synchronization, and resource contention become critical while still maintaining accurate motion performance. To address these challenges, this research integrates EtherCAT with a semaphore-based real-time scheduling method through the real-time operating system Xenomai, uses the PID control method to control motors on the network, and proposes a comprehensive solution for multi-axis open-source control. This approach enhances responsiveness in motor control tasks while managing resource allocation in a multi-tasking environment. Experimental and simulation results show that the minimum delay of single-axis control is 1 microsecond at 30 rev/s, and the relative speed error is 1.47%. In multi-axis situation, the system maintains predictable synchronization with minimal offset, even as the number of axes increases. When there are 3 devices, the semaphore-based scheduling method increases the number of task executions of lower-priority devices by 32%, so that when there are multiple devices, higher-priority devices can maintain good resource utilization efficiency while lowering the Compared with traditional scheduling methods, devices with higher priority can also obtain better resource allocation. These results provide new solutions for industrial automation and robotics, and helps promote further development in

doi:10.6342/NTU202403297

related fields.

Keywords:

EtherCAT, Real-time system, Real-time scheduling, Semaphore, PID control, Software and hardware integration

doi:10.6342/NTU202403297

doi:10.6342/NTU202403297

CONTENTS

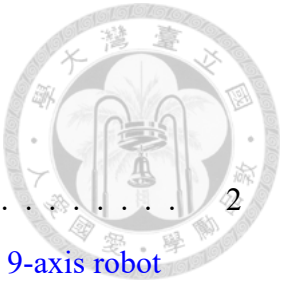
doi:10.6342/NTU202403297

doi:10.6342/NTU202403297

doi:10.6342/NTU202403297

doi:10.6342/NTU202403297

LIST OF FIGURES



doi:10.6342/NTU202403297

doi:10.6342/NTU202403297

LIST OF TABLES

doi:10.6342/NTU202403297

doi:10.6342/NTU202403297

Denotation

T_i	Task model
C_i	Release cycle of T_i
ϕ_i	Phase that is defined as the relative offset of the first start time
R_i	Response time of T_i
J_i	Jitter of T_i
P_i	Fix priority of T_i
τ_i^j	The instance of T_i at the jth period
a_i^j	Ideal starting time of the instance τ_i^j
t_i^j	Actual starting time of τ_i^j
$U(T_i)$	Time utility of T_i
$u(\tau_i^j)$	Time utility of τ_i^j
D_p	Propagation delay
D_t	Transmission delay
$D_{latency}$	Total latency for the EtherCAT network
$D_{sc}(t)$	Time delay induced by network from sensor to controller

doi:10.6342/NTU202403297

$D_{ca}(t)$	Time delay induced by network from controller to actuator
$D_{comm}(t)$	Single time communication delay
$D_{drive}(t)$	Drive-local command delay
$D(n,t)$	Total end to end command delay
t_{drive}	Forwarding time of a message from one drive to another one
$t_{host}(t)$	Forwarding time from host to the first node in the network

doi:10.6342/NTU202403297

Chapter 1

Introduction

In this chapter, the motivation will be shown in Section 1.1. The problem formulation will be discussed in Section 1.2. The contribution and the organization of this thesis are provided in Section 1.3 and Section 1.4.

1.1 Motivation

In modern manufacturing, networked actuator systems play a crucial role in automating and enhancing production processes. These systems consist of multiple actuators connected and controlled through a network, enabling precise and coordinated actions in real-time as shown in Figure 1.1. The real-time concept is vital in networked actuator systems as it ensures that commands and feedback are executed and received with minimal delay. This timeliness is essential for maintaining synchronization, optimizing performance, and ensuring the safety and efficiency of industrial operations. Without real-time capabilities, the potential for delays and errors increases, which can lead to suboptimal production quality and even hazardous situations.

In the system, the choice of communication protocol for network is important to ensure efficient and timely data exchange among system components. One notable protocol that has gained significant traction in recent years is EtherCAT (Ethernet for Control Automation Technology). According to [1: Rostan *et al.* 2010], EtherCAT stands out for its exceptional real-time performance, making it particularly well-suited for demanding industrial automation and control applications.

doi:10.6342/NTU202403297

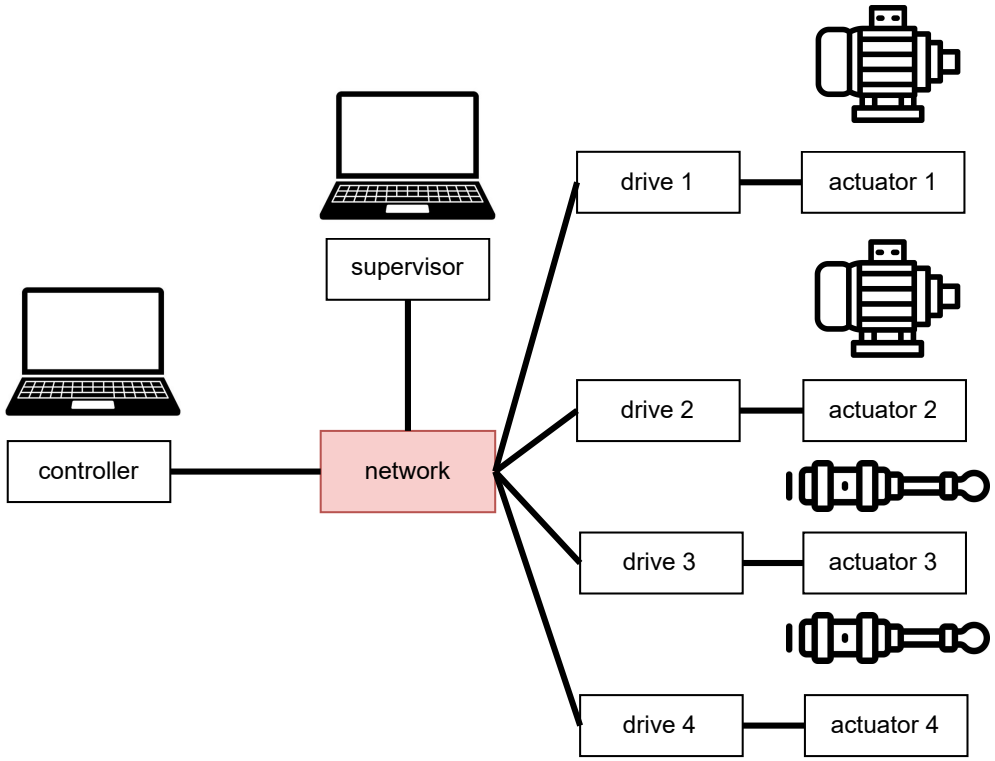
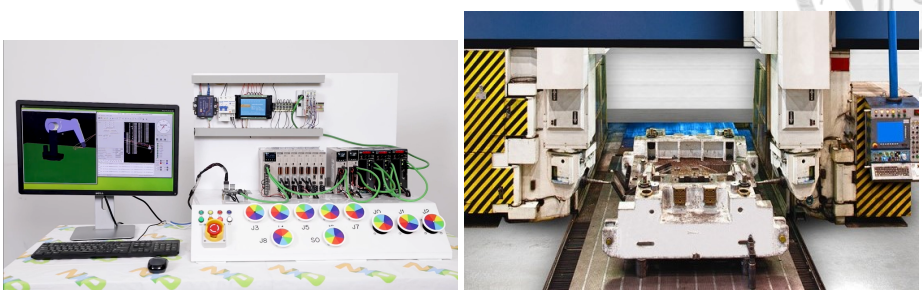


Figure 1.1: The concept of networked actuator system

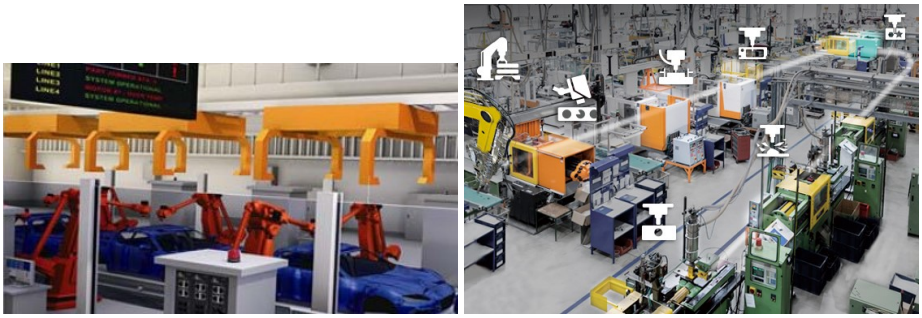
According to [2: Potra and Sebestyen 2006], EtherCAT's real-time capabilities empower engineers to design highly responsive and precise control loops, ensuring rapid feedback and adjustment to change environmental conditions or system disturbances. For instance, in motion control applications as shown in Figure 1.2, EtherCAT's ability to deliver synchronized, high-speed data enables precise coordination of motion axes, leading to smoother trajectories, reduced settling times, and improved accuracy. Because of this advantage, the application of motion control (Figure 1.2(a)) and CNC controller (Figure 1.2(b)) are common. Similarly, EtherCAT facilitates seamless communication between PLCs, HMIs, and I/O modules, enabling coordinated control and monitoring of complex manufacturing processes in distributed control systems for industrial processes. For example, EtherCAT plays a pivotal role in factory Automation (Figure 1.2(c)) and smart factories (Figure 1.2(d)), enabling seamless real-time communication and synchroniza-

doi:10.6342/NTU202403297

tion among various industrial devices, thereby facilitating agile production processes and efficient resource utilization.



(a) Credit to [3: NXP 2022] (b) Credit to [4: NEXCOM 2014]



(c) Credit to [5: Axiomtek 2016] (d) Credit to [6: NEXCOM 2016]

Figure 1.2: Scenarios of the real environment with EtherCAT
(a) 9-axis robot arm control
(b) CNC
(c) Factory Automation
(d) Industry 4.0 IIoT smart factory

Recent years, EtherCAT has been used extensively in the field of robotics. For example, authors developed a real-time EtherCAT Master library in [7: Moon *et al.* 2009], applied EtherCAT technology on servo motor with softPLC and Codesys through Windows in [8: Wang *et al.* 2010] and [9: Zhou *et al.* 2015], applied EtherCAT with real-time operating system in [10: Delgado and Choi 2017], and developed EtherCAT with Robot Operating System(ROS) to control 7-DOF robot in [11: Zhang *et al.* 2019].

The decision to focus on EtherCAT stems from the notable absence of comprehensive research and analysis encompassing both Xenomai and softPLC within existing literature. While numerous studies have explored the applications and performance of EtherCAT in real-time systems, the intersection with Xenomai and softPLC remains largely uncharted

doi:10.6342/NTU202403297

territory. By directing attention towards this overlooked area, this research seeks to bridge the gap in current knowledge and provide insights into the integration of EtherCAT with Xenomai-based real-time systems and softPLC environments. Understanding how EtherCAT can be effectively utilized alongside Xenomai and softPLC holds significant implications for industries reliant on precise and responsive control systems, such as manufacturing, robotics, and automation. By elucidating the synergies and potential challenges inherent in combining these technologies, this study aims to contribute to the advancement of real-time control solutions and inform future developments in the field.

1.2 Problem Formulation

In order to cope with networked systems with a large number of nodes, the concept of traffic lights inspires me. The traffic light system manages traffic flow by indicating when vehicles should stop, get ready, and pass, ensuring that vehicles and pedestrians in each direction can pass through intersections in an orderly manner to avoid traffic jams and accidents. This control method of traffic lights can help us understand how signal volume manages resource access and task scheduling in computer systems.

In real-time systems, multiple tasks may need to access shared resources at the same time, and these resources are often limited. Without an effective management mechanism, contention between tasks will lead to reduced system performance and even deadlock. The semaphore plays a role similar to that of a traffic light, coordinating the access of various tasks to resources and ensuring that the system operates efficiently and orderly. The semaphore controls the use of resources through counters and waiting mechanisms, similar to how traffic lights control traffic flow through color signals.

Figure 1.3 provides an illustration of the formulated problems. It involves key vari-

doi:10.6342/NTU202403297

ables and objectives that are directly impacted by scheduling in networked systems. The primary inputs to consider are command velocity and position, while the outputs are motor velocity and position. A crucial variable in this context is the delay D (latency in signal transmission), and the number of nodes n in the network. Efficient scheduling must account for these inputs and outputs, ensuring that the delay and network congestion do not adversely affect the system's performance.

Effective scheduling algorithms are critical in managing the timing and sequence of tasks across the network nodes, ensuring that command signals (velocity and position) are processed and transmitted to the motors without undue delay. As the number of nodes n increases, the complexity of scheduling also rises, necessitating more sophisticated algorithms to prevent network congestion and priority inversion. This is essential to avoid missed deadlines and ensure that critical data is prioritized, maintaining the system's real-time performance.

In addition to scheduling, in order to achieve real-time control of a mesh system with multiple nodes on a network, the following issues need to be considered. Frist, **integrate EtherCAT Master with Xenomai and SoftPLC**. One of the primary challenges in real-time systems is the seamless integration of EtherCAT masters with Xenomai and softPLC. While EtherCAT offers high-speed and deterministic communication, ensuring compatibility and interoperability with Xenomai and softPLC environments presents significant hurdles. The integration process involves addressing issues such as synchronization between EtherCAT communication cycles and the real-time scheduling mechanisms of Xenomai, as well as ensuring reliable data exchange between the EtherCAT master and softPLC for precise control and monitoring of industrial processes.

doi:10.6342/NTU202403297

Second, **achieve control accuracy with multiple axes**. Another critical problem in real-time systems revolves around achieving control accuracy in applications involving multiple axes of motion or control. Whether it's robotic arms or industrial automation systems, maintaining precise coordination and synchronization across several axes is essential for optimal performance and efficiency. Challenges arise in designing control algorithms, communication protocols, and feedback mechanisms that can handle the complexities of multi-axis control while meeting stringent timing requirements.

Third, **design a platform for remote control**. In this interconnected world, the demand for remote monitoring and control of real-time systems is on the rise. Designing a platform that enables remote access and control of EtherCAT-based systems poses several challenges. These include ensuring secure communication over networks, minimizing latency to maintain real-time responsiveness, and providing a user-friendly interface for remote operators. Addressing these challenges requires careful consideration of system architecture, communication protocols, and cybersecurity measures to deliver a robust and reliable remote control solution for EtherCAT-based real-time systems.

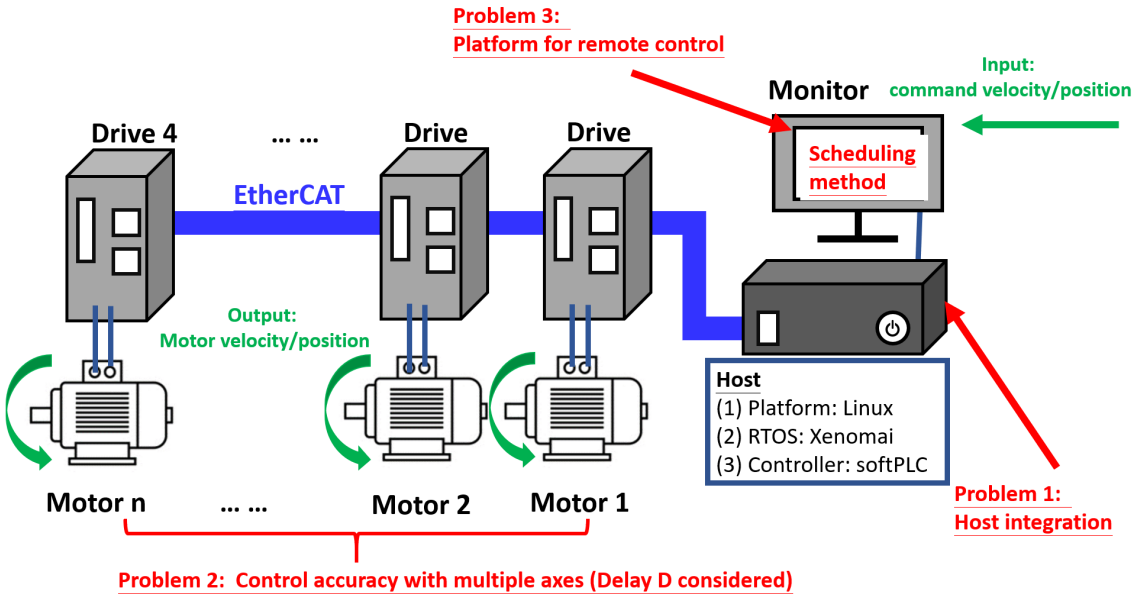


Figure 1.3: Illustration of the formulated problems

doi:10.6342/NTU202403297

1.3 Contribution

The contribution of this thesis focuses on the field of real-time motion control and scheduling within industrial applications by leveraging Intel Edge Controls. The primary contributions are listed belowed.

1. **A novel solution with the combination of Xenomai and openPLC via Intel Edge Control for Industrial**: The thesis provides an open-source and free solution to control motor in real-time situation, which gives a novel solution in industrial automation.

2. **A scheduling method that fits the designed system**: A set of edge-based control algorithms is designed and implemented, taking advantage of the computational capabilities of Intel Edge devices. These algorithms enhance the responsiveness and precision of motion control tasks, crucial for industrial automation.

3. **The great motion performance and real-time perfornance**: With the real-time motion control system platform and scheduling method, great motion and real-time performance is displayed.

doi:10.6342/NTU202403297

1.4 Organization of the Thesis

In Chapter 2, the survey of the real-time system, EtherCAT, and PLC are discussed. In Chapter 3, the system architecture is showed and some models used in this paper are also provided. Chapter 4 contains the details of delat effects and semaphore-based scheduling method. In Chapter 5, the software and hardware integration of system platform is displayed. In Chapter 6, the setups, results and analysis of experiment and simulation are introduced. In Chapter 7, the conclusion and future work are listed.

doi:10.6342/NTU202403297

Chapter 2

Background and Literature Survey

In this chapter, the concept of real-time system will be introduced first, which is the system used in this paper. Then, communication protocol including EtherCAT and the technology, softPLC, used for motor control are discussed. The literature survey is presented at the last.

2.1 Background of Real-time System

Real-time system means that the system must respond correctly within a fixed time and requires strict accuracy and timing. More precisely, real-time is the relationship between event and temporal.

Real-time system can be divided into two main types according to [12: Davis and Burns 2011]:

1. **Hard Real-time**: Tasks must be scheduled and executed such that they always meet their deadlines. This requires deterministic guarantees about the maximum time it takes to complete a task.

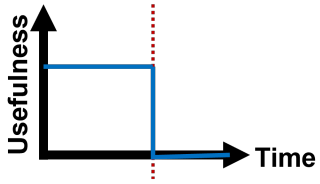
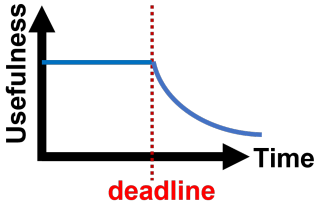
2. **Soft Real-time**: While meeting deadlines is desirable, occasional deadline misses may be tolerated as long as they happen infrequently and within acceptable bounds.

The comparison between soft and hard real-time are listed in Table 2.1. Hard real-time systems prioritize deterministic scheduling for critical tasks, ensuring efficient packing, reliable peak load performance, and stringent safety measures. They excel in main-

doi:10.6342/NTU202403297

taining data integrity and error detection. Soft real-time systems offer more flexibility in task scheduling but may sacrifice deterministic performance and safety guarantees. They prioritize overall system performance over strict timing constraints, potentially leading to less efficient packing and looser error detection mechanisms.

Table 2.1: Comparison of soft real-time and hard real-time

Characteristic	Hard real-time	Soft real-time
Explanation	Responses occur within the required deadline	Deadlines can be occasionally missed
Deadlines	hard-required	soft-required
Packing	environment	computer
Peak load performance	predictable	degraded
Safety	often crirical	non-critical
Data intedrity	short-term	long-tem
Error detection	autonomous	user
Time Utility Curves		

2.1.1 Concept and Common Types of Real-time Scheduling

In order to ensure that tasks are executed within strict deadlines, task scheduling is a common way to fulfill the requirement. According to [13: Liu and Layland 1973], real-time scheduling is a method used in computer systems to ensure that tasks or processes are completed within specific time constraints, known as deadlines. This is crucial in systems where timely execution is essential, such as in embedded systems, industrial control systems, multimedia applications, and many others. There are some common method of scheduling listed below.

1. **Rate-monotonic scheduling (RMS)**:

   According to [14: Lehoczky *et al.* 1989], RMS is a priority-based scheduling

doi:10.6342/NTU202403297

algorithm that assigns priorities to tasks based on their periods. That is, the shorter the period, the higher the priority. RMS assumes periodic tasks with known execution times and periods. The scheduler continuously monitors the system and assigns the CPU to the task with the highest priority (shortest period) among those ready to execute. When a higher-priority task becomes ready, it preempts the currently executing task. The pattern of RMS is shown in Figure 2.1(a).

