CODEN:LUTEDX/(TEIE-5523)/1-48(2024)

# Scalability and Predictability Model of PLC Systems

**Max Fornander**

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Lund University

Faculty of Engineering

Division of Industrial Electrical Engineering and Automation

# Scalability and Predictability Model of PLC Systems

**Academic supervisor:**

Ulf Jeppsson

**Industrial supervisors:**

Ulf Thomsen Plym

Peter Stoltenberg

**Examiner:**

Gunnar Lindstedt

Thesis written by:

Max Fornander

October 2024

# Abstract

This master's thesis is commissioned by SAAB Group with the purpose of developing a model which can be used to estimate the margin for expansion of devices in PLC systems. Although commercial PLC systems are hard real-time systems, they have inherited unpredictable characteristics due to off-the-shelves components. This makes it difficult to measure the impact on performance when a new device is connected to the system.

By pressuring the CPU using a computationally heavy control program, the scheduler is forced to prioritize tasks associated with the addition of a device. This may increase the accuracy of measuring the impact of an increasing number of devices on CPU performance. The commercial PLC system and all devices used are manufactured by B&R, a member of ABB group providing solutions within industrial automation.

The method consists of measuring idle and cyclic CPU usage (%) before and after a device is connected to the system. This is done using Profiler, a profiling tool provided by B&R. The resulting models, using linear approximation and ridge regression, is capable of predicting the impact on idle and cyclic CPU usage (%) when adding a device to a test setup of similar size to previously measured systems. Although the gathered quantity of data samples can be considered low, the predictions regarding idle load (%) was accurate.

Furthermore, the industrial Ethernet protocol (Powerlink) used in the B&R system is evaluated. Specifically to detect circumstances which might impact the CPU load and the number of devices existing within a single network. Program optimization is briefly investigated and two programming conventions are identified as important for maintaining a large number of devices.

# Sammanfattning

Detta examensarbete är gjort i samarbete med Saab Group med syftet att utveckla en modell för att estimera den marginal som finns gällande antalet enheter i ett PLC system. Även om kommersiella PLC system är hårda realtidssystem, så innehar de till viss del fortfarande oprediktterbara beteenden. Detta medför en viss svårighet med att mäta inverkan på systemets prestanda när en ny enhet kopplas in i systemet.

Genom att öka arbetslasten på processorn med hjälp av ett beräkningsmässigt krävande kontrollprogram, så tvingas systemet att prioritera dess resurser kopplade till den nya enheten. Hypotesen är att detta leder till en förbättrad mätmetod, som ger ett tydligare resultat för hur mycket en enhet faktiskt påverkar processorlasten. PLC systemet och all kringutrustning som använts tillverkas av företaget B&R, som ingår i ABB-koncernen och är specialiserat på lösningar inom industriell automation.

Metoden mäter skillnaden i tomgång och cyklisk last (%) före och efter att en ny enhet kopplas in i systemet. Processorlast mäts med Profiler, ett profileringsverktyg som tillhandahålls av B&R. Med hjälp av resultatet utvecklas två modeller, varav ena baseras på linjär approximering och andra ridge regression. Modellerna predikterar påverkan på processorlasten då nya enheter kopplas in i testsystem av liknande storlek, trots en låg mängd insamlade mätvärden. Specifikt sågs goda prediktioner gällande tomgångslasten.

Därefter undersöks det industriella kommunikationsprotokoll som finns i PLC-systemet. Dels för att hitta eventuella begränsningar gällande antalet inkopplade enheter, dels för att se hur processorn påverkas av större nätverk. Till sist undersöks kodningskonventioner som kan förbättra prestandan för system med många enheter.

**Acknowledgements**

Jag vill börja med att tacka Joakim på Saab som gav mig möjligheten att genomföra detta examensarbete. Det har varit mycket utvecklande att själv få forma arbetet och hitta egna svar, samt få en inblick i hur en bra arbetsplats med god stämning och glada kollegor ska se ut!

Ett väldigt stort tack till både Ulf och Peter på Saab som hjälpt mig att komma igång och ro arbetet i hamn. Med stort tålamod och pedagogisk förmåga har de besvarat alla frågeställningar under arbetets gång och samtidigt bidragit till att man känt sig välkommen på företaget. Stort tack!

Till handledare Ulf och examinator Gunnar på LTH vill jag även rikta min stora tacksamhet. Trots sina fulla agendor har de bidragit med sin tid, kunskap och känsla för akademiska detaljer.

Sist men inte minst vill jag tacka min familj (som under arbetets gång utökats med en fantastisk liten individ) för deras tålamod, stöd och hejarop!

# Contents

# Terminology

A list of terms and abbreviations used in the thesis.

**Terms**

- **Channel** - Each module has a varying number of channels through which digital or analog signals can be received or sent.
- **Device** - Refers to both modules, nodes, or any other component connected to the PLC directly or indirectly, which could potentially impact performance.
- **Module** - A single I/O component attached directly either to a node or serially to the PLC itself.
- **Node** - A connection point within the network onto which modules are attached.
- **Powerlink** - An industrial Internet protocol.
- **Task** - A set of instructions executed by the CPU.
- **Wireshark** - Tool used to observe/capture network communication.

**Acronyms**

- **ARP** - Adress resolution protocol, an internet communication protocol.
- **API** - Application programming interface, offers services between software applications.
- **CPU** - Central Processing Unit, as in computer hardware.
- **FB** - Function block, used in PLC programming.
- **FBD** - Function block diagram, used in PLC programming.
- **FC** - Function, used in PLC programming.
- **IEC** - International Electrotechnical Commission, an international standards organization.
- **IL** - Instruction list, used in PLC programming.
- **ISR** - Interrupt service routine, part of operating system.
- **I/O** - Input or output.
- **I/O Pair** - One input module cross-coupled to an output module.
- **LD** - Ladder diagram, used in PLC programming.
- **LLDP** - Link-layer discovery protocol, internet communication protocol.
- **PLC** - Programmable Logic Controller.
- **PLK** - Powerlink, B&R's industrial communication protocol.
- **PMU** - Performance Measurement Unit, part of Intel's processors.
- **POU** - Program organization unit, used in PLC programming.
- **PRG** - Program,, used in PLC programming.
- **RAM** - Random Access Memory, as in computer hardware.
- **RTOS** - Real-Time Operating System.
- **SFC** - Sequential Function Chart, used in PLC programming.
- **ST** - Structured text, used in PLC programming.
- **TCB** - Task Control Block, contains task related information.
- **UDP** - User Datagram Protocol, an Internet protocol.
- **UDT** - User Defined Data Type, used in PLC programming.

# I Introduction

This chapter serves as a brief overview of the thesis by first introducing PLC systems and providing the reader with the necessary background information to delve into the problem statement. The final section describes the objectives and limitations regarding developing a model for scalability and predictability.

## I.I Background

A Programmable Logic Controller (PLC) sets itself apart from other types of controllers used in industrial automation by prioritizing robustness and real-time signal processing. Set in context, the PLC reads inputs such as signals from sensors, processes these values according to a user defined control program, and finally updates outputs such as signals to actuators. The described sequence of operations, known as the scan cycle, occurs periodically with a fixed periodicity.

The PLC is referred to as a hard real-time system, meaning any time exceeding the scan cycle period results in system failure. This makes the PLC ideal for controlling processes where timing and reliability is vital. Every hardware component in the PLC is chosen to mitigate any risk of delays and to decrease response time while also ensuring operating for extended periods of time in harsh industrial environments. The downside with this feature is the resulting limitations on, e.g., number of CPU cores, or memory size, which decreases the PLCs computational capabilities.

Today, most PLC systems are modular, meaning they can be expanded with additional devices which enhance the system's capabilities such as *modules* for increased I/O functionality. Modules can in turn be attached to the system through devices known as *nodes*. A PLC system's ability to expand the number of connected devices while maintaining system performance is referred to as its *scalability*. A system's scalability could theoretically be measured by evaluating how much a device impacts system performance indicators such as CPU usage (%).

Adding a device affects multiple layers of the system, such as activating additional tasks, introducing delays through sampling of signals and then sending/receiving data through a network of nodes. A system's *predictability* refers to how consistently and reliably the system behaves in response to changes. Every PLC type reacts differently to the addition of a device, the performance might either gradually or suddenly change. This project investigates whether it is possible to develop a model describing a PLC system's scalability and predictability. The intended use of the model is to predict a margin for how much more the system can be expanded with additional devices. The thesis is commissioned by SAAB Group who provides solutions within military defence and civil security.

Research in this field presents several challenges. To begin with, the thesis focuses on commercially available systems. Those often use proprietary hardware and software solutions, resulting in limited transparency regarding their design and performance. Only studies tied to the evaluation of real-time systems in general was found through the literature study. In the rare case that the performance of PLC systems was evaluated, it was solely related to novelty hardware architectures, and not commercial systems.

The chosen work method to evaluate the impact on performance by a single device is to measure CPU usage (%) before and after a new device is connected. All while a computationally heavy control program is executed on the PLC to further isolate the impact of said device. Differences between analog and digital modules will first be investigated. Followed by different configurations of nodes and modules. Since there exist numerous node and module configurations, two different cases will be investigated. At first using a "balanced" setup, with an equal number of modules per node, followed by an "unbalanced" setup, with a different number of modules per node.