RMS has several advantages that the algorithm is relatively simple to implement and understand and is optimal for independent periodic tasks with known execution times and periods. When tasks have periods that are integer multiples of each other, RMS can achieve optimal processor utilization. However, RMS also has limitations like its inflexibility,which may not be suitable for systems with a mix of periodic and aperiodic tasks or tasks with varying execution times. Also, if tasks have tight deadlines or if the system becomes overloaded, RMS may lead to missed deadlines, especially if tasks are not schedulable.

2. **Earliest deadline first (EDF)**:

Unlike RMS, EDF schedules tasks based on their absolute deadlines. The task with the earliest absolute deadline is given the highest priority. The pattern of EDF is shown in Figure 2.1(b). According to [15: Short 2010], EDF offers several advantages as its optimality. EDF is optimal for scheduling independent tasks with arbitrary release times, execution times, and deadlines, under the condition that the system is schedulable. It can achieve the highest possible system utilization in such scenarios. EDF is also suitable for systems with both periodic and aperiodic tasks, as well as tasks with varying execution times and deadlines. However, EDF also has limitations. EDF requires maintaining a priority queue of tasks sorted by their

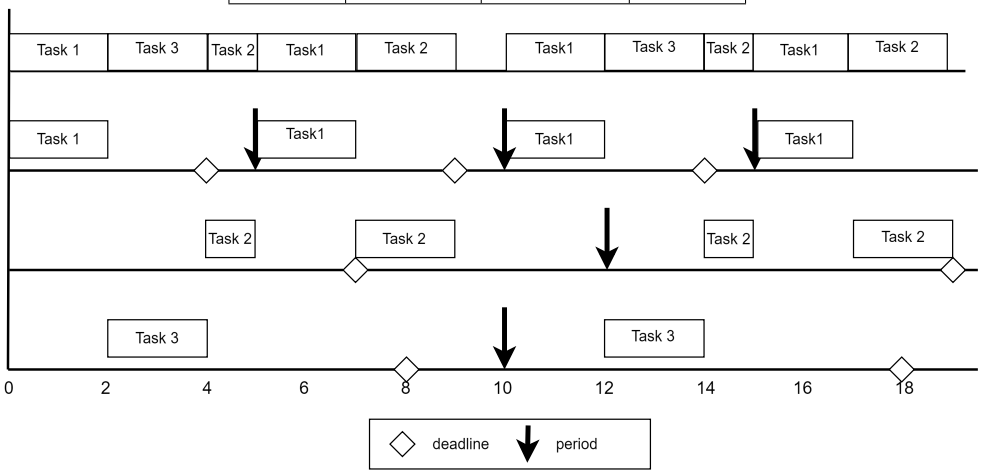
doi:10.6342/NTU202403297

deadlines, which can introduce overhead, especially in systems with a large number of tasks. Moreover, determining whether a set of tasks is schedulable under EDF can be computationally complex. Despite its limitations, EDF remains widely used in real-time systems, particularly in scenarios where tasks have varying execution times and deadlines or when optimality is crucial.

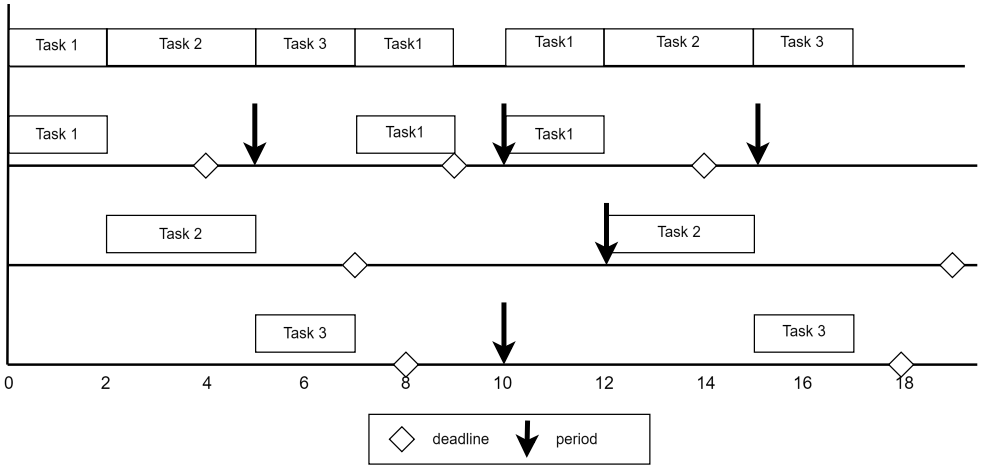
Real-time scheduling is a complex area with many challenges, including ensuring schedulability, managing system resources efficiently, dealing with task dependencies, and handling priority inversion and preemption issues. As a result, designing real-time systems requires careful consideration of these factors to meet the required timing constraints and ensure system reliability.

doi:10.6342/NTU202403297

	Execute	Deadline	Period
Task 1	2	4	5
Task 2	3	7	12
Task 3	3	8	10



(a) Rate monotonic scheduling



(b) Early deadline first

Figure 2.1: The pattern of different scheduling methods

doi:10.6342/NTU202403297

2.1.2 Utilization and Common Types of Real-time Operating System

According to [13: Liu and Layland 1973], a real-time operating system (RTOS) is a specialized software component designed to manage and control the execution of tasks in real-time systems. With this tool, real-time scheduling can be easily conducted. Unlike general-purpose operating systems, which prioritize tasks based on factors such as fairness and efficiency, RTOS prioritizes tasks based on their timing requirements and criticality. One of the defining features of an RTOS is its ability to provide deterministic behavior, ensuring that tasks meet their timing deadlines consistently and predictably.

RTOS typically offers features such as preemptive multitasking, where tasks are preempted based on their priority levels, allowing higher-priority tasks to interrupt lower-priority ones when necessary. Additionally, RTOS often provide mechanisms for task scheduling, inter-task communication, synchronization, and resource management, all optimized for real-time performance according to [16: Burns 1991]. These features enable developers to design and implement complex control algorithms, sensor data processing, and communication protocols while guaranteeing timely and deterministic execution.

There are some common RTOSs from [17: Barbalace *et al.* 2007] listed below.

1. **PREEMT-RT**:

   Preemption Real-Time patch, which is abbreviated as PREEMPT-RT, is an enhancement to the Linux kernel that aims to bring real-time capabilities to standard Linux distributions. According to [18: Reghenzani *et al.* 2019], it introduces full preemption support to the Linux kernel, allowing it to respond quickly to external events and meet real-time requirements. PREEMPT-RT modifies the Linux kernel itself to introduce full preemption support, making it more tightly integrated with

doi:10.6342/NTU202403297

the mainline Linux development process.The structure is shown in Figure 2.2(a). PREEMPT-RT primarily focuses on reducing interrupt latency and improving kernel responsiveness through full preemption support. It doesn't enforce specific real-time scheduling policies but provides a foundation for implementing them.

2. **RTAI**:

   RTAI is an open-source software developed by the Department of Aerospace Engineering, Politecnico di Milano (DIAPM), which follows the GNU General Public License (GPL). According to its homepage from [19: RTAI home page], it provides the hard real-time and retains the services provided by Linux. RTAI uses ADEOS as the hardware architecture layer (HAL) under the Linux kernel, and implements initialization, interrupt application through ADEOS. The structure is displayed in Figure 2.2(b).

   RTAI as the real-time kernel has a higher priority than Linux kernel. When an interrupt occurs, ADEOS first calls RTAI to process the interrupt and executes the real-time task. Only when there is no real-time task or interrupt that needs to be processed, ADEOS will permissions be given to Linux.

3. **Xenomai**:

   Xenomai uses ADEOS as the middle HAL layer like RTAI. The difference is that Xenomai's hardware layer is completely separated by ADEOS. RTAI is committed to the lowest technically feasible latency; Xenomai also attaches great importance to extended functionalilability.

   Xenomai supports various real-time scheduling algorithms, including fixed priority scheduling (Cobalt) and earliest deadline first (Mercury) according to its home-

doi:10.6342/NTU202403297

page from [20: Xenomai home page]. Developers can choose the scheduling algorithm that best suits their application's requirements. The main advantage is that Xenomai employs a dual-kernel approach, consisting of a primary Linux kernel and a secondary real-time kernel, known as the Xenomai nucleus. The Xenomai nucleus runs alongside the Linux kernel, and it is responsible for managing real-time tasks with precise timing requirements.The structure is displayed in Figure 2.2(c).

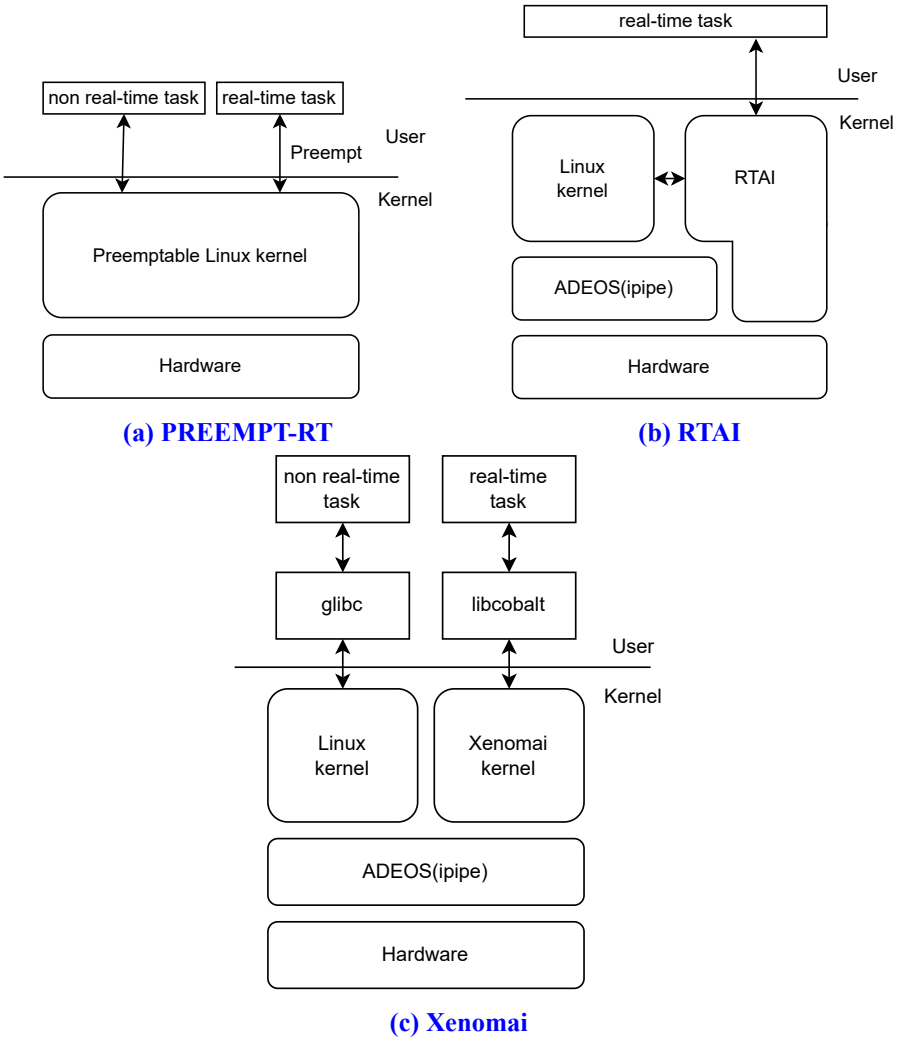


Figure 2.2: Architecture of different RTOSs

The paper [17: Barbalace *et al.* 2007] mainly evaluates the performance of interrupt delay, rescheduling and network communication. The experimental results are shown in Table 2.2. We found that the performance of RTAI is better than that of Xenomai.

doi:10.6342/NTU202403297

The reason is attributed to the processing method of ADEOS. RTAI directly intercepts unnecessary interruptions and then passes them through ADEOS, while Xenomai passes through ADEOS every time, so there will be some additional latency.

Table 2.2: Comparison of RTOSs from paper [17: Barbalace *et al.* 2007]

Measured delays with real-time network communication			
	Linux	RTAI	Xenomai
jitter(s)	11.1	3.0	3.3
delay(s)	113	101	104.5
Measured delay and jitter including rescheduling			
	Linux	RTAI	Xenomai
jitter(s)	1.5	0.4	0.45
rescheduling(s)	5.60	0.20	2.70
delay(s)	72.8	71.8	73.2

Xenomai, RTAI, and PREEMPT-RT are all powerful tools for real-time development on Linux-based systems, each offering unique features and approaches to achieve real-time capabilities. RTAI works the greatest real-time performance; Xenomai supports various real-time scheduling algorithms with the highest flexibility; PREEMPT-RT maintains compatibility with existing Linux software and applications, allowing real-time tasks to coexist with standard Linux processes seamlessly. It doesn't impose a specific API for real-time application development. The choice between them depends on each requirements.

2.2 Background of Communication Protocol: EtherCAT

EtherCAT is a kind of industrial control communication technology based on Ethernet which was proposed by Beckhoff Company. EtherCAT realizes high-performance, low-latency, and high-precision data exchange according to [21: EtherCAT technology group]. Therefore, EtherCAT is widely used in automation control systems, such as robots, machine tools, automated production lines, and factory automation. In 2006, EtherLAB

doi:10.6342/NTU202403297

proposed an open-source software, IgH EtherCAT Master, as the main tool to develop related technology.

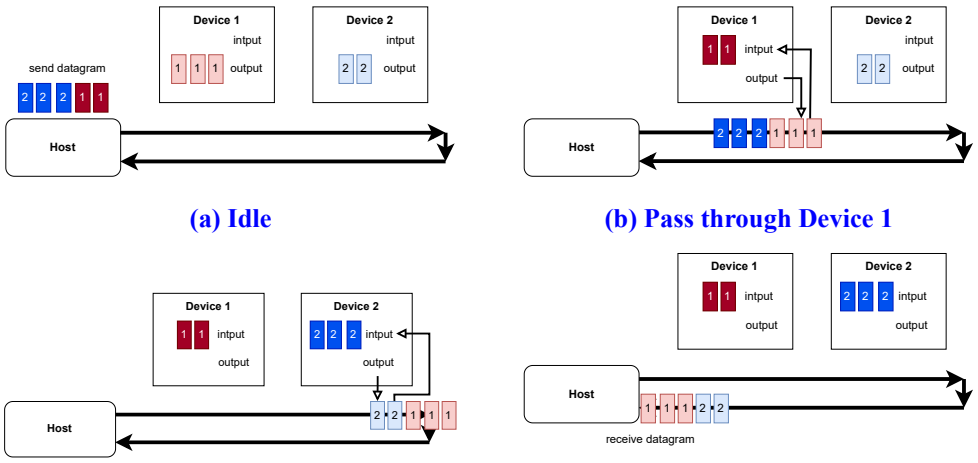
EtherCAT offers significant advantages in industrial automation and real-time control systems due to its high-speed communication capabilities, enabling extremely low latency and jitter essential for precise and synchronized operations. Its ability to process data on-the-fly drastically reduces cycle times, making it suitable for applications requiring high responsiveness. EtherCAT's scalability and flexible topologies allow easy expansion and adaptation to various system layouts, supporting many nodes without compromising performance. Efficient bandwidth usage maximizes data throughput, enabling the integration of more devices and sensors, enhancing overall system capabilities. Additionally, its compatibility with a broad array of standard and vendor-specific protocols ensures interoperability with diverse devices and equipment, promoting a versatile and cost-effective approach to automation system development and maintenance.

2.2.1 Functional Principle of EtherCAT

EtherCAT architecture operates on a host/device basis. The main feature of EtherCAT is the "on the fly" technology which improves network transmission delays and asynchronization problems according to [2: Potra and Sebestyen 2006]. Host transmits data packets that comply with the IEEE 802.3 standard and passes through all nodes in the network architecture. Therefore, the device side can obtain the required data when the packet arrives and put the data which needs to be transmitted into the frame as shown in Figure 2.3. Thus, the feature, on the fly, will only be affected by the operating delay of the hardware. Since the Ethernet network has the characteristics of full-duplex, the message will eventually be sent back to the master. Therefore, according to [22: Nguyen and Jeon 2016], the transmitting data rate can exceed 100 Mbps, allowing EtherCAT to become a

doi:10.6342/NTU202403297

high-performance distributed I/O system.



(a) Idle (b) Pass through Device 1

(c) Pass through Device 2 (d) Return to Master

Figure 2.3: The concept of on the fly mechanism

2.2.2 Synchronization Mechanism: Distributed Clock

EtherCAT achieves precise synchronization through the mechanism of distributed clock (DC). Since the Ethernet physical layer has a full-duplex and ring structure, nodes can use the timestamp to measure the difference of the time between the packet leaving and the return when the EtherCAT packet passes through. Therefore, according to [23: Shen *et al.* 2020], by setting the first one of all nodes as the reference clock (Clock 0), the difference of the time referred to Clock 0 on other nodes can be calculated during each cycle. The host can calculate the propagation offset of each independent nodes with a simple and effective way. Distributed clock is adjusted according to the offset, which can also reduce the jitter and meet the requirement of precise synchronization control with the delay less than 1 μs, and support multi-axis synchronous motion control.

doi:10.6342/NTU202403297

2.2.3 Format of EtherCAT Protocol

EtherCAT communication protocol is optimized for data processing. The master can communicate with nodes via broadcast. As shown in Table 2.3, an EtherCAT frame consists of Ethernet header, EtherCAT telegram and frame check sequence. The detailed items are provided in Table 2.3(a). EtherCAT frame starts with a standard Ethernet header, including fields the source and destination MAC addresses, and is followed by EtherType field, which use standard IEEE 802.3 Ethernet frame transmission. The Ethertype is 0x88a4.

Then, the information about EtherCAT is contained in EtherCAT telegram as shown in Table 2.3(b). An EtherCAT frame can consist of several datagrams, each of which serves a specific memory area of a logical addressing. This area can be up to 4GB bytes. The EtherCAT header follows the Ethernet header . This header contains crucial information for EtherCAT processing,which is divided into a length specification, a reserved bit 0 and a specification for the protocol type.

According to [21: EtherCAT technology group], the part of EtherCAT datagram is shown in Table 2.3(c). If there are n nodes in the EtherCAT network, EtherCAT datagram will contain n datagrams in each frame. Datagram header indicates what type of the master is and what would like to execute such as read, write, read-write. Logical addressing is used for the exchange of process data. Each datagram addresses a specific part of the process image in the EtherCAT segment. During network working, each node is assigned one or more addresses in this global address space. Since datagrams contain all the data related information, the host can decide when and which data to access.

doi:10.6342/NTU202403297

Table 2.3: Format of EtherCAT protocol frame

-	Ethernet header			EtherCAT telegram	FCS
Bytes	14			Table 2.3(b)	4
Item	Destination or target adress (DA)	Source address (SA)	Type		-
Bytes	6	6	2		

(a) Overall datagram

-	ECAT header			EtherCAT datagram	Padding bytes
Bytes	2			A	0∼32
Item	Length of A	Reserved	Protocol type	Table 2.3(c)	-
Bits	11	1	4		

(b) EtherCAT telegram

-	Datagram 1									...	Datagram n
Bytes	10+n+2									...	10+k+2
Item	Datagram header									Data	WKC	as Datagram 1
Bytes	10									n	2	
Item	CMD	IDX	Address	Length of n	R	C	R	M	IRQ	-		
Bits	8	8	32	11	2	1	1	1	16			

(c) EtherCAT datagram

2.2.4 EtherCAT State Machine

EtherCAT State Machine (ESM) is responsible for coordinating the status relationship and transition between the host station and the device station application during initialization and running. From the initialization state to the running state, the process of " Initialization -> Pre-operation -> Safe Operation -> Operation" must be followed sequential transformation. Skip-level transition is possible when returning from the running state. According to [21: EtherCAT technology group], ESM is shown in Figure 2.4.