Another aspect regarding scalability and predictability is the impact on the network from adding a device. This is investigated by simulating heavy data traffic using packet flooding, while monitoring CPU usage (%) and irregularities in both the Ethernet and industrial Ethernet protocol network. Finally, PLC performance is tied to the optimization level of the program code. Since program optimization regarding PLC systems is a vast research field, only very basic aspects regarding code optimization for maintaining an expanding system will be investigated.

## I.II Objectives

The thesis was commissioned by submarine manufacturer SAAB Kockums, part of SAAB Group. Needless to say, storage is a scarce commodity onboard submarines. This is why great care has to be taken into planning the PLC system and any potential future upgrades. Hence, SAAB Kockums seeks to develop a model which predicts the performance of a PLC system with certain numbers (and types) of connected devices. The model should provide a marginal for the number of devices that can be connected in the future. The model should help in identifying factors in the system which inhibit scalability and predictability. The thesis will at the end give an answer to the following problem statement: Is it possible to create an applicable model for scalability and predictability for a commercially available PLC system?

The following objectives were defined at the start of the project regarding developing a model for scalability and predictability:

- Establish the relationship between impact on system performance and the number/type of device(s) for a commercially available PLC system.
- Investigate limitations surrounding the network regarding number of devices.
- Investigate programming conventions which could increase the number of devices.

## I.III Limitations

The PLC and all its components (nodes and modules) used for testing is built by the manufacturer B&R. This limits general application of the model since there are many design differences in hardware and software between manufacturers, and even within product line-ups themselves. To mitigate the risk of the thesis turning into an evaluation of B&R's systems, the chosen measurements (idle and cyclic CPU usage (%)) can be measured using common profiling tools, further improving the usability of the chosen method and model.

The thesis was written by one engineering student with the aid of both industrial supervisors at SAAB Group and academic supervisors at LTH. The time frame for the thesis spanned from October 2023 to June 2024. During the period quite some time was spent familiarizing with both the PLC hardware and accompanying software suite. Furthermore, some time was set aside for security clearance to visit the offices holding the testing equipment.

Another hindrance was the lack of academic studies found during the literature study related to scalability and/or predictability models of PLC systems. As such, additional time was spent devising both a working methodology for reliably gathering useful measurements, and finally an applicable model. As previously mentioned, there are also generally limited information regarding commercial systems since manufacturers rarely disclose performance numbers.

# II Literature review

This chapter contains general information regarding PLC systems as well as information regarding academic foundation.

## II.I General information regarding PLC systems

The programmable logic controller was first invented by two competing companies, Allen Bradley and Bedford Associates, who both responded to General Motors wishes to replace mechanical relay sequencing used in the automobile industry [1]. While the mechanical relays provided real-time communication capabilities, they did not allow for quick changes to the internal logic. Any mistakes made along the development of the circuitry resulted in increased development costs and production delays. During the 1960's the computer was very fragile and required dust-free environments, opposite of the conditions in which the mechanical relays were operating. Furthermore, the early computers were large, slow, and expensive [1].

Thus, a new type of computer had to be developed. The computer used in the industrial setting had to be robust. It should handle sudden power supply changes, behave predictably during abrupt interrupts, and withstand environmental complications such as temperature or signal interference due to radiation. The computer would also be remained untouched for long periods of time after installation, putting high priority on predictability [1]. The new computer should also be easily programmed by the same workers who developed the sequential relays. Hence, the resulting computer had a very different keyboard-layout compared to today, see Figure 1.

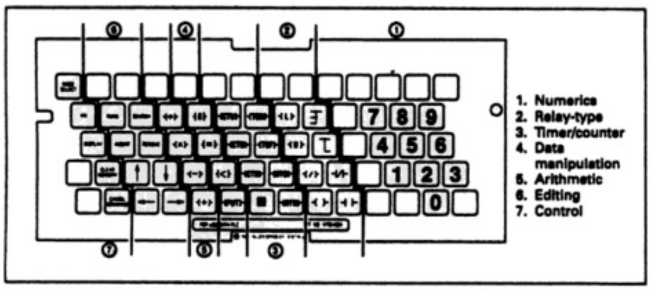


Figure 1: The PLC developed by Allen-Bradley used the same symbols as in conventional sequential relays.

The modern PLC system has stood the test of time because it fills a very specific role. Today, there are more than twenty global manufacturers of PLC systems, each with their own design, software suite, and specialized applications [2]. A comparison between the modern [3] and early [4] Allen-Bradley PLC system shows that the very first PLCs had designs similar to those of today, see Figure 2-3. However, the functionality of PLCs has drastically evolved since the 1960's. While PLCs initially emerged in the automobile industry, they are now used across nearly all sectors of society, e.g., energy, healthcare, and infrastructure.

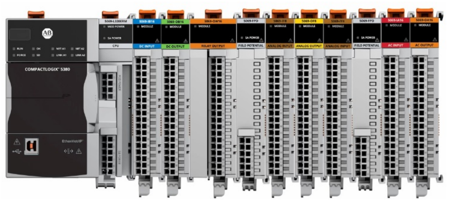


Figure 2: Allen-Bradley's CompactLogix 5380, launched in 2016.

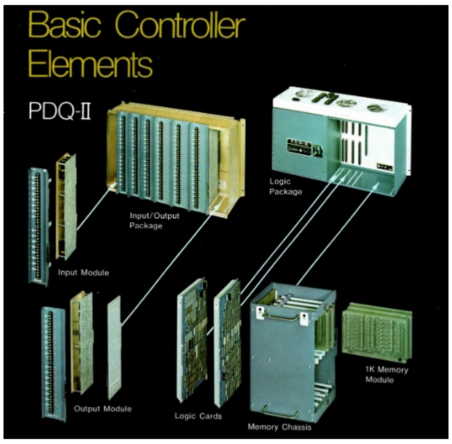


Figure 3: Allen-Bradley's first commercial PLC, the PDQ-II, launched in 1969.

## II.II B&R's system

The system used in this thesis is manufactured by B&R, a member of the ABB-group specializing in solutions within industrial automation such as safety PLC's for processing lines or machines automated with B&R controllers [5]. The PLC unit used is the CP3586 (see Figure 4) containing an Intel Atom clocked at 1.6 MHz. Modules can be connected directly to the PLC using a X2X-link connection, or through nodes using B&R's proprietary industrial communication protocol known as Powerlink.

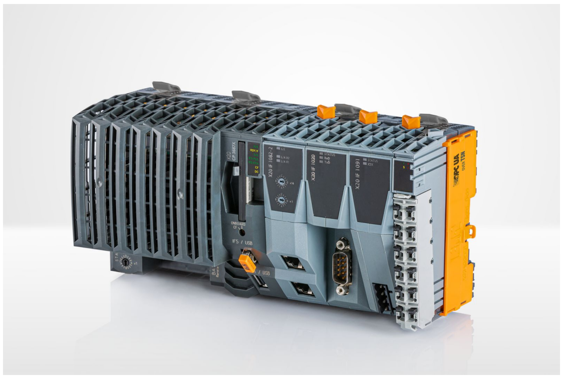


Figure 4: The PLC used in this project; the CP3586, manufactured by B&R. Here with additional modules attached serially using the X2X-link connection.

## II.III Academic foundation

Studies found regarding performance of commercially available PLCs in general are scarce. While some studies are found in which an attempt at standardizing benchmarks between manufacturers where found (e.g., [6]), these types of studies does not focus on translating the measured CPU performance into an applicable number of devices. Other studies focused on hardware architectures (e.g.,[7]), which evaluate the impact on scan cycle performance in PLC systems from using multi-core CPUs. However, little is said about how these improvements can affect scalability. Instead, the literature review focused on aspects within hardware (e.g., [8]) and software (e.g., [9]) which generally impacts the performance of hard real-time systems, not always specifically PLCs.

# III Theory

Here technical aspects related to PLC systems, model development, program optimization, and network evaluation are discussed.

## III.I The scan cycle

The *Programmable Logic Controller (PLC)* is commonly used within industrial automation to control physical processes which require precise timing. To contextualize, a PLC could be used to regulate the temperature by controlling a heating element. The process is controlled in three sequential steps. First by reading inputs, in this case from a temperature sensor, and storing the values in a data structure known as an *input image table* [10]. A user defined control program processes the input image table and sequentially executes every instruction. The corresponding output variables from the control program is stored in an *output image table* and sent to the actuator connected to the heating element. These three steps are known as the *input phase, program phase,* and *output phase*, respectively, and are together referred to as the *scan cycle* (see Figure 5). [10]

The scan cycle was standardized by the organization *International Electrotechnical Commission (IEC)*, it is periodic and has a fixed periodicity [10]. The defining feature of the PLC system is the fact that it is referred to as a hard real-time system, meaning any time exceeding the scan period results in system failure known as *time cycle violation.* The scan cycle sometimes has an additional intermittent phase known as *housekeeping*, which differs between manufacturers. Housekeeping might consist of internal checks on memory, system operation, diagnostics, as well as communication requests generated between other hosts or the control program itself [10].

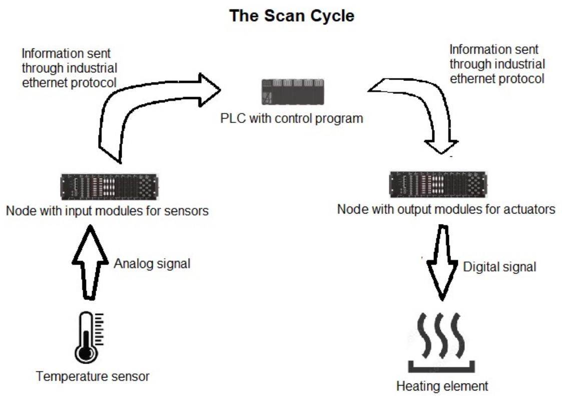

The Scan Cycle
Information sent through industrial ethernet protocol
PLC with control program
Information sent through industrial ethernet protocol
Node with input modules for sensors
Node with output modules for actuators
Analog signal
Digital signal
Temperature sensor
Heating element

Figure 5: Simple flow-chart of how information travels during the scan cycle.

Returning to the heating element example, the signals sent from the temperature sensor and to the heating element can either be *analog* or *digital* [11][12]. Different *modules* are used to process and generate digital or analog *input/output (I/O)* signals. The analog or digital I/O modules are placed at *nodes* which either process received data from the PLC during the output phase, or package data collected from the modules and send them to the PLC during the input phase. The signals are usually sent through some industrial communication protocol. In the case of B&R's X20 systems used in this project, the protocol is called *Powerlink*, which is the company's proprietary protocol [11][12].

Every communication protocol operates differently. In Powerlink the communication is organized in such a way that only one node can send data at a time [11]. The communication can be synchronized according to four different specified time synchronizations; *system time (internal on the PLC), X2X-link Time (for each X2X-link network), Powerlink Time (for each Powerlink network),* and *Time Data Points of I/O Modules* [13].

The data is transferred in three periods. First an initialization frame is sent to all nodes, thereafter cyclic data is transmitted, followed by transmitting non-time critical data [11]. Up to 240 real-time devices (including the main node) can be connected in one network segment. Each node can receive and transmit 1490 bytes of process data and 1500 bytes non time-critical data per cycle with the lowest theoretical cycle-time at 100 µs. There are options available to increase the amount of cyclic (using "Poll Request" and "Poll Response") and non time-critical data (using "Asynchronous Send") [11][12]. The data transmitted to and from the PLC is stored in the previously mentioned corresponding I/O image table which resides inside the RAM of the PLC [7].

Except data sent between nodes and the PLC, other hosts can exist in the network and send communication requests. Data can be sent either synchronously if the other host requires time critical data, or asynchronously [11]. An example of when asynchronous data is used is for devices known as *Human Machine Interfaces (HMIs)*, which provide information to personnel or operators regarding the physical process and to interact with the control system. Updating these types of devices every few milliseconds is sufficient, and therefore such data can be sent asynchronously [11][9]. Performance increments can be gained from allowing *multicast* packets to be sent if several nodes with only a small amount of process data are connected [11].

NOTE: There are events taking place during startup and shutdown. However, these events are not important for the project, the important events occur during the cyclic operation.

## III.II The real-time operating system

To deterministically process inputs, execute the logic program, and update the corresponding outputs, PLCs employ *real-time operating systems (RTOS).* The fundamental building block of the RTOS is the *kernel*, which generally consists of a *scheduler, objects, and services* [14]. Every event within the scan cycle can be divided into *tasks.* Tasks are schedulable objects containing sets of instructions and an assigned *priority.*

The scheduler uses algorithms to allocate CPU time to complete tasks within an established time frame [14]. In the case of the B&R system, which is based on the RTOS known as *VxWorks*, the algorithms used are *priority based preemptive* and *round-robin* [15]. Preemptive scheduling means that a currently running task can be preempted if there is another task with a higher priority ready. Meanwhile, round-robin allocates bursts of CPU time for every task until completion [15]. A task can be in one of three states: *running, ready,* or *blocked.* In the running state, the task has the highest priority and is currently being executed on the CPU core. In the ready state, the task is ready to run but lacks priority. In the blocked state, the task has requested access to an unavailable resource or is waiting for an event [14].

Every task is associated with a data structure called the *task-control block (TCB)* [14]. The TCB stores information regarding the specific task, including the program counter, stack pointer, other register contents, identification token, status, priority, and pointers to reserved resources as well as to the next or previous task's TCB. Whenever a task is ready which has higher priority than the one currently running, a *context switch* occurs. The context in this case is every parameter required for the CPU to execute said task. The current task's context is stored in its associated TCB, the task is released from the CPU and RAM, and the next task's context is loaded from its associated TCB and now running on the CPU. Objects are specific kernel structures, apart from tasks, there are:

- *semaphores*: allow a fixed number of tasks to enter a shared resource.
- *mutexes*: ensures only one task can access a shared resource.
- *events*: contains a set of *flags* to indicate specific events which a task could be waiting for.
- *mailboxes*: belong to a task and stores messages.
- *queues*: a buffer like FIFO data-structure to exchange data between tasks.

A combination of suitable objects are used to make up services such as inter-task communication, synchronization, or reserving resources [14]. CPU usage (%) and tasks are synonymous and to properly evaluate how much a device impacts the system a method called *task isolation* can be utilized. Tasks associated with the addition of a new type of device can be identified from logging task execution times. From the logs timing information parameters such as frequency and elapsed time can be used to pinpoint important tasks associated with the new device [16]. Measurements can now be improved if the PLC system allows user defined blockage of tasks which are non-vital for the new device.

To do a proper evaluation would also entail measuring how the CPU usage (%) responds based on the input from said device. All while capturing how concurrent tasks communicate, as well as how the RTOS scheduler preempts the tasks [16]. However, accessing all this information is sometimes not feasible in common commercial PLC systems due to information stored in TCBs being several abstraction layers away from the user. Furthermore, altering the state, priority or any other information in the task's TCB is non-advisable without proper knowledge about the system.

Within the B&R system there exists an *idle task* with the priority 0, which is the lowest priority in the entire system [17]. The idle task is scheduled whenever no user task or runtime associated task is running. A decrease in idle task time typically reflects an increase in system load. As such, measuring the change in idle task when adding a new device provides information regarding the change to the entire system. However, this also includes irrelevant activities unrelated to the new device such as updating HMIs or Ethernet communication. This is why task isolation would be recommended for higher accuracy.

Although highly prioritized and made up of many different tasks, the user defined cyclic control program does not have a specific priority within the system. For the B&R system there are something known as a task class [17]. The task class is made up of tasks with the same cycle time and priority. The user defined cyclic control program can be placed in one of eight task classes. Placing the control program in a lower task class increases the priority of every task associated with the program. As such, the control program used for increasing the CPU usage (%) is placed in the lowest task class. The allocated CPU time for idle task and cyclic control program is referred to as idle usage (%) and cyclic usage (%) respectively [17].

There are two aspects associated with the RTOS which specifically impacts the run-time performance of the PLC, namely *context switching time* and *interrupt latency* [15]. Context switching time is the summarized period of time during which the context of the currently running task is stored and the next task's context is loaded. The number of preemption's occurring during the scan cycle depends on the algorithms used by the scheduler, and relates directly to the total context switching time [15].

Interrupt latency on the other hand is the time difference between the moment an interrupt is generated, and the associated interrupt handler generates an external response [15]. The RTOS has to react quickly to external interrupt requests generated by either internal mechanisms, e.g., timers or counters, or peripherals such as I/O modules [18]. *Interrupt service routine (ISR)* is the service used by the kernel to protect the integrity of kernel data whenever an interrupt occurs, since interrupts introduce concurrency access to sensitive kernel data. The ISR uses synchronization mechanisms to protect the data. The latency is measured as the time that it takes from the arrival of an interrupt at the processor, to the start of the associated ISR. The latency varies between systems and depends on the mechanisms used, the order in which they are serviced, and the nature of the task which was interrupted [18] [14].

While the RTOS manages hardware resources and provides fundamental real-time services, the *runtime environment* serves as the abstraction layer between the control program and the RTOS [19]. Sometimes the word RTOS and runtime is used interchangeably, as in the case of the *Automation Runtime* used in B&R's system [20]. The runtime environment is usually built on top of the RTOS using *application programming interfaces (APIs)*. The runtime environment interprets the control program's instructions and interacts with the RTOS to ensure real-time execution [21].

The runtime environment also includes monitoring, which uses performance indicators such as period of tasks, maximum time frames, worst-case execution time, or response time to assess the system performance [19]. If any deviation from expected system behavior is detected, the runtime monitoring can use said performance markers to detect and mitigate the root cause [19]. There are several different ways CPUs provide monitoring tools and the Intel based B&R system uses an internal Performance Monitoring Unit (PMU) [22].

## III.III Hardware

Most manufacturers design the PLC to act as a modular system. The base unit consists of essential hardware such as CPU, memory, and communication ports. The system can then be expanded with additional devices or functionalities using modules, see Figure 6. Modules can be attached to the PLC system either *serially*, using X2X-link communication, or through nodes using Powerlink. There exist numerous types of modules, this project specifically investigates modules providing additional analog or digital I/O communication.

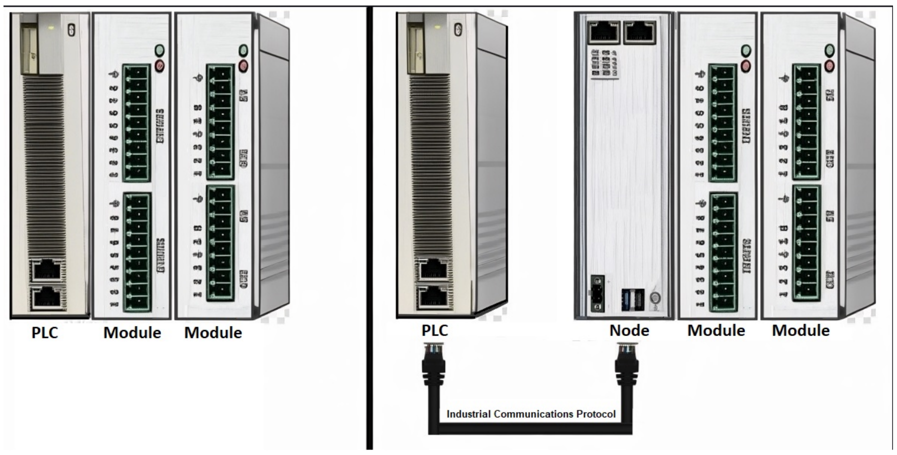


Figure 6: Connection example: modules connected serially (left), through nodes using Powerlink (right).