1. **INIT state**: The device is in the INIT state when it is powered on. The host can read information from devices and prepare to enter the pre-op initialization like setting the node address, mailbox information, DC transmission delay and offset for synchronization by Sync Manager. After devices check right, the host can request to enter the pre- op state.

doi:10.6342/NTU202403297

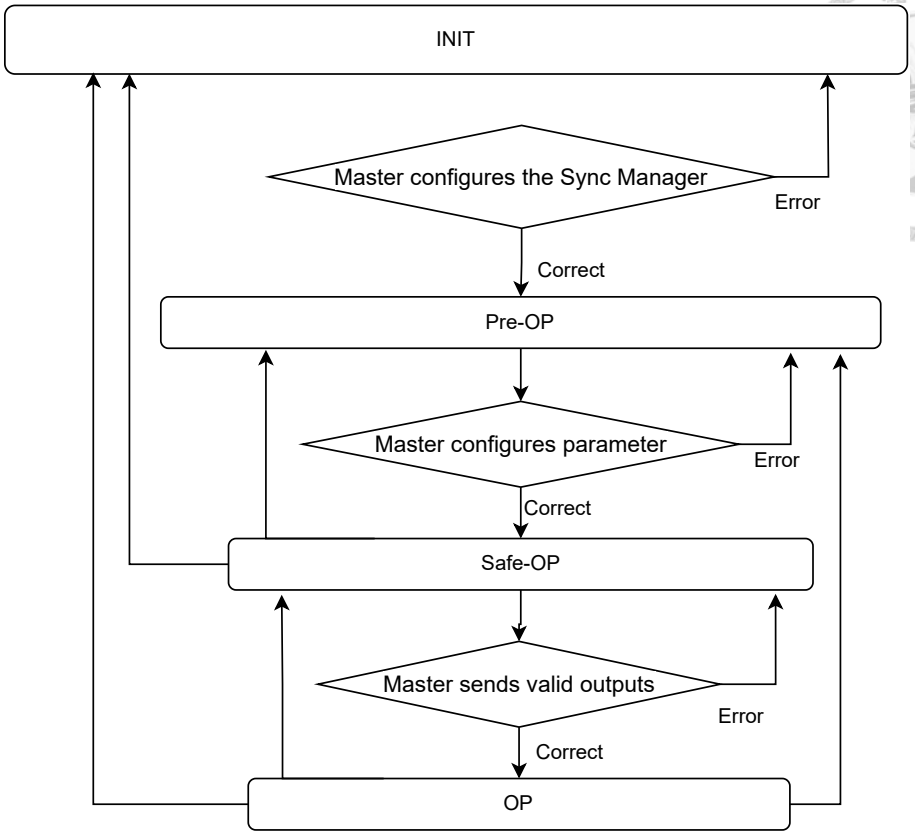


Figure 2.4: EtherCAT state machine diagram

2. **Pre-op state**: The host sets the processing data object (PDO) that the device need to map, and configures the mapping in this state. Also, the master will set the DC cycle time, startup time, trigger mode, and start synchronizing the DC clock. After setting the above information and checking right, the host requests to enter the safe-op state.

3. **Safe-op state**: In safe-op state, the host mainly sets the data in the PDO to the nodes, confirms whether the device will report an error, and tests the DC synchronization of nodes to reach a stable value. If the device does not report an error, it will request to enter the OP state.

4. **OP state**: After entering the OP state, system can perform motion control through EtherCAT.

doi:10.6342/NTU202403297

2.3 Background of Controller: SoftPLC

In 1969, American Digital Equipment Corporation (DEC) developed the first industrial programmable controller (PLC) to solve the problem about the maintenance and modification on traditional relay. PLC was then widely used in industrial production processes. According to [24: John and Tiegelkamp 2010] in 1982, International Electro-technical Commission (IEC) promulgated the PLC standard after many revisions, and PLC international standard IEC61131 was formulated . With the promotion of IEC61131 standard, a new technology SoftPLC had been developed. According to [25: Song *et al.* 2007], it is a software running on industrial computers which follows the IEC61131 standard about PLC. SoftPLC uses software to replace traditional hardware PLC controller and has an open architecture, supports control algorithms, such as PID control, etc.

To make programming easier, Mário proposed MATPLC, which follows IEC 61131-3 and utilizes five PLC languages into C programming language. MATPLC can also compile it into a programmable program through Linux's GCC and G++ compilation software according to [26: de Sousa and Carvalho 2003]. In 2007, Edouard developed an open-source integrated development environment named Beremiz, which provides a PLC program editor function and can also execute PLC, complying with the five PLC standards of IEC61131-3. language from [27: Tisserant *et al.* 2007]. For hardware real-time multitasking, authors proposed an effective reproducible debugging solution by leveraging the special properties of these programs from [28: Prahofer *et al.* 2011]. This technology enables recording a minimum amount of data and adhere to real-time constraints.

doi:10.6342/NTU202403297

2.3.1 Motion Control Library: PLCopen Motion Control

IEC has organized and formulated the IEC61131-3 protocol, which is the standard language of the international PLCopen architecture. According to [29: Eldijk 2018], the IEC61131-3 protocol includes five languages, ladder diagram (LD), function block diagram(FBD), structure text (ST), instruction list (IL), and sequential function chart (SFC). While promoting the IEC 61131-3 standard, PLCopen introduced motion control technology into the research and development of IEC. environment and developed a set of standard motion control function blocks. According to [30: Lin *et al.* 2018], PLCopen adopts the IEC 61131-3 standard function blocks programming language as the motion control programming language.

PLCopen motion control can perform corresponding single-axis, multi-axis and axis positioning, fixed speed and other functions based on the motion parameters input by the user, including displacement, speed, acceleration, deceleration and jerk. The operation of PLCopen motion control is based on the axis state machine. These state machines include: discrete motion state, continuous motion state, stopping state, homing state, synchronized Motion state, errorStop state, standStill state and disabled state. The PLCopen state machine diagram is shown in Figure 2.5.

Under the architecture of distributed motion control, software and hardware can be flexibly modified. PLCopen Motion Control facilitates seamless integration with other automation systems and devices, enabling comprehensive control and synchronization of motion tasks within larger production processes.

doi:10.6342/NTU202403297

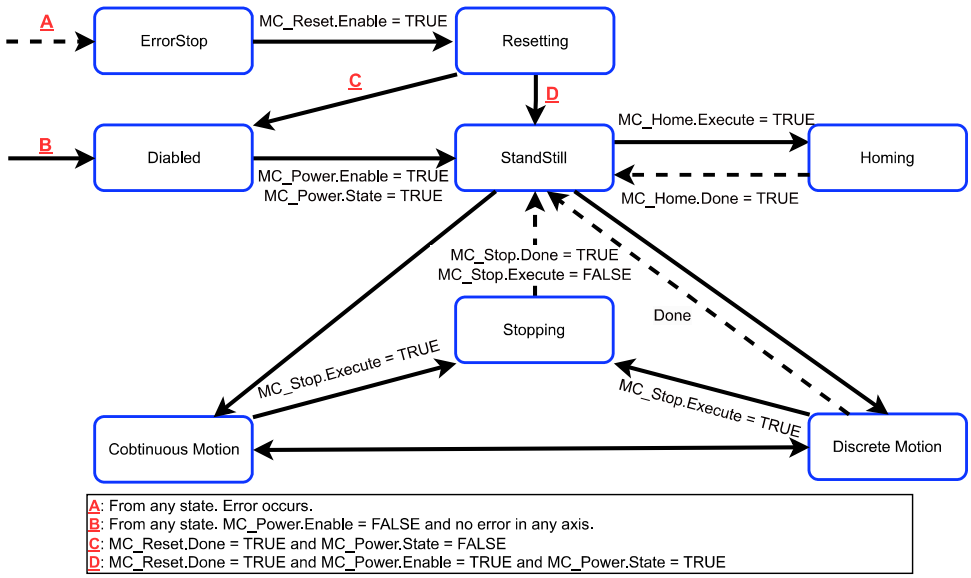


Figure 2.5: PLCopen motion control state machine diagram

2.4 Literature Survey

This section provides an extensive literature survey on three critical areas in real-time systems: the application of semaphores in scheduling, the evolution and comparison of communication protocols, and the integration of robotics with real-time systems.

2.4.1 Application of Semaphore in Scheduling

In real-time systems, the efficiency and reliability of task scheduling are crucial to the overall performance of the system. As a classic synchronization mechanism, semaphore is widely used to manage resource access of concurrent tasks.

According to [31: Downey 2009], semaphores were introduced in 1965. Semaphores control access to shared resources through an integer counter. When the resource is occupied, the counter decreases. On the other hand, when the resource is released, the counter increases. The task checks the value of the semaphore before accessing the resource. If the counter is greater than zero, access is allowed. Otherwise, the task enters the waiting

doi:10.6342/NTU202403297

state.

Many studies have shown that semaphore-based scheduling algorithms perform well in managing concurrent access to real-time tasks. For example, according to [32: Andersson and Tovar 2006], Andersson and Tovar explored the use of semaphores in a multiprocessor environment in their study. They proposed a global scheduling algorithm based on semaphores, which can effectively reduce the number of task context switches and improve the throughput of the system. Another study from [33: Sha *et al.* 1990] analyzed the application of semaphores in real-time operating systems and proposed an improved priority inheritance mechanism to avoid priority inversion through semaphores, thereby improving the response speed and reliability of the system.

In the fields of industrial control and automation, semaphore-based scheduling algorithms also have important applications. Instant communication protocols require precise timing control to ensure low-latency transmission and efficient processing of data.

2.4.2 Evolution and Comparison of Communication Protocol

The evolution of communication protocols has been driven by the need for faster, more reliable, and more efficient data transfer methods in increasingly networked systems. In the early days of digital communication, according to [34: EIA 2010], protocols like RS-232 were sufficient for simple point-to-point connections. However, as industrial automation and networking demands grew, more sophisticated protocols emerged. According to [35: Organization], the advent of fieldbus technologies in the 1980s, such as Modbus, allowed for better handling of data in industrial environments, providing deterministic and robust communication. In the 1990s and 2000s, according to [36: Alliance], Ethernet-based protocols began to dominate due to their high speed and wide accep-

doi:10.6342/NTU202403297

tance. This period saw the rise of protocols like EtherNet/IP, Profinet, and EtherCAT, which leveraged the ubiquity and performance of Ethernet while adding features critical for industrial applications.

Table 2.4 displays the comparison of different protocols. EtherCAT stands out among communication protocols for its high performance and efficiency, particularly in real-time industrial automation environments. Unlike traditional Ethernet, which handles data packets in a store-and-forward manner, EtherCAT processes frames on-the-fly as they pass through each node. This method significantly reduces latency and improves synchronization, making it ideal for applications requiring precise control and fast response times. In contrast, protocols like Profibus and Modbus, although reliable and widely used, offer lower data rates and higher latency, which can be limiting for modern, high-speed applications. EtherNet/IP, while offering good interoperability and scalability, tends to have higher overheads compared to EtherCAT. Each of these protocols has its strengths and ideal use cases, but EtherCAT's unique approach to data handling and its superior real-time performance give it a distinct advantage in many demanding industrial settings.

Table 2.4: Comparison of different protocols

Feature	EtherCAT	EtherNet/IP	Profibus	Modbus
Data Handling	On-the-fly processing	Cyclic and acyclic	Cyclic	Cyclic
Latency	Very low	Moderate	Moderate to high	Moderate to high
Real-time Performance	Excellent	Good	Fair	Fair
Bandwidth	Up to 100 Mbps	Up to 100 Mbps	Up to 12 Mbps	Up to 1 Mbps
Determinism	High	Moderate	High	Moderate
Topology	Flexible (line, star, tree)	Flexible	Line, star	Line, star
Synchronization	Distributed clocks	CIP Sync	None	None

doi:10.6342/NTU202403297

2.4.3 Robotics with Real-time System

The application of robotics with EtherCAT technology revolutionizes industrial automation by providing high-speed, deterministic communication and synchronization capabilities essential for precise robotic control. With EtherCAT's real-time communication, robots can receive instantaneous commands and feedback, facilitating agile and adaptive motion control for tasks. Additionally, EtherCAT's distributed clock synchronization ensures accurate timing and coordination among multiple robotic axes, enabling synchronized motion and collaborative operation in complex production scenarios.

EtherCAT have been used extensively in the field of robotics. In [7: Moon *et al.* 2009], the authors developed a real-time EtherCAT Master library, which can supported application programming interfaces(APIs) for programming of real-time application to control EtherCAT network. In [8: Wang *et al.* 2010] and [9: Zhou *et al.* 2015], the authors applied EtherCAT technology on servo motor with softPLC and Codesys as the development environment respectively. However, both of them utilized the platform based on Mindows provided by Beckhoff company. In [10: Delgado and Choi 2017], the authors applied EtherCAT with real-time operating system, Xenomai and PREEMT-RT and conclued that Xenomai has more accurate cyclic task periodicity. And also, they applied EtherCAT with Xenomai on mibile robot in [37: Delgado *et al.* 2016]. With the trajectory planning method proposed, mobile robot can work in high accuracy and performance.

On the other hand, the authors utilized EtherCAT with Codesys to improve the synchronicity between the network clock and the local motor control task on multi-axis, which get a better real-time performance on the control of multi motors in [23: Shen *et al.* 2020]. However, the application can only utilize on the certain platform. The authors developed

doi:10.6342/NTU202403297

EtherCAT with Robot Operating System(ROS) to control 7-DOF robot and they did some experiment to show the high real-time performance of this control system in [11: Zhang *et al.* 2019].

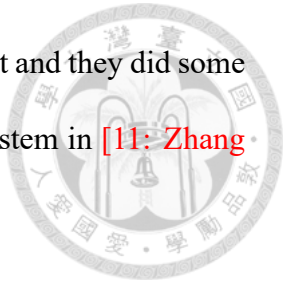


Table 2.5: The category of real-rime implementation from recent works

	SoftPLC	ROS	CodeSys
Non real-time	[8: Wang *et al.* 2010]		**chang-chengDesignServoDrive2015** [23: Shen *et al.* 2020]
Linux		[11: Zhang *et al.* 2019]	
Xenomai		[38: Zhou *et al.* 2019]	[37: Delgado *et al.* 2016]

doi:10.6342/NTU202403297

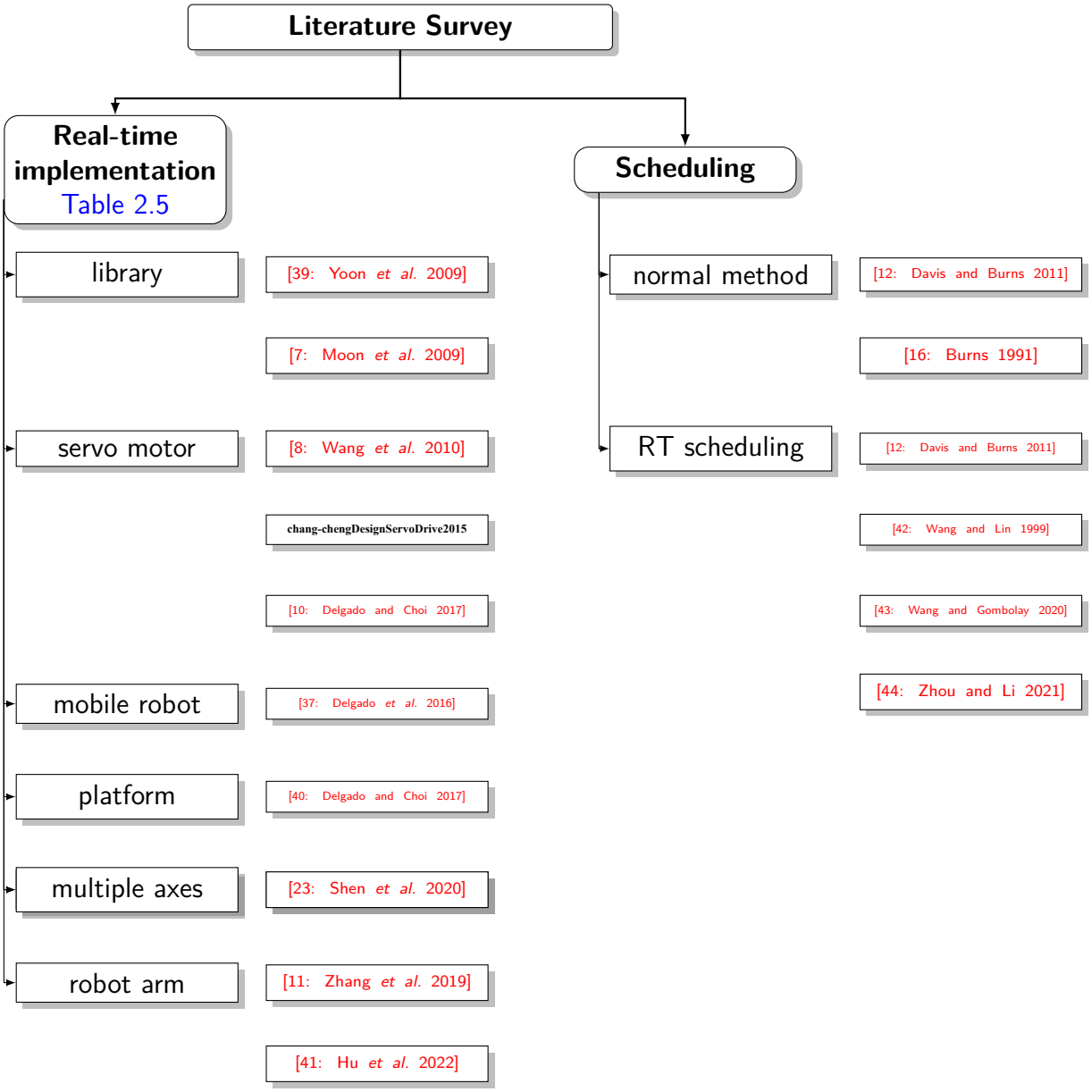


Figure 2.6: Key survey of this theis.

doi:10.6342/NTU202403297

Chapter 3

System Overview

3.1 System Architecture

The overall system consists of structural part and scheduling part. As shown in Figure 3.1, the system is composed of a control host and EtherCAT networks with drives and motors. In the control host, the real-time motion control system based on Intel Edge Controls for industrial applications offers a robust framework for precise and efficient control, and the structural architecture is shown in Figure 3.2. A modular architecture is designed for scalability and flexibility. At its core, the system integrates Intel's cutting-edge hardware and software capabilities to enable real-time processing and decision-making at the edge, minimizing latency and enhancing responsiveness. The architecture can be separated into communication part and motion comtrol part,which are running as the real-time task. In the software layer, EtherCAT master is conducued to create the network communication and the openPLC rumtime is utilized as the motion controller in the user layer part. Then, a supervisory control and data acquisition (SCADA) software is built for monitoring.

doi:10.6342/NTU202403297

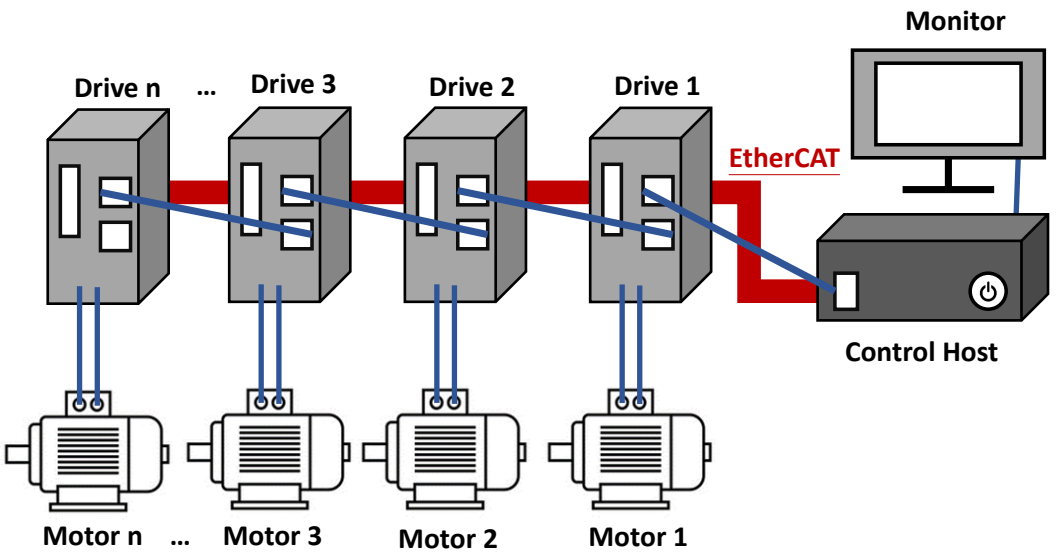


Figure 3.1: Schematic diagram of the system

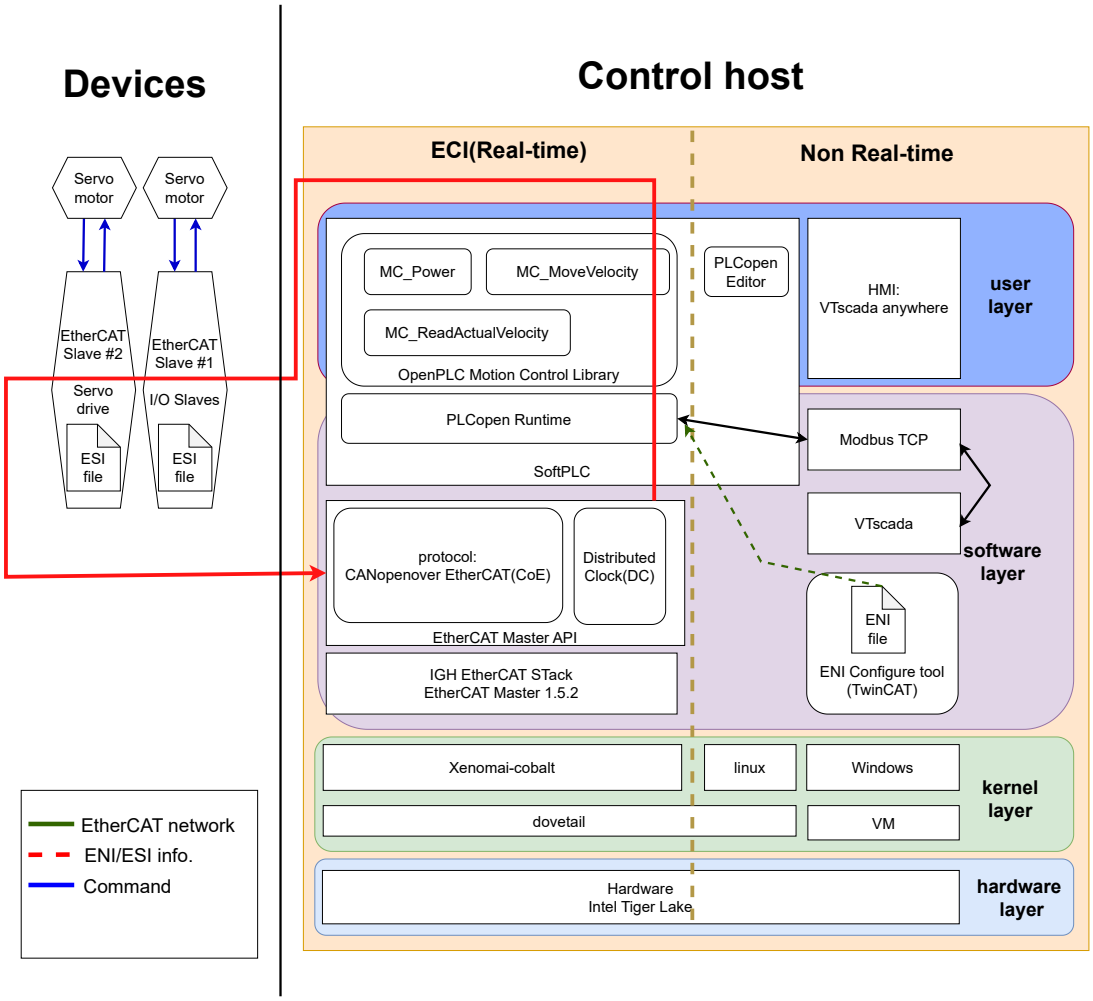


Figure 3.2: The system architecture

doi:10.6342/NTU202403297

3.2 System Model

3.2.1 Real-time System Model

In this section system models are described. Assuming the adoption of a fixed priority scheduling algorithm is RMS, the task model T_i can be characterized by Equation (3.1).

$$T_i = (C_i, \phi_i, R_i, J_i, P_i) \tag{3.1}$$

C_i is the release cycle of T_i; ϕ_i represents the phase that is defined as the relative offset of the first start time; R_i is the response time; J_i is the jitter; P_i is the fixed priority.

Then, define the instance of T_i at the jth period is represented by τ_i^j. There are two specific time are marked as, a_i^j, and t_i^j. a_i^j represents the ideal starting time of the instance τ_i^j, which fit Equation (3.2). t_i^j is the actual starting time of τ_i^j. Beacause of the existence of jitter, the actual starting time will be delayed. The relation is shown in Equation (3.3).

$$a_i^{j+1} = a_i^j + C_i \tag{3.2}$$

$$t_i^j = a_i^j + J_i \tag{3.3}$$

To define the deadline, (m,k)-firm dealine is used in this thesis. In real-time systems, (m,k)-firm deadlines represent a critical scheduling concept where tasks must meet a certain level of reliability and timing guarantees according to [45: Hamdaoui and Ramanathan 1995]. It must complete successfully at least m out of the last k instances within a specified deadline. This model accommodates occasional missed deadlines due to system uncertainties or transient faults while maintaining overall system integrity and performance. In this thesis, the deadline is assumed to be eual to the period for each task.

doi:10.6342/NTU202403297

Therefore, given the tasks T_i constrained with (m_i, k_i)-firm deadline, the time utility $U(T_i)$ can be described in Equation (3.4).

$$
\begin{aligned}
U(T_i) &= \frac{1}{k_i}\sum_{j=1}^{k} u(\tau_i^j), \\
s.t \quad u(\tau_i^j) &= \begin{cases} 1, & R_i^j < C_i^j, \\ 0, & R_i^j \geq C_i^j \end{cases}
\end{aligned}
\tag{3.4}
$$

$u(\tau_i^j)$ is the utility within the instance τ_i^j. For the system satisfying the deadline, the time utility $U(T_i)$ must larger than m_i/k_i.

3.2.2 EtherCAT Model

EtherCAT structural model contains various elements that are essential for understanding and implementing EtherCAT-based control systems. At its core, EtherCAT is built on host-device communication architecture, where a master device coordinates communication with multiple slave devices distributed across the network.

The physical layer of the EtherCAT architectural model defines the network topology and hardware components. In this thesis, EtherCAT network is configured in a line topology, depending on the specific requirements of the application. Communication within the EtherCAT network is controlled by protocol stacking, facilitating real-time data exchange and synchronization between host and devices. The protocol defines frame formats, data transmission methods and error handling mechanisms to ensure high-speed and deterministic communication.

To create a mathematical model for EtherCAT, assumptions and parameters for EtherCAT model are listed below.

doi:10.6342/NTU202403297

Assumptions and Parameters:

1. N: Number of devices (nodes) on the network.
2. L: Length of the data payload in bytes.
3. D_p: Propagation delay
4. D_t: Transmission delay

Propagation delay(D_p) depends on the speed of the signal in the medium,which is typically close to the speed of light in a vacuum but slightly slower in cables. For a Ethernet cable, the propagation delay is approximately 5 nanoseconds per meter.

Processing delay in EtherCAT is very low because data is processed instantly as it passes through each node. Therefore processing delay is ignored in this article.

Transmission delay(D_t) is the time taken to push all the bits of the frame onto the wire. For Fast Ethernet (100 Mbps), the transmission delay for a frame size of L bytes is listed in Equation (3.5).

$$D_t = \frac{L * 8}{100 * 10^6} \tag{3.5}$$

The total latency for the EtherCAT network is the summation of propagation delay, transmission delay, and processing delay. Beacause processing delay is low , the relation is shown in Equation (3.6).

$$D_{latency} = D_t + D_p \tag{3.6}$$

doi:10.6342/NTU202403297

doi:10.6342/NTU202403297

Chapter 4

Semaphore-Based Real-time Scheduling

4.1 Delay Effect on Multiple Devices System

4.1.1 System Design

For the networked system, the performance directly influences the whole precision and response time of the servo system. The networked actuator system is designed from the inner loop to the outer, so that each loop is made stable. The system is described in Equation (4.1).

$$\dot{x}(t) = A_p x(t) + B_p u(t) \tag{4.1}$$

$x(t) \in R^n$ and $u(t) \in R^m$. If the data process delay is small enough to be neglected, the time delay induced by network is composed of sensor to controller delay $D_{sc}(t)$ and controller to actuator delay $D_{ca}(t)$. To simplify analysis, according to [46: Tran *et al.* 2013], $D_{sc}(t)$ and $D_{ca}(t)$ could be integrated into a single time communication delay $D_{comm}(t)$ without affecting the control performance. The relation is displayed in Equation (4.2) .

$$D_{comm}(t) = D_{sc}(t) + D_{ca}(t) \tag{4.2}$$

doi:10.6342/NTU202403297

4.1.2 Multiple Tasks in the Drive for Scheduling

The drive software has been developed in a multitasked structure using priority-based preemptive scheduling. Considering the resource-constrained drive environment, the kernel was carefully designed to minimize the overheads by including only basic primitives such as task scheduling, timer, and synchronization. Based on the kernel primitives, the drive function has been decomposed into four tasks: the MotorAct, MotorMon, MsgInt, and MsgMain tasks. The timing diagram is displayed in Figure 4.1.

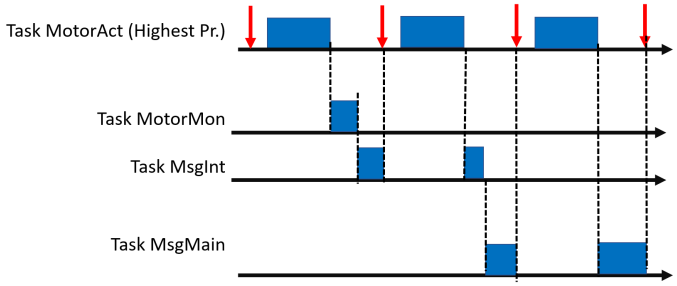


Figure 4.1: Timing diagram for the execution of tasks

For the **task MotorAct** ($T_{MotorAct}$), it executes the control loop to reach the requested target position/velocity, which has the highest priority within these for tasks. The release time of task MotorAct is equal to the time of interrupt. The **task MotorMon** ($T_{MotorMon}$) is defined to monitor the the motor status. The **task MsgInt** (T_{MsgInt}) deals with the real-time PDO for interrupt from the EtherCAT SubDevice Controller (ESC). These data are shared with the task MotorAct and status update. This task is seen as a periodic task, which set the master cycle time as the period.

From [47: Sung *et al.* 2011], results displayed that interrupts from the EtherCAT have a high degree of periodicity with a jitter of 2 - 3 μs. The **task MsgMain** executes the corresponding operations in response to state change requests initiated by the EtherCAT master. Another function of the task MsgMain is to deal with the non-real-time data transfer. Because of the characteristics of the task MsgMain, it runs in background with

doi:10.6342/NTU202403297

the lowest priority. The order of priority is shown in Equation (4.3).

$$P_{MotorAct} > P_{MotorMon} > P_{MsgInt} > P_{MsgMain} \tag{4.3}$$

4.1.3 Drive-Local Delay for Host Command

When a packet is sent from the host to a device, the device will check whether the packet has been received or not. For the k-th packet, the variable $\alpha_i(k)$ to show the relation can be discribed in Equation (4.4).

$$\alpha_i(k) = \begin{cases} 0, packet \quad loss \\ 1, packet \quad received \end{cases} \tag{4.4}$$

The host generates the packet, and this packet is then transmitted over EtherCAT. Upon reaching the device, the device's internal controller then interprets the command and executes the required operations, such as reading or writing to storage. Refer to [46: Tran *et al.* 2013], drive-local command delay is analyzed. As one of the essential performance metrics of the drive, the drive-local command delay $D_{drive}(t)$ can be calculated in Equation (4.5).

$$D_{drive}(t) = R_{MsgInt}(t) + C_{MotorAct}(t) + R_{MotorAct}(t) \tag{4.5}$$

Where $R_{MsgInt}(t)$ is worst-case response time of T_{MsgInt}; $C_{MotorAct}(t)$ is the period of $T_{MotorAct}$; $R_{MotorAct}(t)$ is the worst-case response time of $T_{MotorAct}$. The task T_{MsgInt} is cyclically activated by the host command, so it can be seen as a periodic task. The values of

doi:10.6342/NTU202403297

R_{MsgInt} and $R_{MotorAct}$ are calculated for each command mode, i.e., the position, velocity, or torque mode according to a deterministic delay bound analysis called the time demand analysis.

In this situation , for the n-th drive in the network, the total end to end command delay($D(n,t)$) is the summation of the delay for communication and the delay from drive, which is shown in Equation (4.6).

$$D(n,t) = D_{comm}(n,t) + D_{drive}(t) \tag{4.6}$$

From [48: Liang *et al.* 2019], the communication delay can be also seen as the summation of time from host and the the k-th time form drive. This can be displayed in Equation (4.7).

$$D_{comm}(n,t) = t_{host}(t) + n * t_{drive} \tag{4.7}$$

t_{drive} is forwarding time of a message from one drive to another one, which is a constant determined by network topology and can be measured by the packet network packet analyzer. $t_{host}(t)$ is a variable that change over time, which means the forwarding time from host to the first node in the network. By Equation (4.2) and Equation (4.7), the communication delay can be measured and calculated.

doi:10.6342/NTU202403297

4.2 System Loop and Discretization

Figure 4.2 illustrates a control system for a single drive, denoted as Gc. The system begins with a summation block that compares the reference input with the feedback signal. The error generated from this comparison is fed into a Proportional-Integral (PI) controller, which adjusts the system response to minimize the error over time by considering both the magnitude and the accumulated error. The PI controller processes this error to generate a reference current (I_{ref}). The actual motor current (I_s) is then subtracted from Iref to produce a current error (I_e). This current error is used by the pulse-width modulation (PWM) control module to generate the duty cycle command (D_{comm}), which adjusts the motor's power input. The motor responds to the Dcomm signal, adjusting its velocity. The actual motor velocity is then fed back to be compared again with the command velocity, closing the control loop and ensuring the motor speed aligns with the desired setpoint. This feedback loop helps maintain precise control over the motor's velocity.

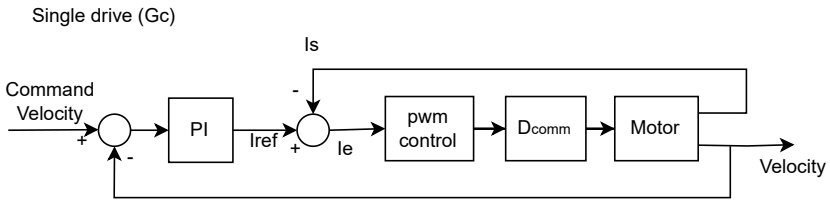


Figure 4.2: Block diagram of the single drive

Figure 4.3 depicts the multi-drive control system managed by a host. Loop consists of a scheduler, a controller, and a drive G_c. This setup ensures synchronized operation and coordination of multiple drives, allowing the host to manage complex tasks by distributing control commands and monitoring performance.

Refer to [49: Branicky *et al.* 2002], there are three assumption to simplify and calculate the system for discretization, and add the delay $D_{comm}(n,t)$ in the common system,

doi:10.6342/NTU202403297

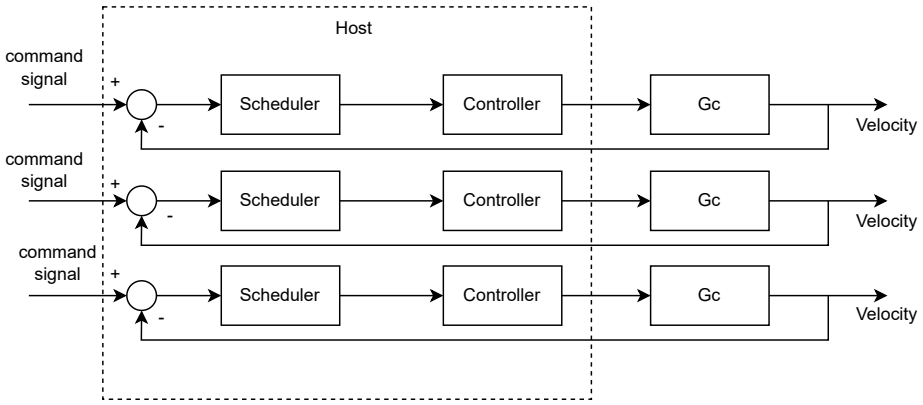


Figure 4.3: Block diagram of multiple drives system

which is written in Equation (4.8).

$$\dot{x_{pi}}(t) = A_{pi}x(t) + B_{pi}u_i(t - D_{comm}(n,t)) + B_{wi}(t) \tag{4.8}$$

Assumption:

1. Controllers and actuators are event-driven.

2. Delay $D_{comm}(n,t)$ exists in the transmission of the network.

3. In the network, there are p actuators.

Tasks must be scheduled and executed at precise intervals. Discretization helps in defining these intervals, ensuring that tasks are executed predictably and consistently, which is critical for maintaining system stability and meeting deadlines. Refer to the method in [49: Branicky *et al.* 2002], discretization is shown in Equation (4.9).

$$x(k+1) = Ax(k) + B_0(D_{comm}(n,t))u(k) + B_1(D_{comm}(n,t))u(k-1) \tag{4.9}$$

doi:10.6342/NTU202403297

In Equation (4.9), the $B_0(D_{comm}(n,t))$, $B_1(D_{comm}(n,t))$ can be rewritten belowed.

$$\begin{aligned} B_0(D_{comm}(n,t)) &= B_0 + HF(D_{comm}(n,t)')B_p, \\ B_1(D_{comm}(n,t)) &= B_1 - HF(D_{comm}(n,t)')B_p, \\ D_{comm}(n,t)' &\in (-s/2, s/2) \end{aligned} \tag{4.10}$$

In Equation (4.10), the B_0, B_1, H are constant matrice and are set as Equation (4.11).

$$\begin{cases} B_0 = \int_0^{s/2} e^{A_p t} B_p \, dt \\ B_1 = \int_{s/2}^{2} e^{A_p t} B_p \, dt \\ H = \| \int_0^{s/2} e^{A_p t} \, dt \|_2 e^{A_p s/2} \end{cases} \tag{4.11}$$

$F(D_{comm}(n,t)')$ is a time-varying matrix and is displayed in Equation (4.12).

$$F(D_{comm}(n,t)') = \| \int_0^{s/2} e^{A_p t} \, dt \| \int_0^{D_{comm}(n,t)'} e^{A_p t} \, dt \tag{4.12}$$

$F(D_{comm}(n,t)')$ with this form will satisfy Equation (4.13).

$$F^T(D_{comm}(n,t)')F(D_{comm}(n,t)') \leq I \tag{4.13}$$

doi:10.6342/NTU202403297

4.3 Details of the Scheduling Method

In the EtherCAT network, the semaphore functions here like a sensor, and its function is to monitor and control the use of various tasks and resources in the network. Specifically, the semaphore is obtained by the EtherCAT Master, which means that the host will automatically detect and obtain the semaphore in the network when the network is connected. These semaphores serve as a scheduling and synchronization mechanism, allowing host to effectively manage resources in the network to ensure timely transmission of data and efficient operation of the system.

The algorithm is designed to facilitate efficient task management and synchronization in the system within a real-time operating system framework Xenomai. At its core, the algorithm ensures precise timing and coordinated execution of critical tasks for applications where timing accuracy and responsiveness are paramount.

The algorithm initializes by defining a MINIMUM_TASK_PERIOD, which establishes the smallest interval at which tasks can execute. This parameter ensures granularity in task scheduling, allowing threads to operate with fine-tuned timing precision. Threads are instantiated and configured with specific priorities and execution frequencies (thread_1, thread_2, thread_3). Each thread performs a set of device operations and operates within its dedicated execution loop, continuously monitoring system time to determine when tasks should be executed. The initialization is displayed in **Algorithm 1**.

Refer to [50: Zhang *et al.* 2018], threads periodically check if the current system time aligns with their predefined execution intervals (MINIMUM_TASK_PERIOD * period_multiple). For instance, thread_1 may execute tasks every MINIMUM_TASK_PERIOD, while thread_2 and thread_3 execute tasks every MINIMUM_TASK_PERIOD * 2 and

doi:10.6342/NTU202403297

Algorithm 1 Initialization for semaphore-based real-time scheduling

```
MINIMUM_TASK_PERIOD ← 10                                   ▷ milliseconds
Thread Functions:
function INITIALIZE_THREADS
    create_thread(thread_1)
    create_thread(thread_2)
    create_thread(thread_3)    ▷ thread represent the computing task for each device
end function
function THREAD
    while true do
        current_time ← get_current_time()
        if current_time ≥ thread_data[0].last_execution_time + (MINIMUM_TASK_PERIOD × thread_data[0].period_multiple) then
            acquire_semaphore(sync_sem)
            execute_task_for_thread_1()
            thread_data[0].last_execution_time ← current_time
            release_semaphore(sync_sem)
        end if
        sleep(MINIMUM_TASK_PERIOD / 2)                     ▷ Sleep half the minimum period
    end while
end function
```

MINIMUM_TASK_PERIOD * 4, respectively. This setup ensures that tasks are executed at predictable intervals. To prevent concurrency issues and ensure orderly task execution, threads acquire the semaphore to gain exclusive access to shared resources before executing tasks.

The main scheduler loop orchestrates the overall system synchronization and timing. It waits for real-time clock interrupts, synchronizing simulation time with system time to maintain temporal accuracy. At predefined synchronization points, threads synchronize their activities using sync_sem. This coordinated approach optimizes resource utilization and ensures consistent execution flow across all threads. The main loop of scheduler is displayed in **Algorithm 2**.