There are constraints issued on the hardware used in PLC systems to increase the predictability and inherently real-time performance [23]. As such, CPU architecture is typically not multi-core since this could potentially affect the predictability of task execution. The available system memory might also be kept small to decrease access times and lower the use of extensive memory management services. Operating conditions such as temperature, humidity, vibrations, and electromagnetic radiation must be accounted for since the PLC is meant to be running for very long periods of time without interruption [23].

The modules used together with the PLC either process analog signals, measured either as a current or voltage in a certain range, or digital signals, coded as true or false. The analog modules introduce a small additional time delay to the system due to sampling and reconstruction. When comparing the analog and digital input modules used (X20AI4622 and X20DI9371), there is a small difference in the minimum allowed cycle time [24] [25].

## III.IV Limitations in network capacity

As previously mentioned, there exist a default limitation of 240 nodes in the Industrial Communication Protocol provided by B&R. But it is not currently known if there exist weaknesses which lowers said limitation. Many PLC systems have two separate networks, Ethernet which is used for communication externally, and an internal network for devices, e.g., Powerlink. A good starting point to figure out any weaknesses is to see if any of the two networks share resources. This can be done by performing something known as UDP-flooding (User-Datagram-Packet-flooding) to the Ethernet network. What makes UDP-packets useful in these scenarios is the lack of “handshakes”, meaning that the PLC system just tries to accept as many of the packets as possible. No concern is given to the order of the packets received. This makes it possible to “flood” the system with data which the PLC might not be able to process. If an impact on the Powerlink network can be seen when performing said flooding, then the conclusion is that the two networks are linked to the same resources.

The Powerlink network can also be subjected to injections of packets which the system is familiar with, but not necessarily expects. If the system lacks protection for filtering which packets to process on the internal network, then the possibility exists that “false” devices can be inserted into the communication. Potential weaknesses can be found by intercepting “normal” communication on the internal network, then modifying the content of each packet, and re-transmitting the modified communication. If no filtering occurs the PLC will answer each modified packet, allowing us to further evaluate the limitations on the network. However, these types of industrial communication protocols are typically very robust and the likelihood of discovering weaknesses is very slim.

Every request sent by the PLC is answered by the nodes with an Ethernet multicast frame, also known as a *Link-Layer Discovery Protocol Multicast* message, abbreviated LLDP-multicast [26]. This fact can be used to simulate other devices on the network as it essentially is a response generated by a query from the PLC. Another form of response from the nodes are *Adress Resolution Protocol* requests, abbreviated ARP request [27]. The ARP request is sent by every device on the network to discover the physical addresses of other devices. The ARP request could potentially be used to overwhelm the PLC since the PLC has to respond to the query to establish a device connection.

As previously mentioned, there are options available to increase the amount of cyclic and non time-critical data. These types of enhancements are not investigated in this work since the model should be generally applicable to any PLC system. Another topic related to the subject is network topologies. Only linear/hub topology is investigated in this work, delving into the intricacies of other topologies is outside the scope of this project.

## III.V Program optimization

The IEC 61131–3 standard defines five programming languages, two textual, i.e., *Structured Text (ST)*, and *Instruction List (IL)*, as well as three graphical, i.e., *Ladder Diagram (LD)*, *Function Block Diagram* (FBD), and *Sequential Function Chart (SFC)* [8]. The graphical languages originate from efforts to visually represent the relay logic controllers that were previously used in industrial automation. As such, the graphical languages provide a good overview of the sequential logic within the PLC system. In contrast, Structured Text (ST) more closely resembles modern high-level programming languages and provides more efficient handling of large quantities of data [8]. Only ST will be used since it provides all functionality needed to evaluate both program optimization and the impact of devices. Investigating all five languages is also unfortunately beyond the scope of this work.

There exist three types of *program organization units (POUs)* to structure PLC code, i.e., *programs (PRGs), function blocks (FBs), and functions (FCs) [8].* The PRG manages the entire PLC and can consist of any of the other POUs, as well as unorganized code. Both PRGs and FBs possess internal memory, while FCs are executed immediately when called. As such, only FBs can be instantiated [8].

There also exist *User-Defined Data Types (UDTs)*, which are structures of other data types [8]. The main purpose is to organize the control program by reducing the amount of code. The difference between POUs and UDTs is the fact that UDTs only represents a structure made of several components of the same or different data types, while POUs contains executable code [28]. It is important to take great care when structuring UDTs to avoid “padding-bytes”, which occur to align memory addresses with the processor’s memory granularity [29]. The Intel Atom used in B&R’s system uses 32-bit instruction sets [30]. As such, the structure of UDTs should align with the 32-bit memory granularity to prevent padding bytes.

## III.VI Developing a scalability and predictability model

Scalability is a term used to evaluate the PLC’s capability to incorporate more devices into the system. Some boundaries have already been mentioned in previous sections, however, the derived model will be built on acquired test results. Predictability on the other hand represent the deterministic nature of the PLC system, i.e., the output should be the same for every repeatable input, with very small variations in execution time.

There are several factors in off-the-shelves PLC units which introduce variability into the system. For instance, general purpose processors are designed with speculative features, such as deep pipelines, out-of-order execution of instructions, unpredictable memory hierarchy, and complex replacement policies [31]. Features which heighten the functionality of the system, but lowers the predictability. Unpredictable features of the hardware also propagate onto software implementation, as it is nearly impossible for the RTOS to statically verify any timing properties [31]. Analysis techniques used to estimate system behavior might not always be valid due to unpredictable factors in the runtime, e.g., asynchronous data transfer, or runtime changeable priorities [19].

The measurements used for testing are the idle and cyclic CPU usage (%). For idle (%) usage it is specifically the assigned CPU time for a task named "idle_task", which has the lowest priority within the system [32]. For cyclic (%) it is the time the CPU spends executing tasks related to the user defined cyclic control program, and the associated task priorities depend on the assigned priority of the cyclic program. An increase in load, such as a more complex control program or added devices, decreases the idle (%) and increases the cyclic (%) CPU usage [32].

The choice of mathematical model used for predicting the impact on performance is primarily determined by the volume of the collected samples. If larger volumes can be gathered the obvious choice for a mathematical model would be to utilize machine learning methods for predicting CPU usage (%) [33]. Unfortunately, due to the time constraint of this project the chance of being able to gather larger sample volumes is slim. Instead, the first choice is to try *linear approximation*, if the predictions match poorly, *polynomial regression* could be utilized to capture non-linear behavior. The *ridge regression* used in the method is essentially polynomial regression with the addition of a normalization factor which prevents coefficients from growing to too large values.

To fully capture the predictability of the system, there exist probabilistic models for capturing the risk of failure due to the addition of a device given the state of the system [34]. But, as with the model for scalability, the time constraint does not allow for the time consuming effort. Instead, any sudden spikes or changes to CPU usage (%) will be documented in an attempt at localizing sources for instability.

# IV Method

This chapter covers the methods used for testing the impact of nodes/modules, the limitation of the network in terms of number of supported devices, and finally program conventions which optimize the number of supported devices. As previously mentioned the system used for testing is the B&R's X20 system with all associated devices.

## IV.I Equipment and setup

Hardware equipment, [35], [36]:

- B&R X20CP3586, PLC with Atom E680T CPU @ 1.6GHz with 512 MB DDR2 SDRAM (RAM) and 1 MB SRAM (User).
- B&R X20BB81, base module which serves as building block for a single node.
- B&R BC8083, bus controller which serves as the access point to a single node.
- B&R HB2880, network hub used to listen in on Internet traffic on the node.
- B&R AI4622, analog input module (only 4 channels are used).
- B&R AO4622, analog output module (only 4 channels are used).
- B&R DI9371, digital input module (only 4 channels are used).
- B&R DI6322, digital output module (only 4 channels are used).
- B&R PS9400, power supply.
- FLUKE 125B, mobile oscilloscope used to measure analog output signal.

Configuration:

- The system tick was synchronized to the Powerlink clock speed.
- Profiler was set to evaluate measurements using 250 000 entries.

Software [32], [37], [38]:

- B&R Automation Studio, development environment used to program the PLC.
- B&R Profiler, diagnostics tool used to measure CPU usage.
- Python 3.8.7 with Scapy, used for manipulating IP-packets.
- Wireshark 4.2.2, used to intercept IP-packets.
- Imported libraries *"brsystem"* and *"LoopContr"* into Automation Studio.