This algorithm effectively manages multiple threads by ensuring that each thread executes its tasks at predetermined intervals based on their specific period multiples. By

doi:10.6342/NTU202403297

Algorithm 2 Main loop for semaphore-based real-time scheduling

```
1:  function SCHEDULER
2:      initialize_threads()
3:      while true do
4:          wait_for_real_time_clock_interrupt()
5:          synchronize_simulation_time_with_system_time()
6:          if current_time%MINIMUM_TASK_PERIOD == 0 then
7:              synchronize_threads()
8:          end if
9:          sleep(MINIMUM_TASK_PERIOD / 2)        ▷ Adjust sleep time based on requirements
10:     end while
11: end function
```

using a semaphore for synchronization, it avoids race conditions and ensures that the critical section of code, where tasks are executed and recorded. This prevents timing conflicts and maintains the integrity of execution times.The hierarchical structuring of task execution times ensures that tasks with higher urgency or frequency are given priority without starving less frequent tasks.

Compare with RMS, EDF, and FCFS, semaphore-based real-time scheduling has its distinct advantages and disadvantages. The comaprison is shown in Table 4.1. Semaphore-based scheduling provides robust synchronization and mutual exclusion, preventing race conditions and ensuring resource availability. However, it may lead to priority inversion, complicating system debugging. RMS, a fixed-priority algorithm, is simple and predictable, making it suitable for systems with static priorities. Its downside is that it can be inefficient for tasks with varying execution times and deadlines. EDF, a dynamic priority algorithm, optimizes CPU utilization by scheduling tasks based on their deadlines, providing flexibility for dynamic workloads. Yet, it can be complex to implement and may struggle with overload conditions. FCFS is the simplest algorithm, easy to implement with minimal overhead, but it fails to prioritize critical tasks, leading to potential deadline misses in real-time systems.

doi:10.6342/NTU202403297

Table 4.1: Comparsion of semaphore-based with other common scheduling method

Feature	Semaphore-Based	RMS	EDF	FCFS
Synchronization	Prevents race conditions	Not inherently synchronized	Not inherently synchronized	Not inherently synchronized
Priority Management	May lead to priority inversion	Fixed-priority	Dynamic priority	No priority management
Complexity	May be complex	Simple	Complex	Very simple
CPU Ussage	Lower	Lower	High	Lower
Implementation	Requires careful debugging	Easy to implement	Complex t0 implement	Easiest to implement

doi:10.6342/NTU202403297

doi:10.6342/NTU202403297

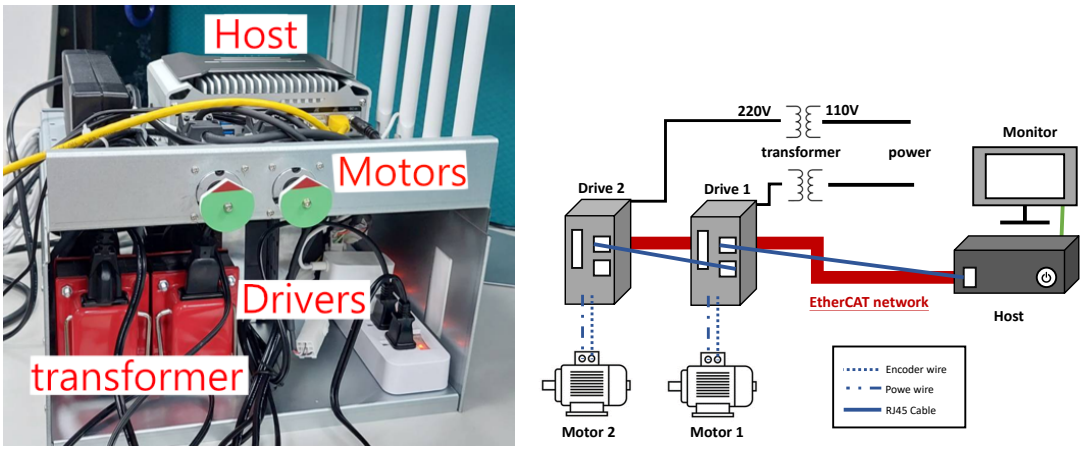
Chapter 5

Software and Hardware Platform

In this chapter, the software and hardware designed are discussed. First, the overall environment is shown. Then, the hardware and software plaform are talked about individually in the next sections.

5.1 Overall of the Environment

The overall system of the environment is displayed in Figure 5.1. The environment is consisted of two motors with its drivers in the middle of the picture. The motors have their pointers so that can be observed easily. The motors use a power wire and an encoder wire to connect to its drivers induvidually. The drives use the RJ45 cable wire to connect with the IPC and another drives so that the EtherCAT network can be built. The schematic diagram is provided in Figure 5.1(b). In the environment, the host and the motors are placed at the top, and the drivers and the transformer are placed on the lower floor. The detailed information of hardwares will be discussed in the next section.



(a) Picture **(b) Schematic diagram**

Figure 5.1: Platform environment

doi:10.6342/NTU202403297

To control and observe the environment, a human machine interface is provided in Figure 5.2. In this pltform, openplc is the main sofware to control motors throgh EtherCAT protocol. The openPLC runtime is running in the background like the picture shown. Then, a monitoring software, VTscasa is runing through a vurtual machine in the front of Figure 5.2. The monitoring software can give the control indication and observe the system with the cylce of 0.1 second via modbus TCP. The details of the software plaform is duscussed in the section 5.3.

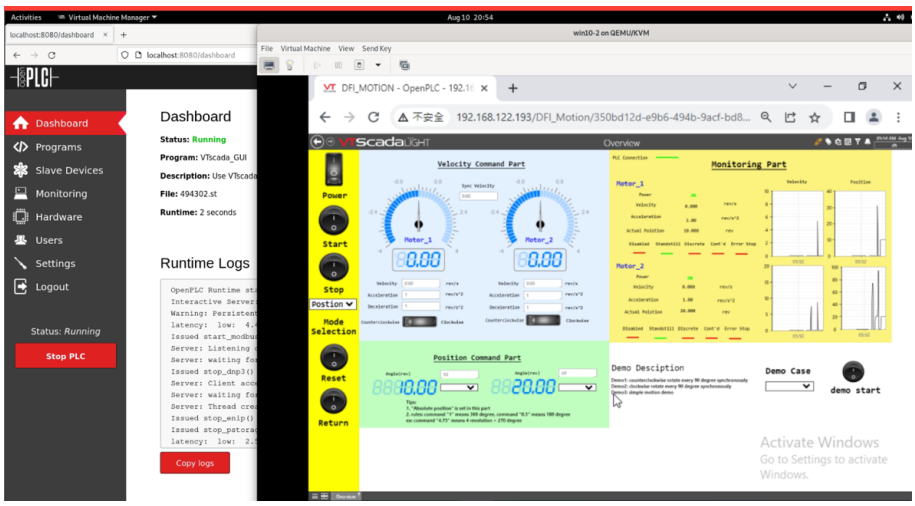


Figure 5.2: Human machine interface

5.2 Hardware Platform

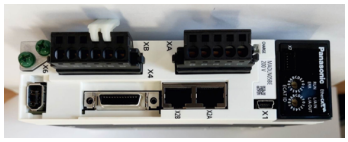
5.2.1 EtherCAT Motors and Drivers

EtherCAT motors typically refer to servo motors or stepper motors equipped with EtherCAT-compatible communication interfaces, allowing them to be controlled and synchronized over an EtherCAT network. The main advantages of EtherCAT motors and drivers is their ability to achieve precise synchronization and coordination of motion across multiple axes in real-time system. The EtherCAT motors used in this paper is an servo motor prduced by Panasonic. Its picture and the specification are listed in Table 5.1.

doi:10.6342/NTU202403297

EtherCAT drivers are the hardware or software components responsible for interfacing with EtherCAT-enabled motors and facilitating their control. These drivers typically reside in programmable logic controllers (PLCs), motion controllers, or other automation devices and are responsible for sending commands, receiving feedback, and managing communication with EtherCAT motors. The EtherCAT drivers used in this paper provides AC for matching servo motor. Its picture and the specification are listed in Table 5.1.

Table 5.1: Specification of the hardwares

Item	Specification	
EtherCAT motors	Model	MSMF012L1U2M
	Brand	Panasonic
	Weight	0.47Kg
	Dimension	6cmx7cmx14cm
	Voltage	200V
	Rated power	100W
	Rated torque	0.32 N·m
	Max. speed	6000 rev/min
Drivers	Model	MADLN05BE
	Brand	Panasonic
	Control/drive type	Servo/AC
	Voltage	200V-240V
	Rated power	200W
IPC	Model	EC70A-TGU
	Brand	DFI
	CPU	11th Gen Intel $Core^{TM}$ i5
	Memory	8GB

5.2.2 Industrial Cpmputer

Industrial computers (IPC) and common computers serve distinct purposes and are optimized for different environments. Industrial computers are designed to withstand harsh conditions such as extreme temperatures, dust, humidity, and vibrations, making them suitable for use in industrial settings. The feature ruggedized construction, specialized components, and extensive connectivity options tailored to industrial applications.

doi:10.6342/NTU202403297

The specification of IPC used in this paper are listed in Table 5.1.

5.3 Software Platform

In this section, the software platform is talked about. First, the platform provided by Intel is introduced. Then, openPLC, the software used for motion control, is discussed. The montoring software and the combination of this platform are presented in the third part of this section.

5.3.1 Platform: Intel Edge Controls for Industrial

Refer to [51: Industrial *et al.*], Intel Edge Controls for Industrial (ECI) are a suite of software solutions designed to optimize and manage industrial processes at the edge of the network. These solutions leverage Intel's expertise in edge computing, real-time analytics, and industrial automation to help users improve efficiency, productivity, and safety in their operations.

In ECI, the usage of virtualization and containerization configurations consolidates mixed-criticality workloads. One key aspect of Intel Edge Controls is the ability to collect, process, and analyze data from industrial equipment and sensors in real-time. This enables predictive maintenance, anomaly detection. Another important feature of Intel Edge Controls is their support for industrial protocols and standards, allowing seamless integration with existing equipment and systems. In this thesis, ECI is used for the basic platform to build the real-time environment.

doi:10.6342/NTU202403297

5.3.2 Motion Controller: OpenPLC

OpenPLC is an open-source platform for Programmable Logic Controllers (PLCs), which is the key point to be chosen in this thesis. OpenPLC allows users to freely access, modify, and distributes the software according to their needs. The additional function of OpenPLC is an editor. It supports standard programming languages, making it easy to compile and execute the code directly in one software. The dashboard of openPLC is shown in Figure 5.3. It also supports various communication protocols commonly used in industrial automation, such as Modbus, EtherNet/IP, Profinet, and MQTT. This interoperability enables seamless integration with existing equipment and systems. That is the main reason that openPLC is chosen as the control application software.

In this control systems, the Proportional-Integral-Derivative (PID) control method is employed for its effectiveness in managing and optimizing industrial processes. PID controllers adjust process variables by calculating the error value as the difference between a desired setpoint and a measured process variable. By accessing a PID function block within the software's library, PID controllerand is integrated and configured by setting the proportional (Kp), integral (Ki), and derivative (Kd) gains to meet the requirements. Define the setpoint value that the PID controller will maintain, and ensure a feedback loop is established for continuous error correction.

Figure 5.4 shows the structure of openPLC. The kernel used is the Linux based Debian 11 (bullseye) with the Xenomai cobalt as the RTOS, letting the scheduling tasks and the motion control task with EtherCAT communication have the highest priority to running in the real-time kernel. In the core of openPLC, modbus is the protocol to communicate

doi:10.6342/NTU202403297

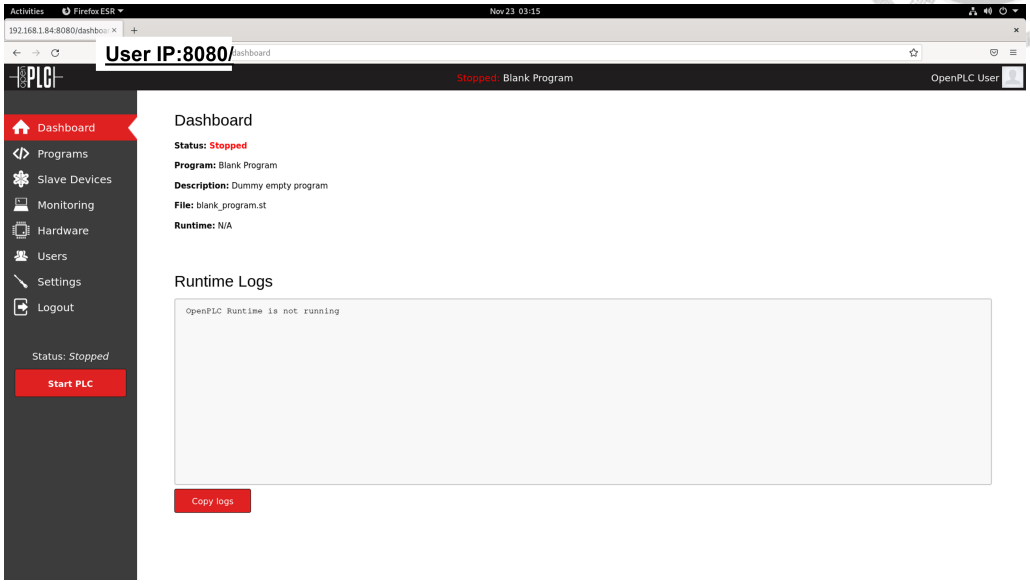


Figure 5.3: Dashboard of the openPLC

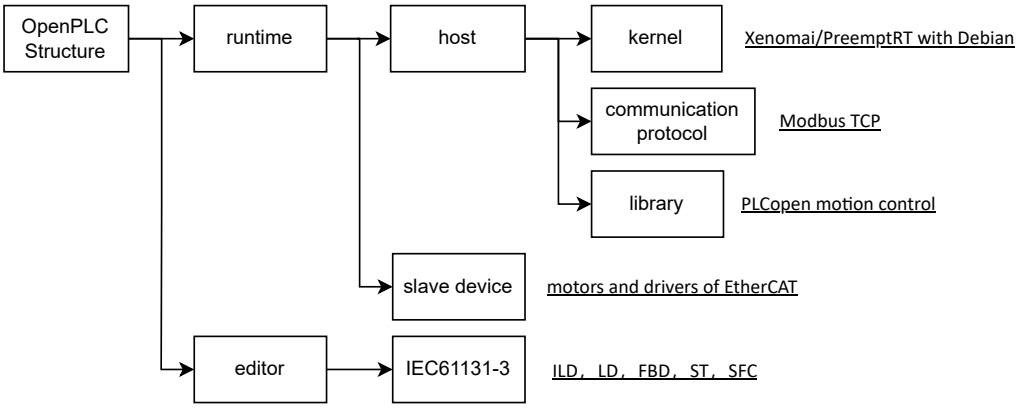


Figure 5.4: Structure of openPLC

doi:10.6342/NTU202403297

and also it can be a great method to combine the the monitoring software.

In the openPLC runtime, three parts can be listed in Figure 5.5. In the core part, files are converted from ST file, the feature that openPLC generated with IEC61131-3 standard, into common C code. Then, it also defines the hardware layer in the needed project,. Another important part is the utils, which contains the function to let the code be generated. The library part provides the whole needed function block.

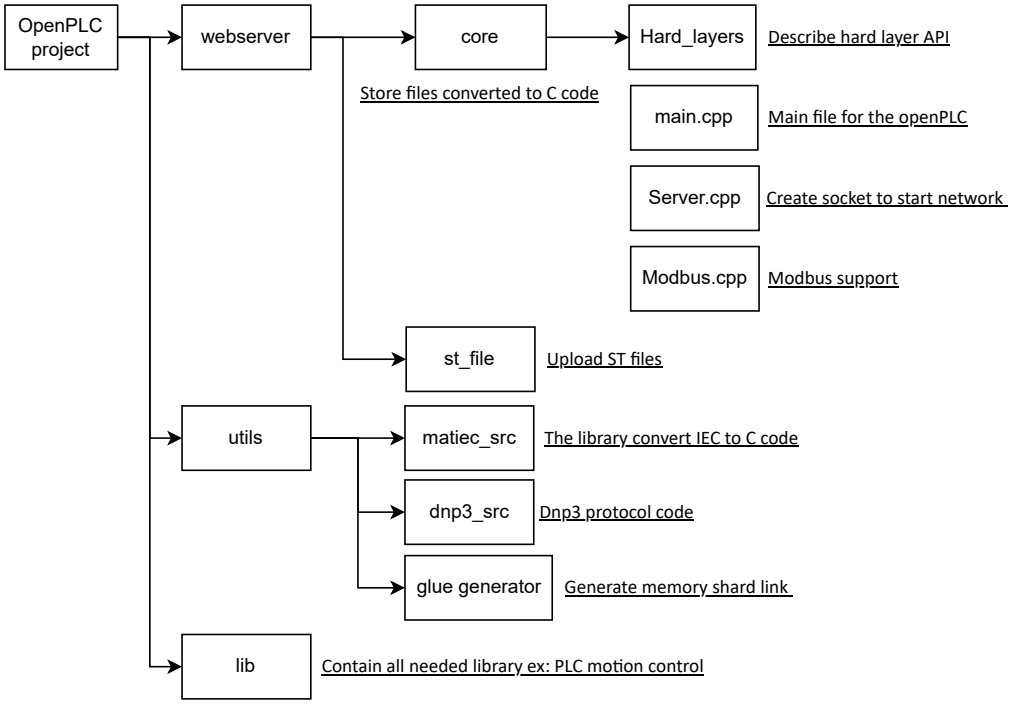


Figure 5.5: Structure of open-source openPLC structure

The following function block (FB) library is designed for the purpose of controlling axes via the language elements consistent with IEC 61131-3 standard, and their function block diagram is listed in Table 5.2. To control each motor and combine the openPLC runtime with monitoring software intuitively, modularization is the method used in this thesis. It separates the code of each motor enforce modularity ,so that there is no trouble if more motors are added to the system. The function blocks are listed below.

1. **MC_Power**: The function block is used to switch the software enable of an axis.

doi:10.6342/NTU202403297

2. **MC_MoveVelocity**: The function block is used to start an endless motion with a specified velocity and direction. The movement can be stopped through a Stop command.

3. **MC_MoveRelative** and **MC_MoveAbsolute**: The function block MC_MoveRelative starts a relative positioning motion based on the current set position, and the function block MC_MoveAbsolute move to an absolute target position.

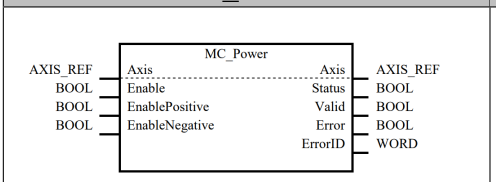
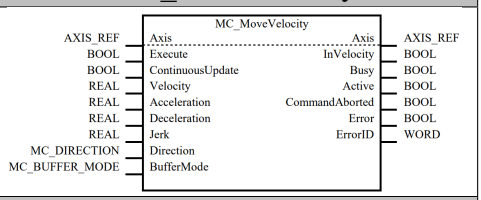
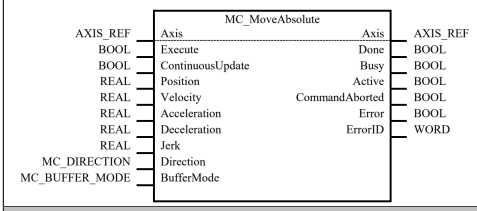
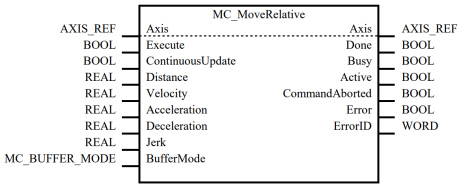
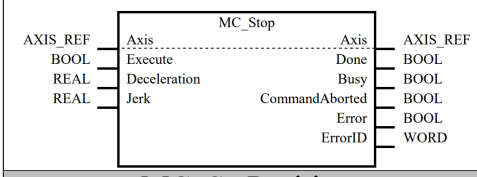
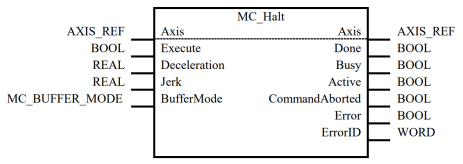
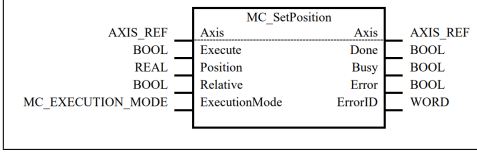
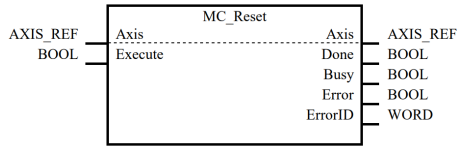
4. **MC_SetPosition**: In absolute mode, the actual position is set to the parameterized absolute position value. In relative mode, the actual position is offset.

5. **MC_Halt** and **MC_Stop**: These two function blocks represent to stop the motion. However, the function block MC_Halt is used to stop an axis with a defined deceleration ramp and in the MC_Stop function, the axis is not locked against further movement commands.

6. **MC_Reset**: The function block is used to reset axis if errors happen.

doi:10.6342/NTU202403297

Table 5.2: Function Blocks used in this thesis

MC_Power	MC_MoveVelocity
	
MC_MoveRelative	MC_MoveAbsolute
	
MC_Halt	MC_Stop
	
MC_SetPosition	MC_Reset
	

doi:10.6342/NTU202403297

5.3.3 Monitoring Application

Although there is a simple monitor in openPLC dashboard, it does not have enough function and visual platform for user to observe directly. Therefore, because of the feature of high compatibility of ECI, a monitoring software can be adapted with virtual machine (VM).

VTscada is chosen in this research. VTScada is a powerful supervisory control and data acquisition (SCADA) software platform developed by Trihedral Engineering Limited. According to [52: Trihedral], SCADA systems like VTScada are crucial components in industrial automation, allowing operators to monitor, control, and optimize processes in real-time system.

VTScada offers an intuitive and user-friendly interface, allowing operators to visualize data from sensors, actuators, and other devices in real-time. The platform provides customizable dashboards, trends, alarms, and reports, making it easy for users to monitor and analyze critical information effectively. VTScada supports a wide range of communication protocols and interfaces, allowing integration with various devices, systems, and databases.

The standard control buttons like power, start, stop and so on are listed below. Users can enter desired pattern to control the motors. The fuction in this HMI can be separated into four parts.

1. **Velocity command part (blue box)**:

   In this part, user can enter desired velocity, acceleration, deceleration and the direction of rotation to control different motors individually. Also, two visual dash-

doi:10.6342/NTU202403297

boards are shown so that users can observe the real-time data intuitively.

To execute velocity control, users need to turn on ”Power” button and select the mode to velocity mode first. Then, push ”Start” button to run motors after entering the velocity data. Motors will accelerate with the desired acceleration to the desired veloity and then will keep at the constant speed. The motion of the velocity command part is diplayed in Figure 5.6(a).

2. **Position command part (green box)**:

User can enter desired position to control different motors individually. For setting pattern quickly, simple drop-down menus are set so that users can choose common pattern like 45, 90 and 180 degree directly.

To execute position control, users need to turn on ”Power” button and select the mode to Position mode first. Then, push ”Start” button to run motors after entering the position data. The system defaults forward direction to counterclockwise. If clockwise position is needed, just use minus notation to represent. Motors will accelerate with the default acceleration to the desired poistion and stop. The motion of the position part is displayed in Figure 5.6(b).

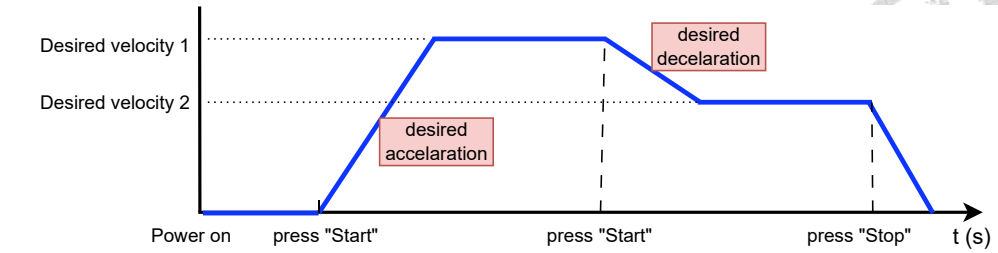
3. **Monitoring part (yellow box)**:

Users can observe the motion of motors more detailed by the instantly updated data and the position-time line chart and velocity-time line chart. For each motors, there are some signal to show the state is so that it can be more intuitive to check the running state and the error is.

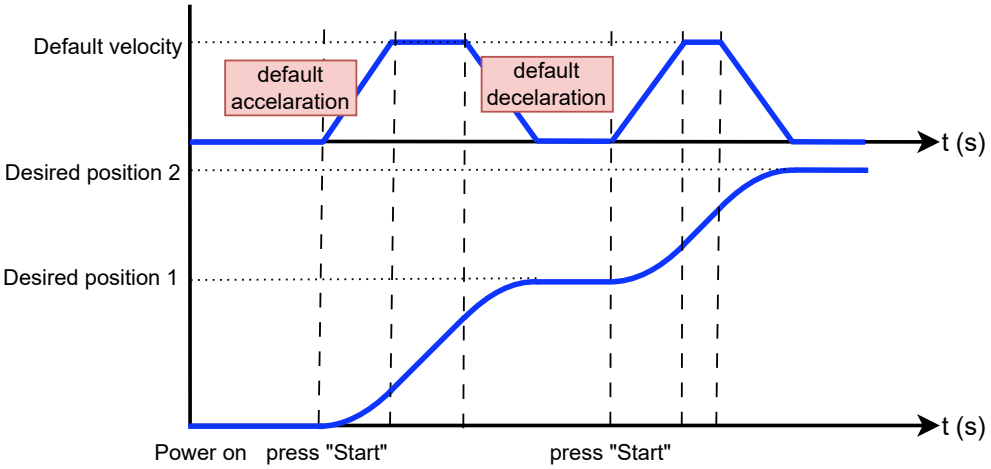
4. **Demo part (white box)**:

Demo part provides some simple demo motion to show. There includes three

doi:10.6342/NTU202403297



(a) Veolocity command part



(b) Position command part

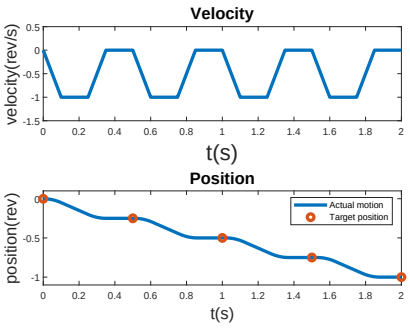
Figure 5.6: Timing diagram for different command parts

cases to understand this system more easiliy. The showcases are listed in Figure 5.7. The first one shows the continuously synchronized counterclockwise rotation with 90 degrees each step. The detailed motion and the timimg diagram are provided in Figure 5.7(a) and Figure 5.7(b). The second showcase shows the continuously synchronized clockwise rotation with 90 degrees each step. Its motion and timing diagram are listed in Figure 5.7(c) and Figure 5.7(d). The last showcase presents different veliocity on two motors. Its detailed motion and timing diagram are shown in Figure 5.7(e) and Figure 5.7(f). With these three demo case, users only need to select demo mode to understand the system.

doi:10.6342/NTU202403297

Demo case 1				
Continuously synchronized CCW rotation 90 degrees				
time (s)	0	0.5	1	1.5
position (°)	360	270	180	90

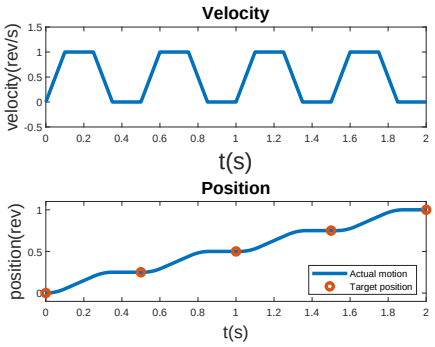
(a) Demo case 1



(b) Timing diagram of demo case 1

Demo case 2				
Continuously synchronized CW rotation 90 degrees				
time (s)	0	0.5	1	1.5
position (°)	0	90	180	270

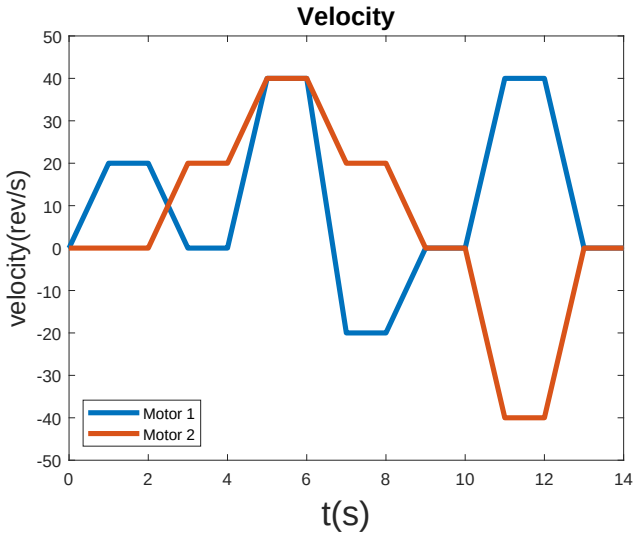
(c) Demo case 2



(d) Timing diagram of demo case 2

Demo case 3								
Different motion showcase								
motor	1	2	1	2	1	2	1	2
time (s)	0		2		4		6	
velocity (rev/s)	0	0	20	0	0	20	40	40
time (s)	8		10		12		14	
velocity (rev/s)	-20	20	0	0	40	-40	0	0

(e) Demo case 3



(f) Timing diagram of demo case 3

Figure 5.7: The pattern of three showcases

doi:10.6342/NTU202403297

The operation of the HMI is provided in Figure 5.8. Through modularization, each button can execute the program individually. The function of each button are listed below:

1. **Power**:

   The main function of Power button is for triggering the PLC motion function: MC_Power. If the Power button is pressed and the MC_Power.Enable equals FALSE, a pulse signal will be given and let MC_Power.Enable be TRUE. Then, the system will request to enter StandStll state. After it checks if the MC_Power.Powerstate is TRUE, which means that the system boots indeedly. The system will go to Stand-Still state and set the current position as zero point.

2. **Mode Selection**:

   By a drop-down menu, users can choose which control mode to use.

3. **Start**:

   The main fuction of the Start button is to give a signal for openPLC runtime, let the MC_MoveVelocity.Enable or MC_MoveRelative.Enable to be TRUE so that the system can start running as the given pattern.

4. **Stop**:

   When the Stop botton is pressed, the function MC_Stop will be triggered and the motors will stop all motion immediately. If the system checks that MC_Stop.Done is TRUE, the system will turn back to StandStill state.

5. **Reset**:

   When errors happen in any state, the system will give an alarm and go to ErrorStop state. In this state, all motion will be forced stop, waiting user to press

doi:10.6342/NTU202403297

Reset button. When users press the button, MC_Reset will be triggered. System will log the error message and check itself to restart. If it restarts sucessfully, the system will turn off the alarm, let MC_Reset be TRUE, and go back to Disabled state. Users need to press power again to reboot the system.

6. **Return**:

The Return button is an auxiliary button to let the motor back to the zero point.

doi:10.6342/NTU202403297

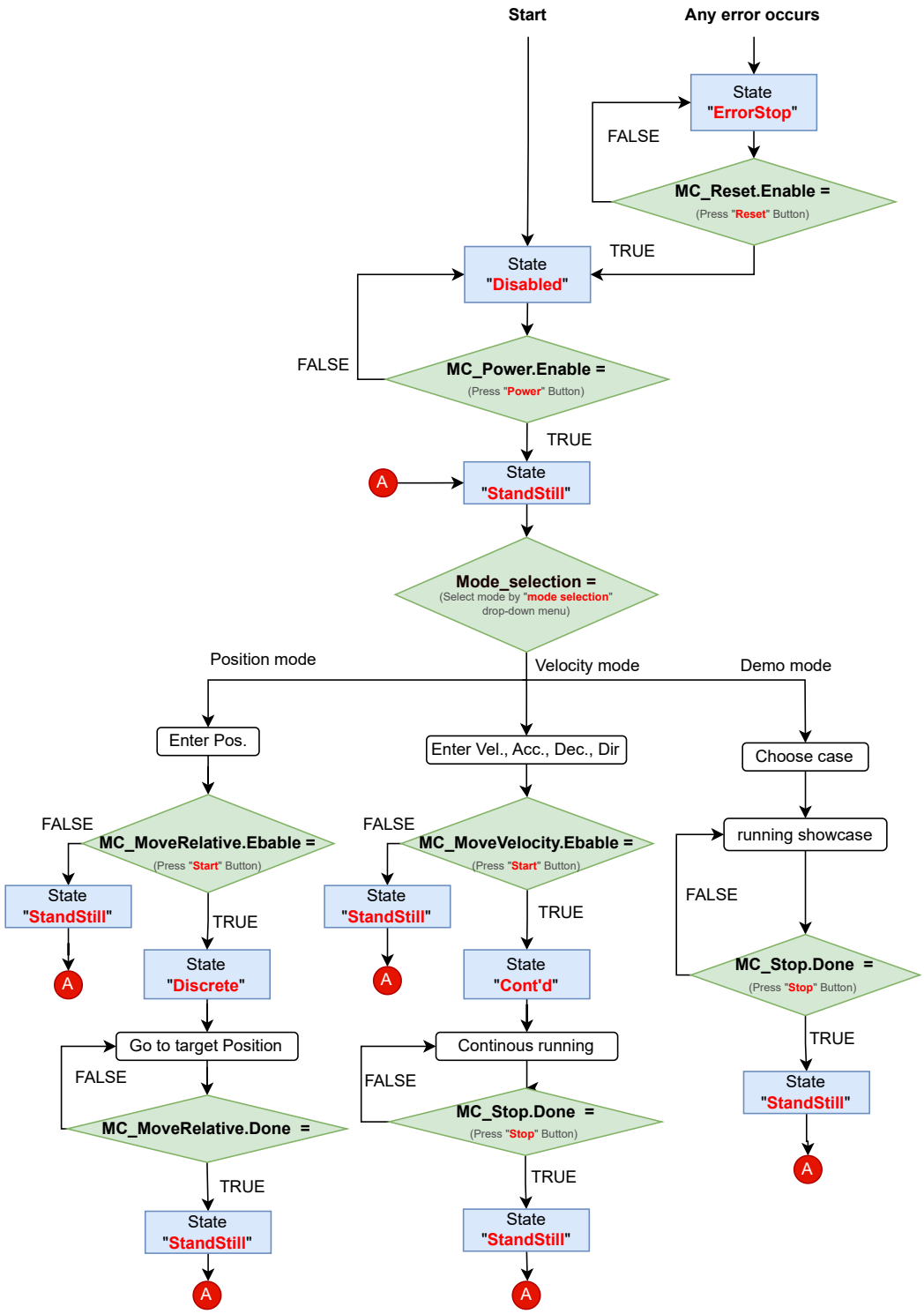


Figure 5.8: Flow chart of the operation in HMI

doi:10.6342/NTU202403297

To communicate with openPLC, modbus protocol is the suitable one in this communication. Modbus is a widely used communication protocol in industrial automation, facilitating the exchange of data between devices such as PLCs, Remote Terminal Units (RTUs), sensors, actuators, and other industrial control equipment according to [35: Organization]. VTScada supports both Modbus TCP/IP (Ethernet) and Modbus RTU (Serial) communication protocols. It only needs to define communication parameters through a graphical interface, simplifying the setup process. The rule of PLC and SCADA address mapping is listed in Table 5.3. The address should transform from PLC code to PLC address first, and then transform from PLC to modbus address. Finally, the address transforms to VTscada. Modbus uses a simple data model consisting of discrete input coils, holding registers, and so on. Discrete input coils represent binary states (on/off), while holding registers store analog data (integer values) such as sensor readings, setpoints, or control parameters. Overall, VTScada provides comprehensive support for Modbus communication, enabling to integrate PLC into SCADA applications.

Data type	Discrete Input Coils	Holding Registers
Usage	binary states	analog data
Data size	1 bit	32 bits
Access	RW	RW
PLC address	%QX0.0 –%QX99.7	%MD0 –%MD1023
Modbus address	0 –799 (ex: 10 = 1*8+2→%1.2)	2048 –4095 (ex: 2050=2048+2*1→%MD1)
VTscada Address	=Modbus address+1	=40000+Modbus address/Float

Table 5.3: Address mapping through different protocols

doi:10.6342/NTU202403297

doi:10.6342/NTU202403297

Chapter 6

Results and Analysis of Experiment and Simulation

In Chapter 6, the experimental setup, results, and analysis has been documented. In this thesis, due to the limitation of the hardware equipment, some experiment can not be implemented. To test the real-time motion control system discussed about in Chapter 5 and semaphore-based real-time scheduling, the experiment and simualtion are separated into three parts:

1. The performance under single axis with the real-time motion control system.
2. The performance under two axes with the real-time motion control system.
3. Simulation of the semaphore-based real-time scheduling.

In the following sections, three parts of experiment and simualtion are discussed separately.

6.1 Single Axis Situation with the Real-time Motion Control System

6.1.1 Experiment Setup

The real-time motion control system is described in Chapter 5. In this section, motion performance and real-time performance are tested under single axis situation. In order to get the experiment result, a non real-time monitoring task is created in the computer.

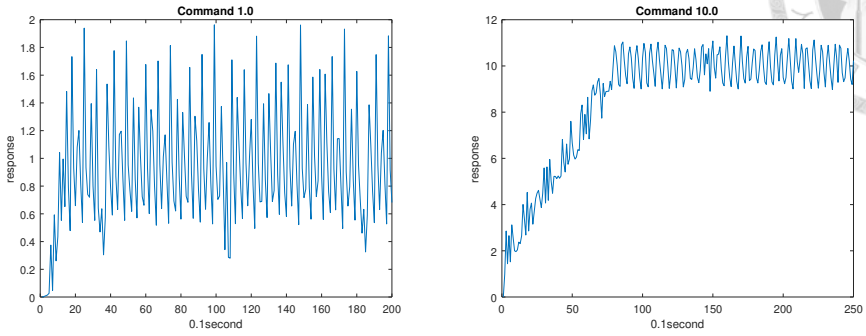
doi:10.6342/NTU202403297

The data from the encoder in the motor are logged to test the motion performance. First, linear calibration is done to get accurate command. The experiment shows the motion performacne with the real-time motion control system. Then, the test of the real-time performance is done and shows high real-time performance in the system under single axis situation.