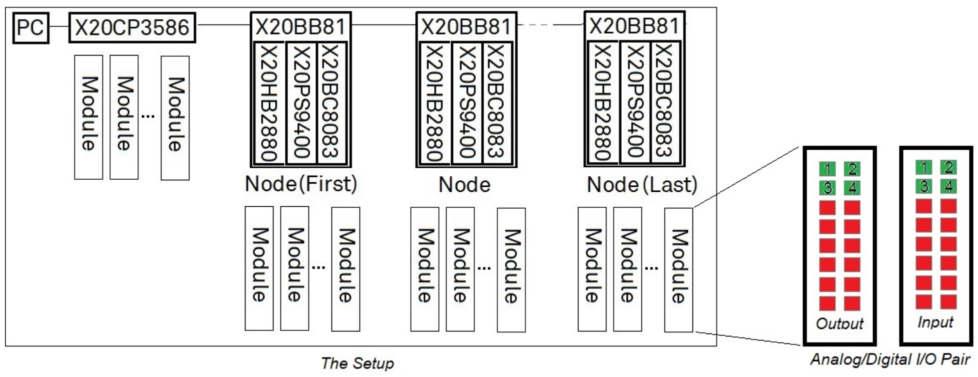


Figure 7: Example of how the B&R PLC system was set up with linear/hub-topology. Each module actually contains I/O pairs where 4 channels were cross-coupled. When modules were attached directly to the PLC (X20CP3586), they are referred to as *serially connected* using X2X-link connections. The PC is connected to the PLC through an Ethernet connection. The nodes and modules are connected to the PLC through a Powerlink connection.

I/O Pairs:
Every module used in the setup has been cross-coupled, see Figure 7. Each of the four channels on the input module is connected to the corresponding channels on the output module. Since analog modules only contains four channels, only four channels has been used on the digital modules as well.

## IV.II Node and I/O testing

In this section the difference in impact on CPU usage (%) between different types of devices is established. First the difference between analog and digital signals, then a single empty node, followed by measuring combinations of nodes and modules. At the end, a verification setup is used for predicting the change in CPU usage (%) when adding a device. Instead of changing a single device in the verification setup, entire nodes with modules are changed.

### Methodology

Each collected data sample contains three measurements. Verification of the collected sample was then done after the PLC system was made powerless for half a minute. It is important to note that the setup which was measured was first fully assembled, then disassembled device by device until only the PLC was left. The reasoning for only measuring fully assembled setups can be seen in Figure 13-14. Not only are the measurements fluctuating in Figure 14, the measurements were also inconsistent. Meanwhile, in Figure 13 the measurements were consistently increasing and decreasing for cyclic usage (%) and idle usage (%), respectively. The conclusion is that the system more quickly stabilizes itself when removing a device compared to adding. Measuring the impact of a single input module was not possible since it required an external signal source, which unfortunately could not be acquired. To solve this, every output module is connected to an input module which in turn formed I/O pairs, see Figure 7. A limit on the number of channels used in each module was set due to the analog modules only having four channels. All nodes are connected in hub/linear topology.

### Analog vs. Digital

Determining the difference between the analog and digital modules was done by iteratively connecting one input and output module together at a time, referred to as an I/O pair, serially to the PLC using X2X-link communication. At first analog I/O pairs were evaluated using the control program (see Code Snippet 1), the test ended at a total of five I/O pairs. The measurements were repeated for digital I/O pairs after removing the analog I/O pairs.

//

```
FOR counter := -1500 TO 1500 BY 1 DO    | FOR counter := -1500 TO 1500 BY 1 DO
    angle := ASIN(angle+0.01);          |     angle := ASIN(angle+0.01);
    myString := 'string';               |     myString := 'string';
    myInt := 5;                         |     myInt := 5;
    myConvert := INT_TO_REAL(myInt);    |     myConvert := INT_TO_REAL(myInt);
END_FOR                                 | END_FOR
out1 := out1+1;                         | out1 := NOT out1;
...                                     | ...
out20 := out20+1;                       | out20 := NOT out20;
```

Code Snippet 1: PLC code for analog (left) and digital (right) I/O modules.

### Empty nodes

Determining the impact of only adding empty nodes (BB81, HB2880, BC8083, PS9400) to the system using the control program in Code Snippet 2. Two samples (cyclic and idle CPU usage (%)) were gathered, first of a single empty node followed by a sample after connecting five more nodes.

//

```
FOR counter := -32000 TO 16000 BY 1 DO
    angle := ASIN(angle+0.001);
END_FOR
```

Code Snippet 2: ST code for empty nodes program

## Balanced cyclic vs. idle CPU usage (%)

Samples (cyclic and idle CPU usage (%)) were gathered for each node and module added to the system using the control program in Code Snippet 3. The final setup consisted of five nodes with two analog I/O pairs per node. At the end, samples were gathered after removing one module at a time until the node was empty, at which point the whole node was removed.

//

```
FOR counter := -32000 TO 17500 BY 1 DO
    angle := ASIN(angle+0.001);
    myInt := 5;
    myString := 'string';
    convert := INT_TO_REAL(myInt);
END_FOR
out1 := 1;
...
```

Code Snippet 3: ST code for balanced cyclic vs. idle usage (%) program.

## Unbalanced idle CPU usage (%)

Using the control program in Code Snippet 4, seven analog I/O pairs were connected to a single node. A second node was then connected, and the I/O pairs were moved to the newly added node, leaving the first node empty. A third node was then connected, and the I/O pairs were once again moved to the third node, leaving two empty nodes connected between the third node and the PLC. Once a difference in performance was noted, only one I/O module was added at a time until a “time cycle violation” occurred. Visual representation of unbalanced setup with modules placed at middle node can be seen in Figure 8.

//

```
FOR counter := -32000 TO 14000 BY 1 DO
    angle := ASIN(angle+0.001);
END_FOR
out1_1 := 1;
out1_2 := 2;
---
out3_7 = 14;
```

Code Snippet 4: ST code used in unbalanced setup to measure idle usage (%) for three nodes and seven analog I/O pairs per node.

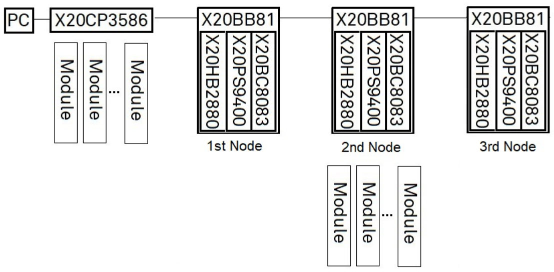


Figure 8: Example of “unbalanced” setup with all modules at the $2^{nd}$ node, while the $1^{st}$ and $3^{rd}$ nodes are empty.

## IV.III Network

The impact on the network when adding devices is evaluated in this section. First the stability of the network and impact on the PLC during UDP-flooding is investigated. Similarly, the performance is measured during an attempt at disrupting the Powerlink-network by injecting packets mimicking traffic when many nodes are active. Finally, the impact on signal output from an analog module is measured as both previous Powerlink-injections are performed once more. A visual representation of the test setup can be seen in Figure 9.

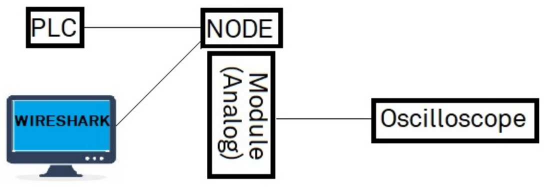


Figure 9: Single node PLC setup containing an analog module. An oscilloscope measures a square wave signal from the module. Network communication through the node is monitored using a PC running Wireshark.

### Injecting UDP-packets

An empty node (BC8083, HB2880, PS9400) was connected to the PLC. A customized UDP-packet (see Code Snippet 5) was designed to be continuously sent, commonly referred to as flooding, to the IP-address of the PLC using its Ethernet port. No configuration was done within Automation Studio to intercept UDP-packets. A PC running Wireshark was used to monitor the communication between the node and the PLC.

//

```
import socket
import time
def send_upd(ip, port, message, a):
    udp_socket = socket.socket(socket.AF_INET, socket.SDCK_DGRAM)
    udp_socket.sendto(message.encode('utf-8'), (ip,port))
    if a == 1:
        udp_socket.shutdown(socket.SHUT_RDWR)
    udp.socket.close()
if __name__ == "__main__":
    local_ip = "127.0.0.1"
    target_port = 1234
    maxSize = 1472
    message_to_send = "A"*maxSize
    try:
        while(1):
            send_udp(local_ip,target_port,message_to_send,0)
            time.sleep(1)
        except KeyboardInterrupt:
            send_udp(local_ip,target_port,message_to_send,1)
```

Code Snippet 5: Python script using the standard "sockets" library to create UDP packets with a specified size and destination. (Disclaimer: port and destination specified are not the ones used for the test.)

### Injecting Powerlink-packets

The setup in section *IV.III Injecting UDP-packets* was reused. However, this time Scapy was used to modify an intercepted Powerlink-packet from the hub, see Figure 9. Clones of the Powerlink-packet were generated, except for the last three address fields of the MAC-address which were set to consecutively increasing values to try and mimick additional devices on the network. The Powerlink-packets were then sent to both the Powerlink-port at the PLC, the Ethernet-port at the hub, and finally to the Powerlink-port at the empty node. Finally, two more Powerlink-packets were intercepted and left unaltered, an ARP-packet and a multicast-packet (LLDP multicast), which were also used for flooding. The Python script used can be seen in Code Snippet 6.
//

```
from scapy.all import *
pkts=rdpcap("myCapture.pcap", numberOfPkts)
for pkt in pkts:
    pkt(Ether).src = new_src_mac
    pkt(Ether).dst = new_dst_mac
    pkt(IP).src= new_src_ip
    pkt(IP).dst = new_dst_ip
    del pkt(IP).chksum
    sendp(pkt)
```

Code Snippet 6: Python script using Scapy to alter captured packets. (Disclaimer: program for intercepting Powerlink packets is not presented in the figure.)