6.1.2 Experiment Results and Analysis

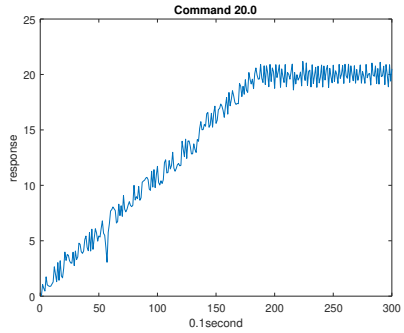
Following the described principle, we achieve the response of the different speed command under single axis situation. The result is shown in Figure 6.1. It shows that the motion of motor follows the given command, accelerating with constant acceleration to the command rotation speed and rotating at the constant speed. The error of the motion is 31.74% at the 1.0 command, 18.56% at the 10.0 command, 3.03% at the 20.0 command, and 1.47% at the 30.0 command. The comparison of different command is displayed in Figure 6.2. We can find that it is easy to understand the motion with the same acceleration. as the command value increases, the system's response improves significantly in terms of stability and accuracy, reflecting a reduction in the error rate.

doi:10.6342/NTU202403297

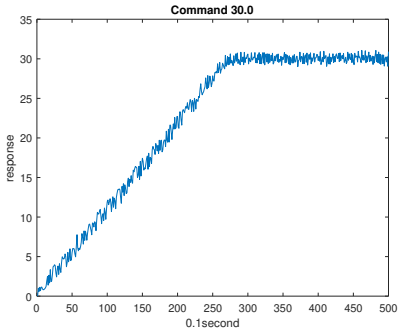


(a) command 1.0

(b) command 10.0



(c) command 20.0



(d) command 30.0

Figure 6.1: Individual timing diagram for response of the different speed commands

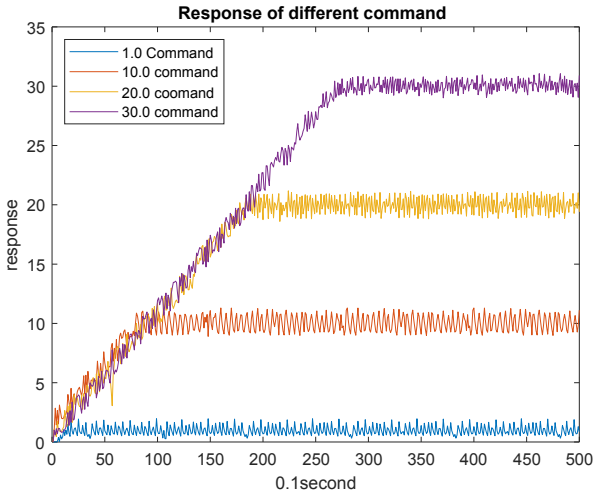


Figure 6.2: Timing diagram for the response of the different speed commands

doi:10.6342/NTU202403297

The result of performance is shown in Table 6.1. The analysis can be separated into four parts:

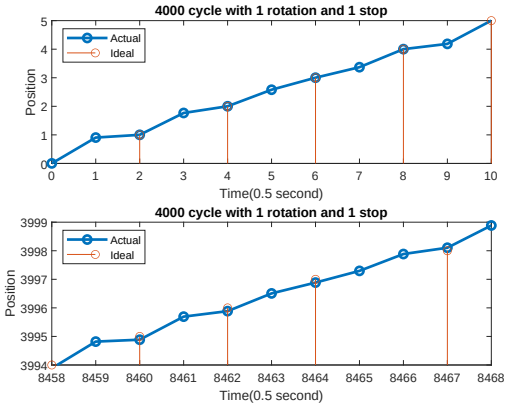
Table 6.1: The result of motor performance

Same rotations			
Item	Test 1	Test 2	Test 3
Behavior	1 rotation and 1 stop	10 rotations and 1 stop	100 rotations and 1 stop
Cycle	4000 times	400 times	40 times
Total rotation	4000	4000	4000
Result	Error at 4000: -0.1052 (-37.8°)	Error at 4000: -0.009 (-3.24°)	Error at 4000: 0.00024 (0.086°)
Same cycles without calibration			
Item	Test 4	Test 5	Test 6
Behavior	1 rotation and 1 stop	10 rotations and 1 stop	100 rotations and 1 stop
Cycle	1000 times	1000 times	1000 times
Total rotation	1000	10000	100000
Result	Error at 1000: -0.0231 (8.316°)	Error at 10000: -0.0244 (8.78°)	Error at 100000: -0.0313 (11.27°)
Same cycles with calibration			
Item	Test 7	Test 8	Test 9
Behavior	1 rotation and 1 stop	10 rotations and 1 stop	100 rotations and 1 stop
Cycle	1000 times	1000 times	1000 times
Total rotation	1000	10000	100000
Result	Error at 1000: -0.0022 (0.79 °)	Error at 10000: -0.0020 (0.72°)	Error at 100000: 0.0039 (1.40°)

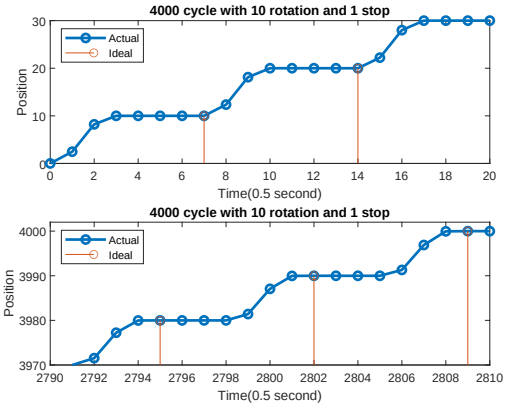
1. **The comparision of different behavior with the same rotation(Test 1 - Test 3):**

   Figure 6.3 shows the behavior of the motors. To check the source of the error, the motor is set to rotate 1 time and urgent as 1 cycle (Test 1). Another 2 tests are similar to this form, rotating 10 times and 100 times individually(Test 2 and Test 3). From test 1 - test 3, we can find that the error enumerated with the cycle. The enumeration of error within 4000 rotation is provided in Figure 6.4. Whether the behavior is, the errors increase as the number of the rotation. At the 4000 rotation, the errors are up to -37.8°, -3.24°, and 0.086°. However, there are significant difference in these three tests as shown in Figure 6.4(d), which means that the total number of roatation is not the main reason of the error.

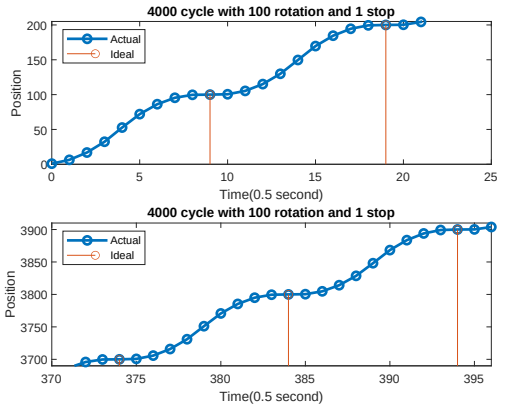
doi:10.6342/NTU202403297



(a) 1 rotation every cycle at 0 and 4000 rotation(Test 1)



(b) 10 rotation every cycle at 0 and 4000 rotation (Test 2)



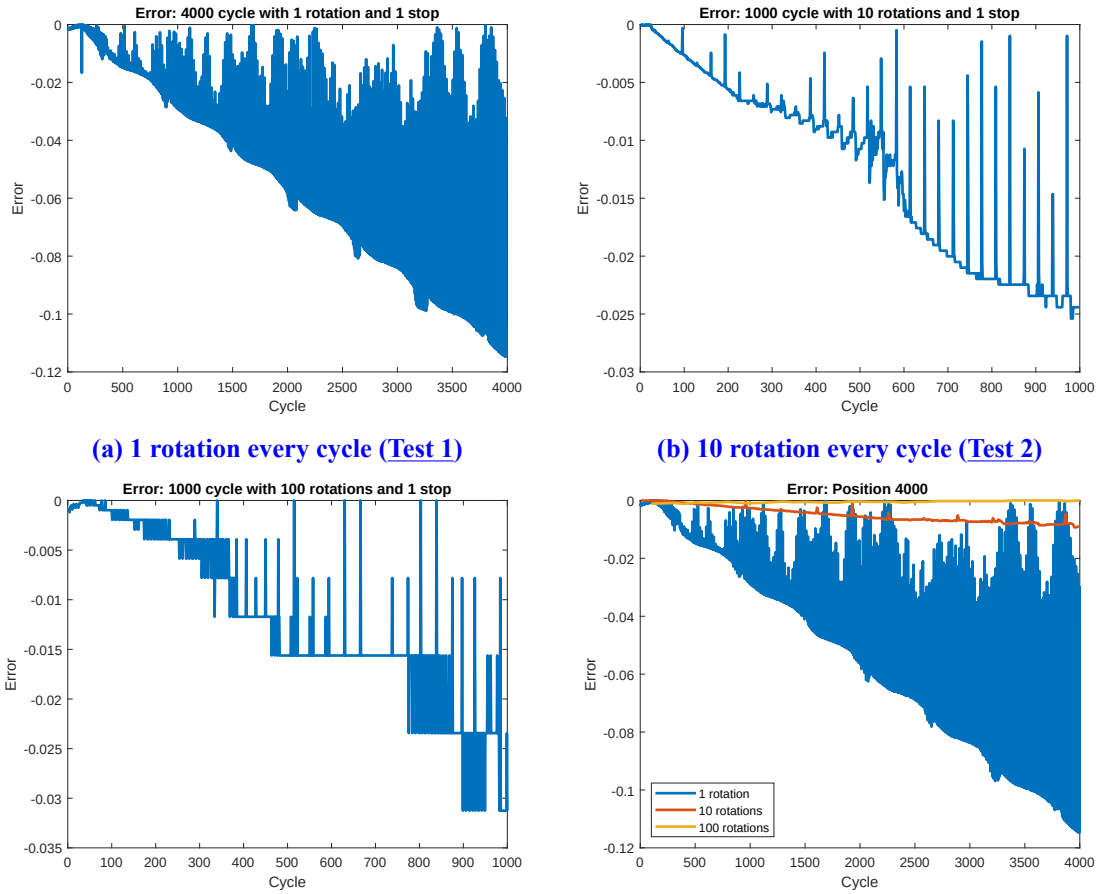
(c) 100 rotation every cycle at 0 and 4000 rotation (Test 3)

Figure 6.3: Error-cycle diagram for the the comparison of calibration

doi:10.6342/NTU202403297

2. **The comparision of different behavior with the same cycle(Test 4 - Test 6):**

Figure 6.5 shows the error for the different behavior with the same cycle. We can find that in the case of the same cycle, the motor has similar error accumulation and the errors at different cycle are also similar. From Table 6.1, the error at 1000 cycle are 8.316°, 8.78°, and 11.27°.



(a) 1 rotation every cycle (Test 1)

(b) 10 rotation every cycle (Test 2)

(c) 100 rotation every cycle (Test 3)

(d) Combination

Figure 6.4: Error-cycle diagram for the same rotation.

doi:10.6342/NTU202403297

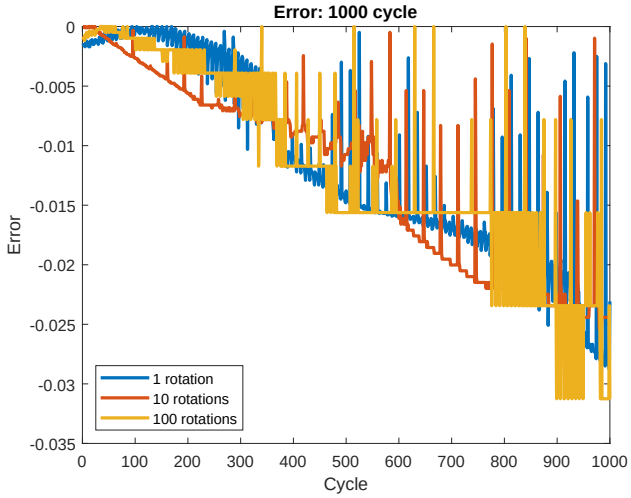


Figure 6.5: Error-cycle diagram for the same cycle (Test 4,5,6)

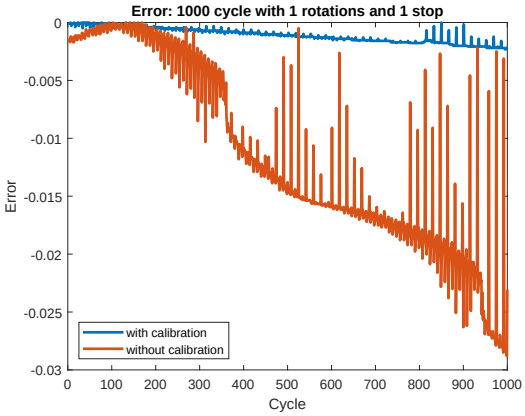
3. **The comparision of the behavior under the same rotations case (Test 1 - Test 3) and the behavior under the same cycles(Test 4 - Test 6):**

   By comparing test 1 - test 3 and test 4 - test 6, the error is positively related to the cycle defined here, which means that the urgent cause the error. To compensate it, a proportional–integral regulator (PI regulator) is used.

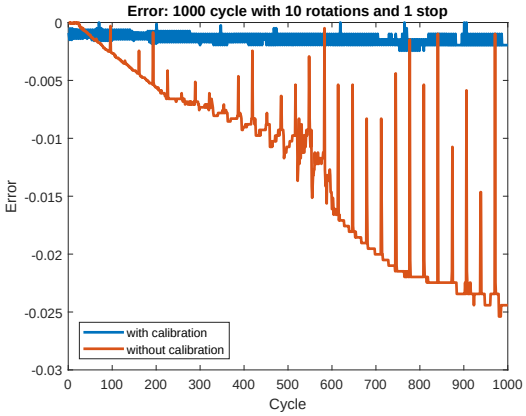
4. **The comparision of the same rotation case without calibration(Test 4 - Test 6) and with calibration(Test 7 - Test 9):**

   Figure 6.6 shows the result of the compensation. The result of the case with compensation is listed in Table 6.1(Test 7 - Test 9). We can find that the compesator can indeed reduce the error accumulation. The errors reduce to around 1° at the 1000 cycle, reducing up to 90%.

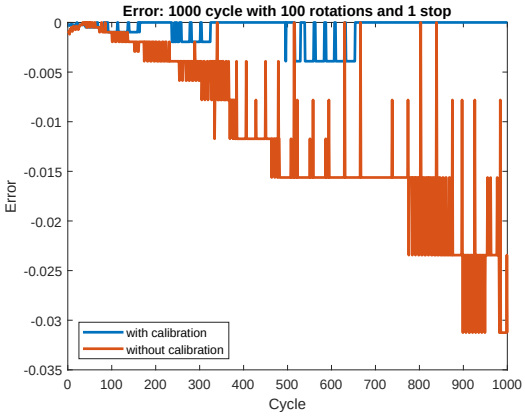
doi:10.6342/NTU202403297



(a) 1 rotation every cycle (Test 7)



(b) 10 rotation every cycle (Test 8)

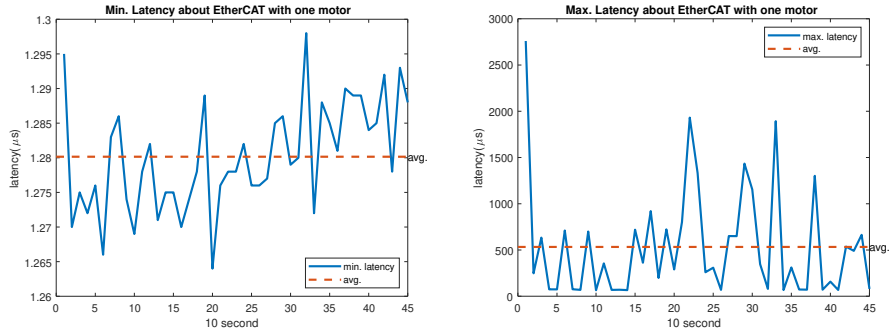


(c) 100 rotation every cycle (Test 9)

Figure 6.6: Error-cycle diagram for the the comparison of calibration

doi:10.6342/NTU202403297

The real-time performance under single axis is shown in Table 6.2 and Figure 6.7. Table 6.2 summarizes the maximum and minimum values of latency. We can remark that the implementation on the real-time motion control platform with PLC motion controller can achieve great real-time performance while maintaining low CPU usages. Besides, it should be noted that the stability and performance of our proposal on the real-time motion control platform have the finer timer resolution of the Xenomai patch (1 μs). In conclusion, these measurement results prove that the design can fulfill the basic functional requirements under single axis condition.



(a) Minimum latency **(b) Maximum latency**

Figure 6.7: Timing diagram for the minimum and maximum latency in single axis situation

Table 6.2: Minimum and maximum latency in single axis situation

	average	std
Min. latency(μs)	1.2802	0.0079
Max. latency(μs)	533.034	596.82
CPU usage(%)	23.47	0.12

doi:10.6342/NTU202403297

6.2 Multiple Axes Situation with the Real-time Motion Control System

6.2.1 Experiment Setup

Next, the situation of multiple axis case is discussed. In this section, motion performance and real-time performance are tested under multiple axes situation. Due to the limitation of the quantity of equipment, the experiment is done with two motors. The situation for over two axes is done with simualtion. The method to test multiple axes is the same as the previous scection that the simuation under single axis. In order to get the experiment result, two non real-time monitoring tasks are created to get the real-time data in the computer separately.

6.2.2 Experiment Results and Analysis

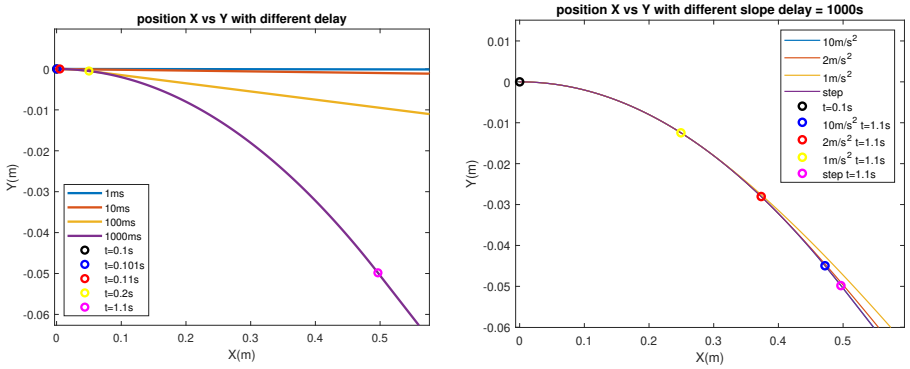
The experiment is operated to verify the synchronicity. If the delay exists, the motion is provided in Figure 6.8. In the ideal case, two motors operating synchronously, the trajectory would be a horizontal line. In Figure 6.8(a), two motors run at 1.0 rev/s with different delay is plotted. we can find that the larger the delay is, the trajectory of motion will have a larger offset. Motors will move linearly with offset direction after two motors accelerate to the command speed, which is the time points marked in this figure.

In the case of motors running at the different acceleration with the same delay(1000 ms), the trajectory is shown in Figure 6.8(b). The motion with 1.0 rev/s^2, 2.0 rev/s^2, and 10.0 rev/s^2 compare with the ideal case(accelerate with step function) and we can find that larger acceleration is, the motion will have the larger offset because the displacement is larger during acceleration. To compensate the offset, the series of experiment is operated.

doi:10.6342/NTU202403297

By Equation (4.7), the result of compensation is shown in Figure 6.9. It displays that the delay caused by network communication can be compensated by linear method. In different delay situation, the delay can be compensate obviously.

The real-time performance under multiple axes is shown in Table 6.3 and Figure 6.10. Table 6.3 summarizes the maximum and minimum values of latency for mutiple axes. We can remark that the implementation on the real-time motion control platform with PLC motion controller can achieve great real-time performance while maintaining low CPU usages. Besides, it should be noted that the stability and performance of our proposal on the real-time motion control platform have the finer timer resolution of the Xenomai patch (1 μs). In conclusion, these measurement results prove that the design can fulfill the basic functional requirements under single axis condition.



(a) Different delay **(b) Different acceleration**

Figure 6.8: Timing diagram for the latency in two motor situation

Table 6.3: Minimum and maximum latency in two axes situation

	average	std
Min. latency(μs)	1.6401	0.0120
Max. latency(μs)	713.859	137.50
CPU ussage(%)	29.6	0.4

doi:10.6342/NTU202403297

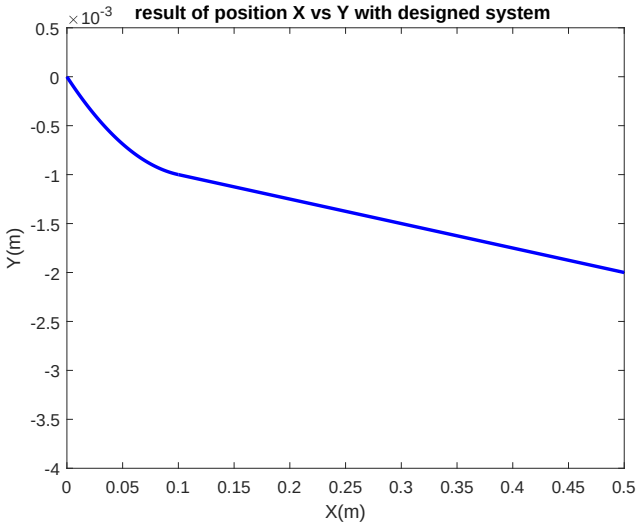
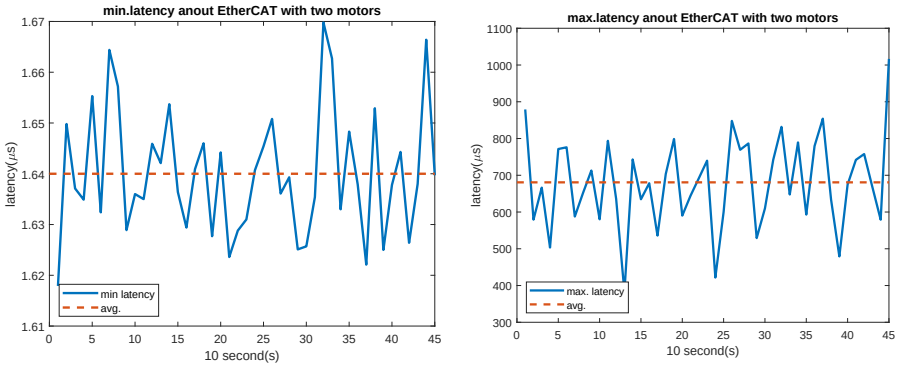


Figure 6.9: The motion performance of two motors



(a) Minimum latency **(b) Maximum latency**

Figure 6.10: Timing diagram for the minimum and maximum latency in two axes situation

doi:10.6342/NTU202403297

To show the situation over two axes, simulation is conducted. Tasks are created with 1 ms load to simulate the case with one axis. With the quantity of the axis increasing, the tasks also increase to simulate the situation of multiple axes. The result is shown in Table 6.4. It shows that both minimum and maximum latencies increase with the number of axes in the system. The increase in minimum latency is relatively small and stable, indicating that even with more axes, the fastest operations do not significantly slow down. In contrast, the increase in maximum latency is more pronounced, reflecting that more complex systems may experience higher peak latencies. These findings suggest that while the complexity added by more axes does impact latency, the effect is more pronounced at the upper bounds of latency.