### Measuring output signal

The setup in section *IV.III Injecting UDP-packets* was reused. However, an analog output module was connected to the node. The control program was modified to send out a square wave at 1 Hz using the analog output module. The amplitude and frequency were measured with an oscilloscope, then both previous PLK and UDP injections were carried out while measuring any change to the output using an oscilloscope.

## IV.IV Program optimization

This section investigates two program optimizations which could improve memory allocation, CPU load and communication.

### Memory allocation when instantiating multiples of the same datatype

The PLC was set powerless for a few minutes to ensure that any stored data was wiped from the system's registers. A clean project was created in Automation Studio and two libraries were imported into the project, first *brsystem* and finally *LoopContr*. A variable of datatype *MEMxInfo* from the *brsystem* library and a FB named *LCPID* from the *LoopContr* library was declared. A program, see Code Snippet 7, designed to only measure memory allocation once a variable *read* was forced true, was set to run at some cycle time (any cycle time works). The program first ran without any instantiated *LCPID* variable, then a single *LCPID* variable was instantiated, and finally twelve *LCPID* variables.
//

```
myLCPID1(enable  := TRUE); (*LCPID objects were removed for the first program*)
...
myLCPID12(enable  := TRUE);
MEMxInfo(enalbe := read, mem_typ ;= brDRAM);
IF read Then
        memVal := MEMxInfo.FreeMemSize;
END_IF
read := FALSE;
```

Code Snippet 7: Program for determining the allocated DRAM heap size for instantiating the same datatype, here *LCPID*, multiple times. First a program for how much memory is allocated at the start was used, without *myLCPID* objects. Then a program for a single *LCPID* object. Finally for instantiating up to twelve *LCPID* objects.

### Verifying padding bytes

The PLC was set powerless for a few minutes to ensure that any stored data was erased from the system's registers. A clean project was created in Automation Studio and an empty IEC library was added. An UDT was defined in the empty library with the following structure of data types; UDINT, BOOL, BOOL, BOOL, BOOL, in that specific order. A variable of the UDT was declared and a program was created running at some cycle time. The variable of the UDT was instantiated in the program and the IEC standard function *SIZEOF()* was used to measure the allocated memory of the UDT. The structure of the UDT was then changed to the specific structure; BOOL, UDINT, BOOL, BOOL, BOOL. Once again the allocated memory was measured using *SIZEOF()*. The two structures can be seen in Code Snippet 8.

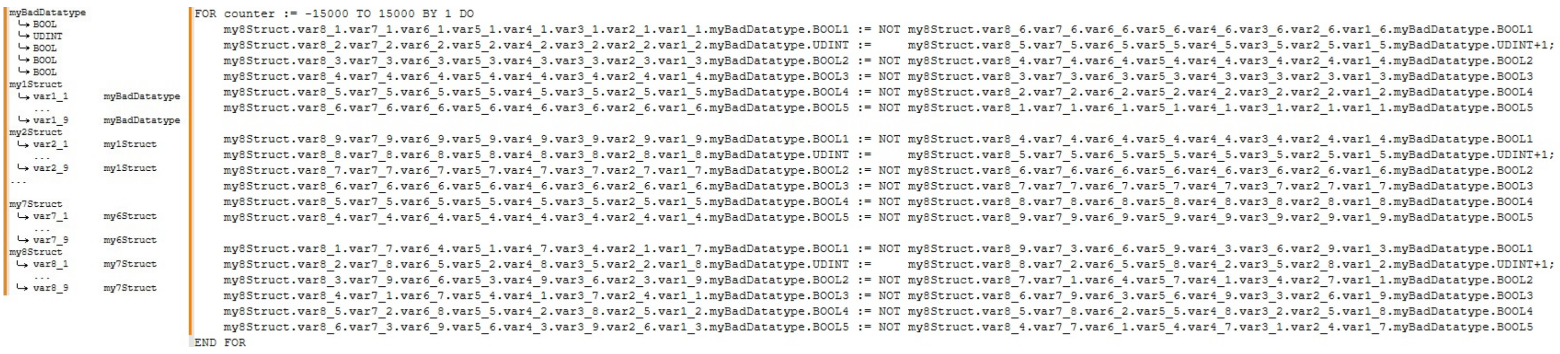
//

```
myBadDatatype:           -> Controlled using SIZEOF(myBadDatatype);
        BOOL
        UDINT
        BOOL
        BOOL
        BOOL
 myGoodDatatype:         -> Controlled using SIZEOF(myGoodDatatype);
        UDINT
        BOOL
        BOOL
        BOOL
        BOOL
```

Code Snippet 8: Structure of the UDTs with the command to verify the allocated memory using SIZEOF() function.

### Impact of padding bytes on performance

Continuing from the previous subsection *IV.IV verifying padding bytes.* A hierarchy data structure was created using first the "bad" UDT structure and finally the "good" UDT structure, see Figure 10 (left). A new program was created in which a for-loop read and write within the hierarchy, see Figure 10 (right). The difference in idle and cyclic CPU usage (%) was measured using the "bad" and "good" data structures.



```
myBadDatatype
 ↳ BOOL
 ↳ UDINT
 ↳ BOOL
 ↳ BOOL
 ↳ BOOL
my1Struct
 ↳ var1_1      myBadDatatype
    ...
 ↳ var1_9      myBadDatatype
my2Struct
 ↳ var2_1      my1Struct
    ...
 ↳ var2_9      my1Struct
...

my7Struct
 ↳ var7_1      my6Struct
    ...
 ↳ var7_9      my6Struct
my8Struct
 ↳ var8_1      my7Struct
    ...
 ↳ var8_9      my7Struct
```

```
FOR counter := -15000 TO 15000 BY 1 DO
    my8Struct.var8_1.var7_1.var6_1.var5_1.var4_1.var3_1.var2_1.var1_1.myBadDatatype.BOOL1 := NOT my8Struct.var8_6.var7_6.var6_6.var5_6.var4_6.var3_6.var2_6.var1_6.myBadDatatype.BOOL1
    my8Struct.var8_2.var7_2.var6_2.var5_2.var4_2.var3_2.var2_2.var1_2.myBadDatatype.UDINT :=     my8Struct.var8_5.var7_5.var6_5.var5_5.var4_5.var3_5.var2_5.var1_5.myBadDatatype.UDINT+1;
    my8Struct.var8_3.var7_3.var6_3.var5_3.var4_3.var3_3.var2_3.var1_3.myBadDatatype.BOOL2 := NOT my8Struct.var8_4.var7_4.var6_4.var5_4.var4_4.var3_4.var2_4.var1_4.myBadDatatype.BOOL2
    my8Struct.var8_4.var7_4.var6_4.var5_4.var4_4.var3_4.var2_4.var1_4.myBadDatatype.BOOL3 := NOT my8Struct.var8_3.var7_3.var6_3.var5_3.var4_3.var3_3.var2_3.var1_3.myBadDatatype.BOOL3
    my8Struct.var8_5.var7_5.var6_5.var5_5.var4_5.var3_5.var2_5.var1_5.myBadDatatype.BOOL4 := NOT my8Struct.var8_2.var7_2.var6_2.var5_2.var4_2.var3_2.var2_2.var1_2.myBadDatatype.BOOL4
    my8Struct.var8_6.var7_6.var6_6.var5_6.var4_6.var3_6.var2_6.var1_6.myBadDatatype.BOOL5 := NOT my8Struct.var8_1.var7_1.var6_1.var5_1.var4_1.var3_1.var2_1.var1_1.myBadDatatype.BOOL5

    my8Struct.var8_9.var7_9.var6_9.var5_9.var4_9.var3_9.var2_9.var1_9.myBadDatatype.BOOL1 := NOT my8Struct.var8_4.var7_4.var6_4.var5_4.var4_4.var3_4.var2_4.var1_4.myBadDatatype.BOOL1
    my8Struct.var8_8.var7_8.var6_8.var5_8.var4_8.var3_8.var2_8.var1_8.myBadDatatype.UDINT :=     my8Struct.var8_5.var7_5.var6_5.var5_5.var4_5.var3_5.var2_5.var1_5.myBadDatatype.UDINT+1;
    my8Struct.var8_7.var7_7.var6_7.var5_7.var4_7.var3_7.var2_7.var1_7.myBadDatatype.BOOL2 := NOT my8Struct.var8_6.var7_6.var6_6.var5_6.var4_6.var3_6.var2_6.var1_6.myBadDatatype.BOOL2
    my8Struct.var8_6.var7_6.var6_6.var5_6.var4_6.var3_6.var2_6.var1_6.myBadDatatype.BOOL3 := NOT my8Struct.var8_7.var7_7.var6_7.var5_7.var4_7.var3_7.var2_7.var1_7.myBadDatatype.BOOL3
    my8Struct.var8_5.var7_5.var6_5.var5_5.var4_5.var3_5.var2_5.var1_5.myBadDatatype.BOOL4 := NOT my8Struct.var8_8.var7_8.var6_8.var5_8.var4_8.var3_8.var2_8.var1_8.myBadDatatype.BOOL4
    my8Struct.var8_4.var7_4.var6_4.var5_4.var4_4.var3_4.var2_4.var1_4.myBadDatatype.BOOL5 := NOT my8Struct.var8_9.var7_9.var6_9.var5_9.var4_9.var3_9.var2_9.var1_9.myBadDatatype.BOOL5

    my8Struct.var8_1.var7_7.var6_4.var5_1.var4_7.var3_4.var2_1.var1_7.myBadDatatype.BOOL1 := NOT my8Struct.var8_9.var7_3.var6_6.var5_9.var4_3.var3_6.var2_9.var1_3.myBadDatatype.BOOL1
    my8Struct.var8_2.var7_8.var6_5.var5_2.var4_8.var3_5.var2_2.var1_8.myBadDatatype.UDINT :=     my8Struct.var8_8.var7_2.var6_5.var5_8.var4_2.var3_5.var2_8.var1_2.myBadDatatype.UDINT+1;
    my8Struct.var8_3.var7_9.var6_6.var5_3.var4_9.var3_6.var2_3.var1_9.myBadDatatype.BOOL2 := NOT my8Struct.var8_7.var7_1.var6_4.var5_7.var4_1.var3_4.var2_7.var1_1.myBadDatatype.BOOL2
    my8Struct.var8_4.var7_1.var6_7.var5_4.var4_1.var3_7.var2_4.var1_1.myBadDatatype.BOOL3 := NOT my8Struct.var8_6.var7_9.var6_3.var5_6.var4_9.var3_3.var2_6.var1_9.myBadDatatype.BOOL3
    my8Struct.var8_5.var7_2.var6_8.var5_5.var4_2.var3_8.var2_5.var1_2.myBadDatatype.BOOL4 := NOT my8Struct.var8_5.var7_8.var6_2.var5_5.var4_8.var3_2.var2_5.var1_8.myBadDatatype.BOOL4
    my8Struct.var8_6.var7_3.var6_9.var5_6.var4_3.var3_9.var2_6.var1_3.myBadDatatype.BOOL5 := NOT my8Struct.var8_4.var7_7.var6_1.var5_4.var4_7.var3_1.var2_4.var1_7.myBadDatatype.BOOL5
END FOR
```

Figure 10: Hierarchy data structure, here with the "bad" data structure which was later replaced with the "good".

## Model verification

The measurements from section *IV.II Node & I/O Testing* are used to establish the best relationship between the number of connected devices. A linear relationship is tested using the setup in Table 1. The nodes are, as for every previous setup, connected in hub/linear topology.

| **Node:** | **Modules:** |
|---|---|
| N1 | AO4622 AI4622 DI9371 DO6322 AO4622 AI4622 DO6322 DI9371 |
| N2 | DI9371 DO6322 AI4622 AO4622 AI4622 AO4622 DI9371 DO6322 AI4622 AO4622 |
| N3 | AI4622 AO4622 AI4622 AO4622 AI4622 AO4622 DI9371 DO6322 DI9371 DO6322 |
| N4 | AO4622 AI4622 AO4622 AI4622 |
| N5 | Empty |

Table 1: The modules connected to each node in their respective order.

# V Results

The following section contains the results from the tests carried out based on chapter *IV Method*. The variability for each measurement is displayed as the standard deviation. Normally, using averages and standard deviation for measurements not following the *normal distribution* should be avoided. However, due to the small sample sizes it is difficult be entirely certain of which distribution the measurements follow. Goodness-of-fit and probability plots (Q-Q plots) provided by 'fitdist' function in Matlab were used to identify the following distributions:

- Measurements in *IV.II Balanced cyclic vs. idle usage (%)* does not follow the normal distribution, idle usage (%) distinctly follows the *general pareto distribution*, while cyclic usage (%) follows the *kernel distribution.*
- Measurements in *IV.II Unbalanced idle usage (%)* the $2^{nd}$ node (blue bars) adheres to a normal distribution, and $3^{rd}$ node (red bars) approximates the general pareto distribution.

Due to earlier remarks regarding the small sample sizes, the choice was to still use averages and standard deviation. However, the identified distributions are still mentioned as not to disclose any information from the reader.

## V.I Node and I/O testing

This section covers results regarding the evaluation of impact on performance from the addition of a device.

### Analog vs. digital

No discernible differences were found between analog and digital modules when connected serially using X2X-link communication to the PLC, see Figure 11

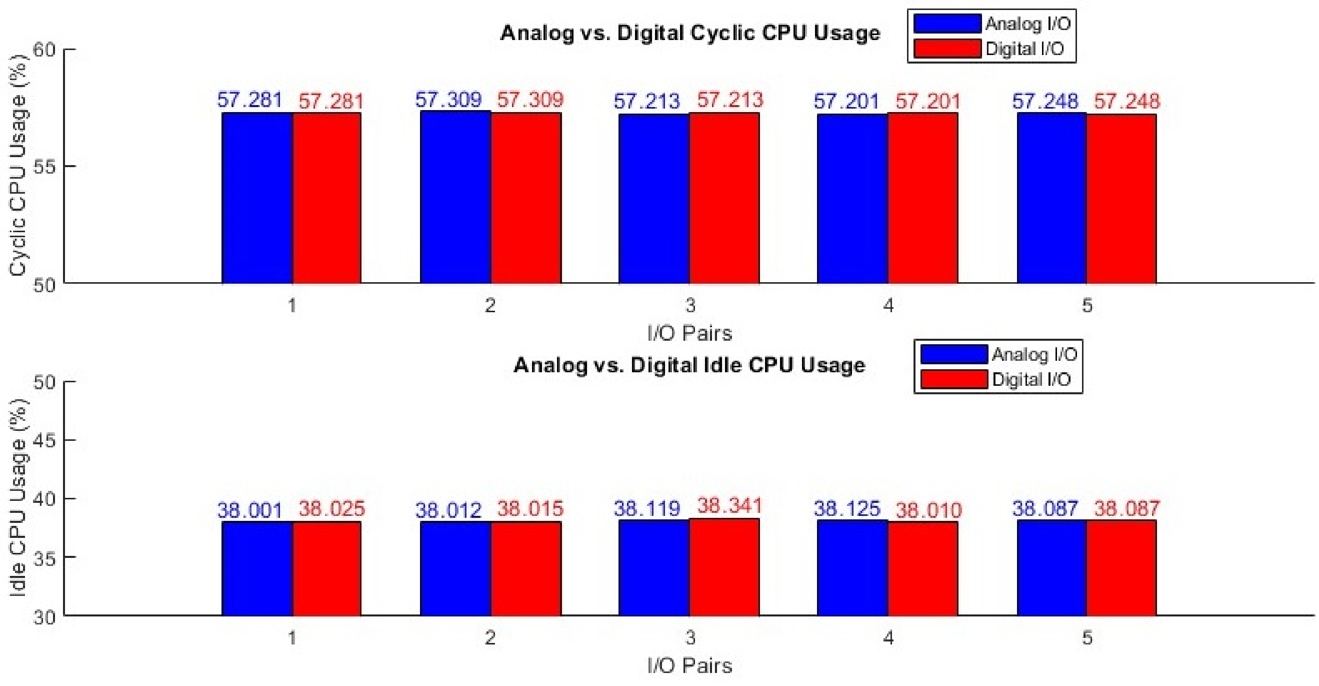


Figure 11: Comparison between analog (blue) and digital (red) modules. Each input module is cross coupled to an output module, referred to as an I/O pair. All modules were attached serially to the PLC using X2X-link communication.

**Empty nodes**

The differences (cyclic and idle usage (%)) between a single empty node and six nodes in hub/linear topology can be seen in Figure 12. Calculated averages can be seen in Table 2. As expected, the cyclic load (%) increases while the idle load (%) decreases. The change in idle load (%) appears to be more significant compared to cyclic load (%) when adding an empty node.

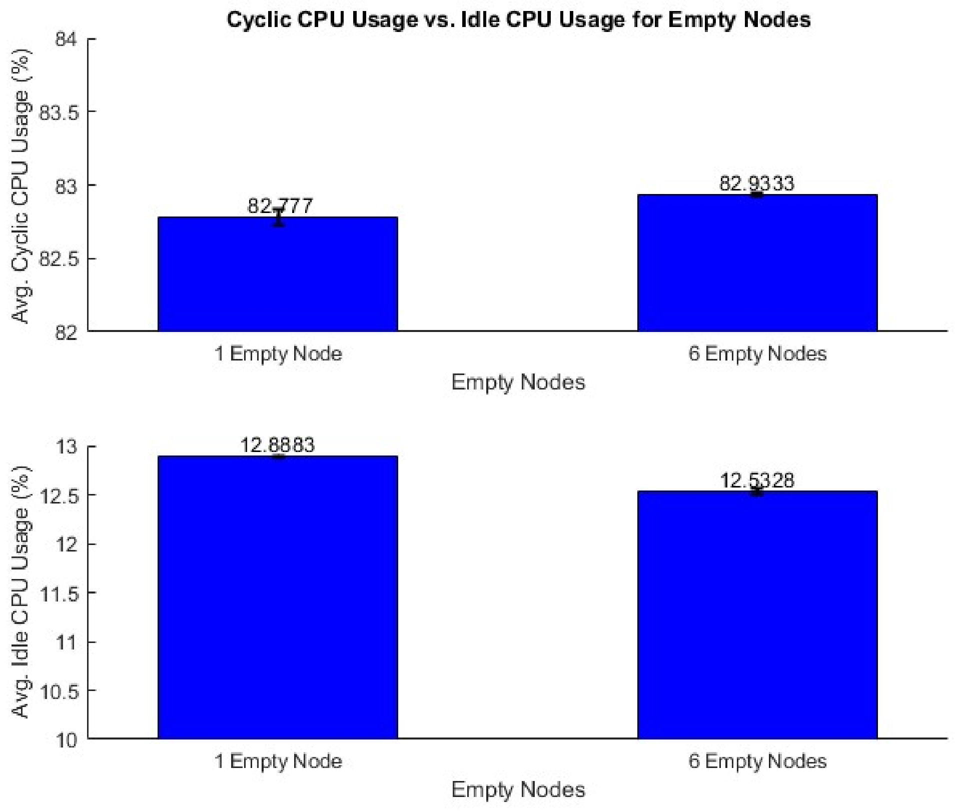


Figure 12: Cyclic and idle CPU usage (%) measurements for one and six empty nodes.

| | |
|---|---|
| Avg. cyclic CPU usage (%) per node | 0.0313 |
| Avg. idle CPU usage (%) per node | -0.0711 |

Table 2: Average CPU usage (%) for adding one empty node. Negative values for idle (%) usage due to the decreased idle CPU usage (%) when adding a device.