Table 6.4: Minimum and maximum latency by simulation

1 axis	average	std
Min. latency(μs)	1.2583	0.0095
Max. latency(μs)	567.92	130.55
2 axes	average	std
Min. latency(μs)	1.8374	0.0098
Max. latency(μs)	611.21	95.34
3 axes	average	std
Min. latency(μs)	1.742	0.0147
Max. latency(μs)	674.12	154.28
10 axes	average	std
Min. latency(μs)	1.877	0.0230
Max. latency(μs)	709.78	120.51

doi:10.6342/NTU202403297

6.3 Simulation of Semaphore-based Real-time Scheduling

6.3.1 Simulation Setup

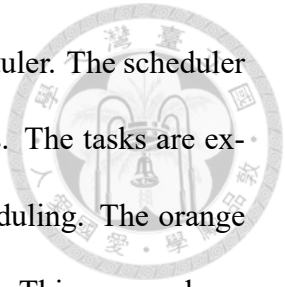
In this section, simulation of the scheduling method is done and compares with the situation with direct FCFS. The tools used in this paper is Matlab. First, tasks is defined with their attributes such as execution time, period, deadline, and priority. Next, a timeline to simulate task execution and track their states are created, incorporating mechanisms to handle task arrivals, preemptions, and context switches. In the simulation, 1000 times tests are done to get the average value. It simulates the semaphore-based real-time scheduler where tasks are executed based on their predefined periods. The scheduler ensures that tasks execute at specific intervals, with the synchronization handled by a semaphore.

6.3.2 Simulation Results and Analysis

The real-time scheduling of events must ensure that the advancement of discrete event times aligns with the system time. For synchronization, each simulation task is required to execute at a fixed update cycle or frequency, with all tasks' update cycles being integer multiples of the minimum task cycle and excluding any random event scheduling code. This minimum period is known as a small frame, and the least common multiple of all cycle multiples, multiplied by the minimum period, is called a large frame. To maintain the real-time performance of each frame, the simulation time is synchronously checked by the real-time clock interrupt before each small frame.

The result is displayed in Figure 6.11 and Table 6.5. Assume that the period of each thread is the same as the deadline. From the results, the blue circles represent the execu-

doi:10.6342/NTU202403297



tion times for tasks scheduled by the semaphore-based real-time scheduler. The scheduler ensures that each task runs at regular intervals based on their periods. The tasks are executed smoothly without significant delays, indicating effective scheduling. The orange diamonds show the execution times for tasks under FCFS scheduling. This approach results in slightly less regular execution patterns compared to the semaphore-based real-time scheduler, with some deviations in task execution times, particularly for tasks with longer periods. The red squares represent the execution times for tasks scheduled using RMS. RMS tends to favor tasks with shorter periods, resulting in more consistent execution for such tasks. However, tasks with longer periods show more variability in their execution times.

From Table 6.5, it shows that in the same simulation time, the semaphore-based real-time scheduler can maintain the most consistent execution times across all threads. Thus, if the number of devices increases, semaphore-based scheduling let devices with lower priority be less affected. The variability in execution count from FCFS and RMS indicates potential delays and less efficient handling of tasks with longer periods.

The semaphore-based real-time scheduler demonstrates superior performance in maintaining consistent execution intervals and handling tasks with varying periods effectively. It ensures real-time performance and minimizes execution delays compared to FCFS and RMS. FCFS shows irregularities in task execution times, particularly for tasks with longer periods. RMS, on the other hand, favors shorter period tasks, leading to underutilization of longer period tasks. The semaphore-based real-time scheduler thus provides a balanced and efficient approach for real-time multi-thread scheduling.

doi:10.6342/NTU202403297

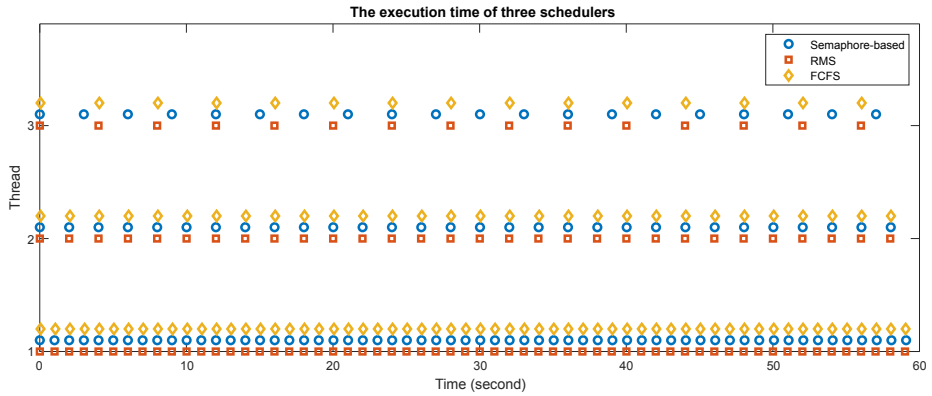


Figure 6.11: The execution time of three scheduling methods

Table 6.5: The number of execution times for three scheduling methods

	Thread 1	Thread 2	Thread 3
Semaphore-based real-time scheduler	600	300	202
FIFO	600	295	158
RMS	554	297	151

doi:10.6342/NTU202403297

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In conclusion, this research demonstrated the feasibility of real-time motion control through EtherCAT. Through experiments, it is found that the relative error of this motion control system gradually decreases with the increase of speed in the case of single axis. At 30 rev/s, the error between speed and command is only 1.47%. In terms of real-time performance, the system's minimum latency is only 1 microsecond and does not occupy too many CPU resources. In the multi-axis case, motion synchronization is slightly offset by start-up times, but the offset is small and predictable. If the number of axes increases, the maximum delay will increase. However, the minimum latency will still remain on the order of 1 microsecond, with little impact.

At the same time, the semaphore-based scheduling memethod effectively manages resource allocation in a multi-tasking environment. Compared with traditional real-time control methods, higher-priority devices can maintain similar execution times, while lower-priority devices increase their execution times by 32% with three devices, ensuring the real-time execution of each task.

This study proposes a new motion control and scheduling solution. This combination provides an open source and free method for real-time motion control and still has excellent real-time performance, promoting the continued development of industrial automation technology.

doi:10.6342/NTU202403297

7.2 Future Works

In the future, the proposed directions for future research include:

1. Enhanced Algorithm Development : Incorporating machine learning techniques to predict and adapt to changing conditions could further improve system efficiency and reliability.

2. Add experimental verification: To verify the scheduling method, more experimental method can be adopted. For example, the additional delay can be added deliberately to observe the effect of delay.

3. Stability calculation: Check the stability of the Semaphore-based real-time scheduling method.

4. Integration with IoT and Industry 4.0 Technologies: Expand the framework to seamlessly integrate with broader Internet of Things (IoT) networks and Industry 4.0 technologies.

By pursuing these future research directions, the potential of real-time motion control and scheduling based on Intel Edge Controls can be fully realized, leading to significant advancements in industrial automation and productivity.

doi:10.6342/NTU202403297

References

[1: Rostan, Stubbs, and Dzilno 2010]

M. Rostan, J. E. Stubbs, and D. Dzilno, "EtherCAT enabled advanced control architecture," in *IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, Jul. 2010, pp. 39–44. DOI: 10.1109/ASMC.2010.5551414.

[2: Potra and Sebestyen 2006]

S. Potra and G. Sebestyen, "EtherCAT Protocol Implementation Issues on an Embedded Linux Platform," in *IEEE International Conference on Automation, Quality and Testing, Robotics*, vol. 1, May 2006, pp. 420–425. DOI: 10.1109/AQTR.2006.254573.

[3: NXP 2022] NXP, *EtherCAT Master Solution: 9-axis Robot Arm Control*, https://www.nxp.com/video/ethercat-leader-solution-9-axis-robot-arm-control:ETHERCAT-SOLUTION-9-AXIS-ROBOT-ARM-CONTROL-VID, 2022.

[4: NEXCOM 2014] NEXCOM, *EtherCAT CNC Controller NControl20 Helps Developers Flex Design Muscles*, https://www.nexcom.com/news/Detail/ethercat-cnc-controller-ncontrol20-helps-developers-flex-design-muscles, 2014.

[5: Axiomtek 2016] Axiomtek, *EtherCAT Master Controller in Factory Automation*, https://www.axiomtek.com.tw/ArticlePageView.aspx?ItemId=331&t=47, 2016.

[6: NEXCOM 2016] NEXCOM, *NEXCOM EtherCAT Master Controllers Accelerate Real-Time Control to Push Production to Full Throttle*, https://www.nexcom.com.tw/news/Detail/nexcom-ethercat-master-controllers-accelerate-real-time-control-to-push-production-to-full-throttle, 2016.

[7: Moon *et al.* 2009]

Y. S. Moon, N. Y. Ko, K.-S. Lee, Y.-C. Bae, and J.-K. Park, "Real-time EtherCAT Master Implementation on Xenomai for a Robot System," *International Journal of Fuzzy Logic and Intelligent Systems*, vol. 9, Sep. 2009. DOI: 10.5391/IJFIS.2009.9.3.244.

doi:10.6342/NTU202403297

[8: Wang *et al.* 2010]

L. Wang, J. Qi, H. Jia, and B. Fang, "The construction of soft servo networked motion control system based on EtherCAT," in *2nd Conference on Environmental Science and Information Application Technology*, vol. 3, Jul. 2010, pp. 356–358. DOI: 10.1109/ESIAT.2010.5568340.

[9: Zhou *et al.* 2015]

C.-C. Zhou, J.-M. Xu, X. Jin, J. Mao, and L.-T. Xu, "Design of servo drive slaves based on EtherCAT," in *27th Chinese Control and Decision Conference (CCDC)*, May 2015, pp. 5999–6004. DOI: 10.1109/CCDC.2015.7161885.

[10: Delgado and Choi 2017]

R. Delgado and B. Choi, "Real-time Servo Control using EtherCAT Master on Real-time Embedded Linux Extensions," *International Journal of Applied Engineering Research*, vol. 12, pp. 1179–1185, Nov. 2017.

[11: Zhang *et al.* 2019]

G. Zhang, Z. Li, F. Ni, and H. Liu, "A Real-time Robot Control Framework Using ROS Control for 7-DoF Light-weight Robot," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, Jul. 2019, pp. 1574–1579. DOI: 10.1109/AIM.2019.8868488.

[12: Davis and Burns 2011]

R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys*, vol. 43, no. 4, 35:1–35:44, 2011, ISSN: 0360-0300. DOI: 10.1145/1978802.1978814.

[13: Liu and Layland 1973]

C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973, ISSN: 0004-5411. DOI: 10.1145/321738.321743.

[14: Lehoczky, Sha, and Ding 1989]

J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Ex-

doi:10.6342/NTU202403297

act characterization and average case behavior," in *Real-Time Systems Symposium*, Feb. 1989, pp. 166–171. DOI: 10.1109/REAL.1989.63567.

[15: Short 2010]

M. Short, "Improved Task Management Techniques for Enforcing EDF Scheduling on Recurring Tasks," in *16th IEEE Real-Time and Embedded Technology and Applications Symposium*, Apr. 2010, pp. 56–65. DOI: 10.1109/RTAS.2010.22.

[16: Burns 1991]

A. Burns, "Scheduling hard real-time systems: A review," *Software Engineering Journal*, vol. 6, no. 3, pp. 116–128, May 1991, ISSN: 2053-910X. DOI: 10.1049/sej.1991.0015.

[17: Barbalace *et al.* 2007]

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," in *15th IEEE-NPSS Real-Time Conference*, Apr. 2007, pp. 1–5. DOI: 10.1109/RTC.2007.4382787.

[18: Reghenzani, Massari, and Fornaciari 2019]

F. Reghenzani, G. Massari, and W. Fornaciari, "The Real-Time Linux Kernel: A Survey on PREEMPT_RT," *ACM Computing Surveys*, vol. 52, no. 1, 18:1–18:36, 2019, ISSN: 0360-0300. DOI: 10.1145/3297714.

[19: RTAI home page] RTAI home page, *RTAI*, https://www.rtai.org/www.rtai.org/.

[20: Xenomai home page] Xenomai home page, *Xenomai home page*, https://www.xenomai.org/.

[21: EtherCAT technology group] EtherCAT technology group, *EtherCAT Technology Group*, https://www.ethercat.org/default.htm.

[22: Nguyen and Jeon 2016]

V. Q. Nguyen and J. W. Jeon, "EtherCAT network latency analysis," in *International Conference on Computing, Communication and Automation (ICCCA)*, Apr. 2016, pp. 432–436. DOI: 10.1109/CCAA.2016.7813815.

doi:10.6342/NTU202403297

[23: Shen, Li, and Luo 2020]

H. Shen, P. Li, and X. Luo, "Synchronous Multi-axis Motion Control Based on Modified EtherCAT Distributed Clock," in *Chinese Automation Congress (CAC)*, Jan. 2020, pp. 3674–3678. DOI: 10.1109/CAC51589.2020.9327605.

[24: John and Tiegelkamp 2010]

K. H. John and M. Tiegelkamp, *IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*. Berlin, Heidelberg: Springer, 2010, ISBN: 978-3-642-12014-5 978-3-642-12015-2. DOI: 10.1007/978-3-642-12015-2.

[25: Song *et al.* 2007]

S. Song, X. Lin, Q. Huang, and C. Wang, "An Embedded SoftLogic Control System Based on S3C44BOX and IEC 61131-3 Standard," in *IEEE International Conference on Control and Automation*, May 2007, pp. 2060–2064. DOI: 10.1109/ICCA.2007.4376723.

[26: de Sousa and Carvalho 2003]

M. de Sousa and A. Carvalho, "An IEC 61131-3 compiler for the MatPLC," in *IEEE Conference on Emerging Technologies and Factory Automation. Proceedings*, vol. 1, Sep. 2003, 485–490 vol.1. DOI: 10.1109/ETFA.2003.1247746.

[27: Tisserant, Bessard, and de Sousa 2007]

E. Tisserant, L. Bessard, and M. de Sousa, "An Open Source IEC 61131-3 Integrated Development Environment," in *5th IEEE International Conference on Industrial Informatics*, vol. 1, Jun. 2007, pp. 183–187. DOI: 10.1109/INDIN.2007.4384753.

[28: Prahofer *et al.* 2011]

H. Prahofer, R. Schatz, C. Wirth, and H. Mossenbock, "A Comprehensive Solution for Deterministic Replay Debugging of SoftPLC Applications," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 641–651, Jan. 2011, ISSN: 1941-0050. DOI: 10.1109/TII.2011.2166768.

doi:10.6342/NTU202403297

[29: Eldijk 2018] V. Eldijk, *PLC Motion Control*, https://plcopen.org/technical-activities/motion-control, Text, Apr. 2018.

[30: Lin *et al.* 2018] H.-Y. Lin, I.-W. Cheng, Y.-H. Huang, H.-C. Liu, Y.-H. Lin, C.-Y. Yang, H. Samani, and Y.-J. Chen, "Applying Socket on Connecting EtherCAT and OpenPLC," in *International Conference on System Science and Engineering (ICSSE)*, Jun. 2018, pp. 1–5. DOI: 10.1109/ICSSE.2018.8520168.

[31: Downey 2009] A. B. Downey, *The Little Book of Semaphores*. Createspace Independent Pub, Mar. 2009.

[32: Andersson and Tovar 2006] B. Andersson and E. Tovar, "Multiprocessor Scheduling with Few Preemptions," in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*, Aug. 2006, pp. 322–334. DOI: 10.1109/RTCSA.2006.45.

[33: Sha, Rajkumar, and Lehoczky 1990] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, Sep. 1990, ISSN: 1557-9956. DOI: 10.1109/12.57058.

[34: EIA 2010] EIA, *Electronic Industries Alliance (EIA)*, https://web.archive.org/web/20100301055711/http://www.eia.org/, Mar. 2010.

[35: Organization] M. Organization, *The Modbus Organization*, https://www.modbus.org/.

[36: Alliance] E. Alliance, *Ethernet Alliance*, https://ethernetalliance.org/.

[37: Delgado *et al.* 2016] R. Delgado, S.-Y. Kim, B.-J. You, and B.-W. Choi, "An EtherCAT-based real-time motion control system in mobile robot application," in *13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Aug. 2016, pp. 710–715. DOI: 10.1109/URAI.2016.7734098.

doi:10.6342/NTU202403297

[38: Zhou *et al.* 2019]

Y. Zhou, Z. Li, Y. Zhang, F. Ni, Y. Sun, and H. Liu, "The Real-time Control Framework for a Modular Snake Robot," in *4th International Conference on Robotics and Automation Engineering (ICRAE)*, Jan. 2019, pp. 168–172. DOI: 10.1109/ICRAE48301.2019.9043797.

[39: Yoon, Song, and Lee 2009]

H. Yoon, J. Song, and J. Lee, "Real-Time Performance Analysis in Linux-Based Robotic Systems," Jan. 2009.

[40: Delgado and Choi 2017]

R. Delgado and B. W. Choi, "On the in-controller performance of an open source EtherCAT master using open platforms," in *14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Jun. 2017, pp. 744–748. DOI: 10.1109/URAI.2017.7992815.

[41: Hu *et al.* 2022]

F. Hu, F. Qu, J. Li, and H. Zhao, "Real-time research based on ARM-based robot EtherCAT master system," in *International Symposium on Control Engineering and Robotics (ISCER)*, Feb. 2022, pp. 12–19. DOI: 10.1109/ISCER55570.2022.00009.

[42: Wang and Lin 1999]

Y.-C. Wang and K.-J. Lin, "Implementing a general real-time scheduling framework in the RED-Linux real-time kernel," in *20th IEEE Real-Time Systems Symposium*, Feb. 1999, pp. 246–255. DOI: 10.1109/REAL.1999.818850.

[43: Wang and Gombolay 2020]

Z. Wang and M. Gombolay, "Learning Scheduling Policies for Multi-Robot Coordination With Graph Attention Networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4509–4516, Jul. 2020, ISSN: 2377-3766. DOI: 10.1109/LRA.2020.3002198.

[44: Zhou and Li 2021]

N. Zhou and D. Li, "Cyber–Physical Codesign of Field-Level Reconfigurations

doi:10.6342/NTU202403297

in Networked Motion Controllers," *IEEE/ASME Transactions on Mechatronics*, vol. 26, no. 4, pp. 2092–2103, Aug. 2021, ISSN: 1941-014X. DOI: 10.1109/TMECH.2020.3032571.

[45: Hamdaoui and Ramanathan 1995]

M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1443–1451, Feb. 1995, ISSN: 1557-9956. DOI: 10.1109/12.477249.

[46: Tran *et al.* 2013]

H.-D. Tran, Z.-H. Guan, X.-K. Dang, X.-M. Cheng, and F.-S. Yuan, "A normalized PID controller in networked control systems with varying time delays," *ISA Transactions*, vol. 52, no. 5, pp. 592–599, Sep. 2013, ISSN: 0019-0578. DOI: 10.1016/j.isatra.2013.05.005.

[47: Sung *et al.* 2011]

M. Sung, K. Kim, H.-W. Jin, and T. Kim, "An EtherCAT-based motor drive for high precision motion systems," in *9th IEEE International Conference on Industrial Informatics*, Jul. 2011, pp. 163–168. DOI: 10.1109/INDIN.2011.6034856.

[48: Liang *et al.* 2019]

Z. Liang, Y. Guo, Y. Yang, and G. Chen, "Distribution network control system scheduling strategy," in *IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, Mar. 2019, pp. 1424–1428. DOI: 10.1109/ITNEC.2019.8729286.

[49: Branicky, Phillips, and Zhang 2002]

M. Branicky, S. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *41st IEEE Conference on Decision and Control*, vol. 2, Feb. 2002, 1211–1217 vol.2. DOI: 10.1109/CDC.2002.1184679.

[50: Zhang *et al.* 2018]

X. Zhang, G. Li, P. Wang, z. Yang, J. Lu, and J. Zhang, "Design and Implementation of Real-Time DEVS Event Scheduling Algorithm," in *3rd International Con-*

doi:10.6342/NTU202403297

ference on Mechanical, Control and Computer Engineering (ICMCCE), Sep. 2018, pp. 163–168. DOI: 10.1109/ICMCCE.2018.00041.

[51: Industrial *et al.*] W. I. E. C. for Industrial, b. c. c. m. l. n. b. D. a software reference platform with compatible hardware, T. S. I. R.-T. D. Compute, S.-B. Connectivity, and I.-l. management with operational technology–like predictability, *Intel® Edge Controls for Industrial*, https://www.intel.com/content/www/us/en/internet-of-things/industrial-iot/edge-controls-industrial.html.

[52: Trihedral] Trihedral, *VTScada by Trihedral*, https://www.vtscada.com/.

doi:10.6342/NTU202403297