## Balanced cyclic Vs. idle CPU usage when removing devices

The measurements (cyclic and idle (%) usage) gathered after removing a device from the "balanced" setup can be seen in Figure 13, and Table 3. Only the differences (cyclic and idle usage (%)) for the addition of a node can be seen in Table 4. As expected, the cyclic load (%) increases while the idle load (%) decreases. Some measurement irregularities such as increased load when measuring idle load (%) and vice versa, as well as a sudden spike in load at the fourth node. The gathered results can be used for developing models predicting the impact on CPU load from both modules as well as nodes.

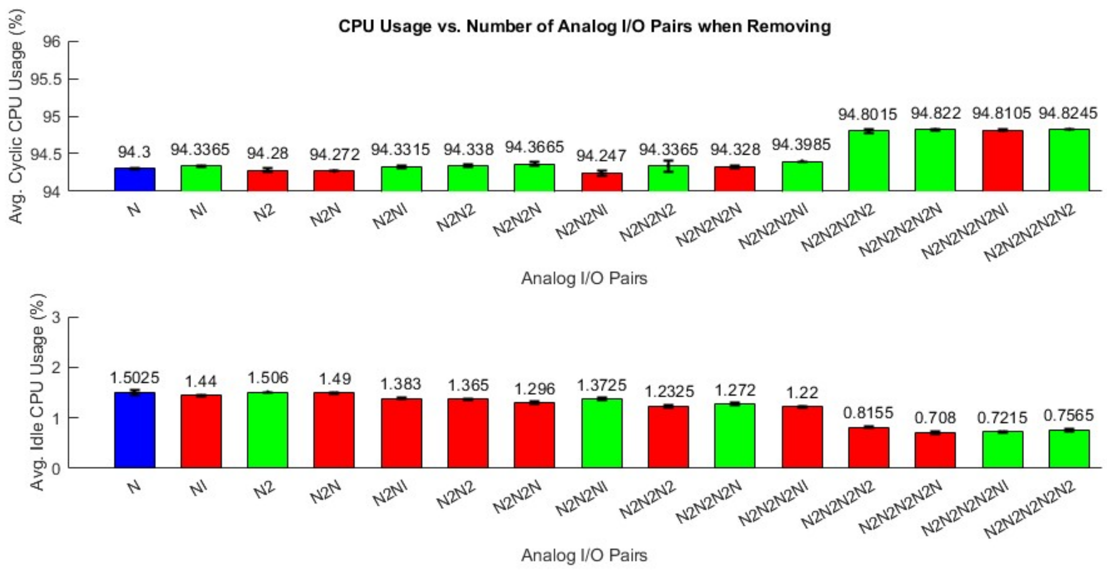


Figure 13: Cyclic and idle usage (%) for consecutively removing devices from preexisting setup consisting of five nodes with two analog I/O pairs per node. The "N" symbol represents a node, the "I" symbol represents one I/O pair and "2" represents two I/O pairs. Color change represents even (blue), increase (green), and decrease (red) in (%) compared to previous bar.

| Node/Pair: | Diff. idle (%): | Diff. cyclic (%): |
|---|---|---|
| 1/1 | 0.0625 | 0.0365 |
| 1/2 | 0.066 | -0.0565 |
| 2/1 | -0.1230 | 0.0515 |
| 2/2 | -0.0180 | 0.0065 |
| 3/1 | 0.0075 | -0.0910 |
| 3/2 | -0.140 | 0.0895 |
| 4/1 | -0.0125 | 0.0620 |
| 4/2 | -0.4045 | 0.4030 |
| 5/1 | -0.0940 | 0.0090 |
| 5/2 | 0.0350 | 0.0140 |
| **Avg. Diff (%):** | -0.0746 | 0.0525 |

Table 3: Differences between average (%) idle and cyclic usage (%) for I/O modules (excluding impact of nodes). Negative values due to the decreased idle usage (%) when adding a device. However, cyclic usage (%) values should stay positive.

| Node/Pair: | Diff. idle (%): | Diff. cyclic (%): |
|---|---|---|
| 2/0 | -0.0160 | -0.0080 |
| 3/0 | -0.0690 | 0.0285 |
| 4/0 | -0.0395 | -0.0081 |
| 5/0 | -0.1075 | 0.0205 |
| **Avg. Diff (%):** | -0.0383 | 0.008225 |

Table 4: Impact of adding an empty node to the setup. The measured difference (%) is between the previous full node and the newly added empty node, i.e., 2/0 representing the measured difference between the first full node 1/2 and the new empty node 2/0.

**Balanced cyclic vs. idle CPU usage when adding devices**

One single instance of measurements (cyclic and idle usage (%)) gathered when measuring after adding a device to a setup. This serves as an example on what the measurements looks like if the system has not stabilized between measurements. While the measurements of cyclic load (%) increases and idle load (%) decreases, it is impossible to distinguish how large of an impact each individual node and module have on the system.

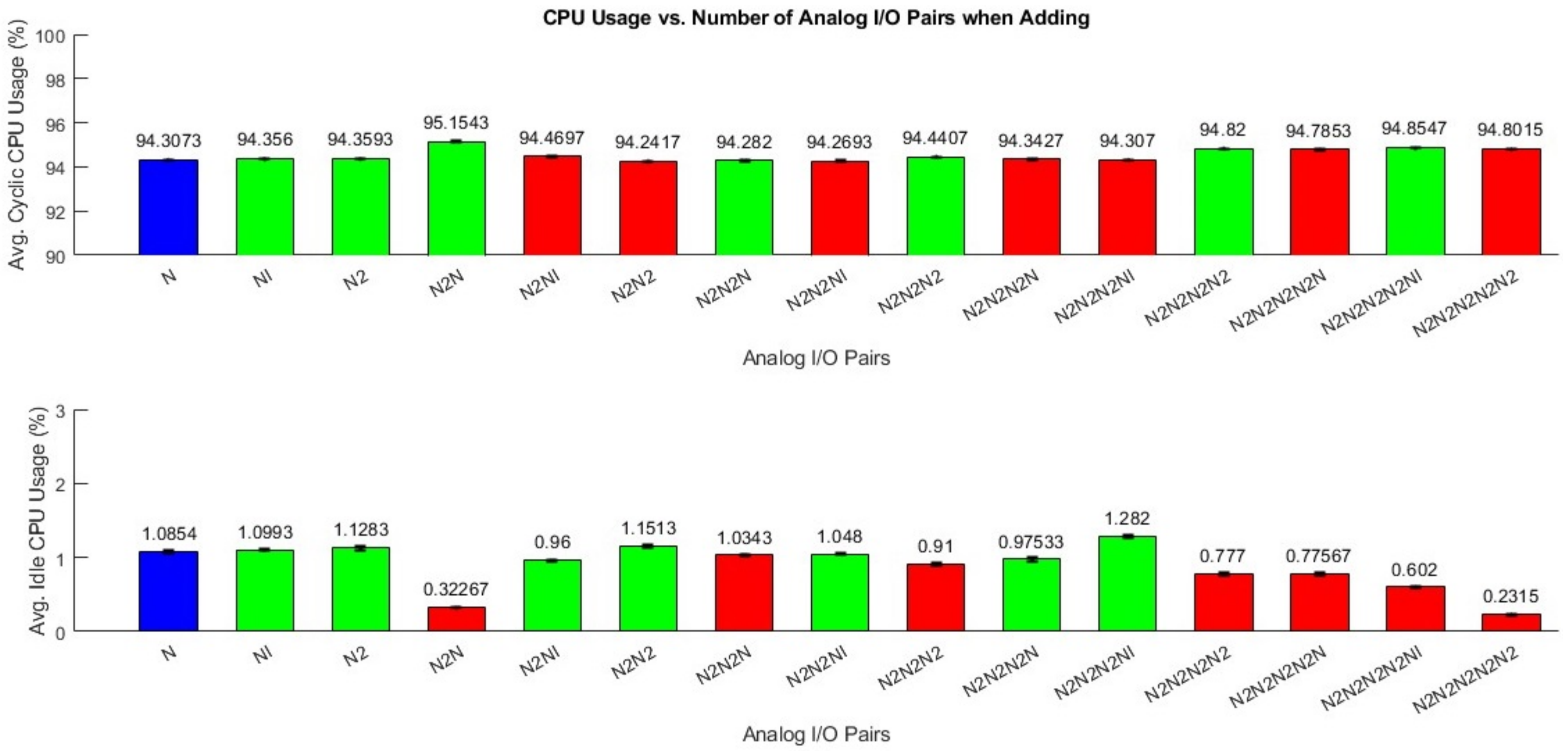
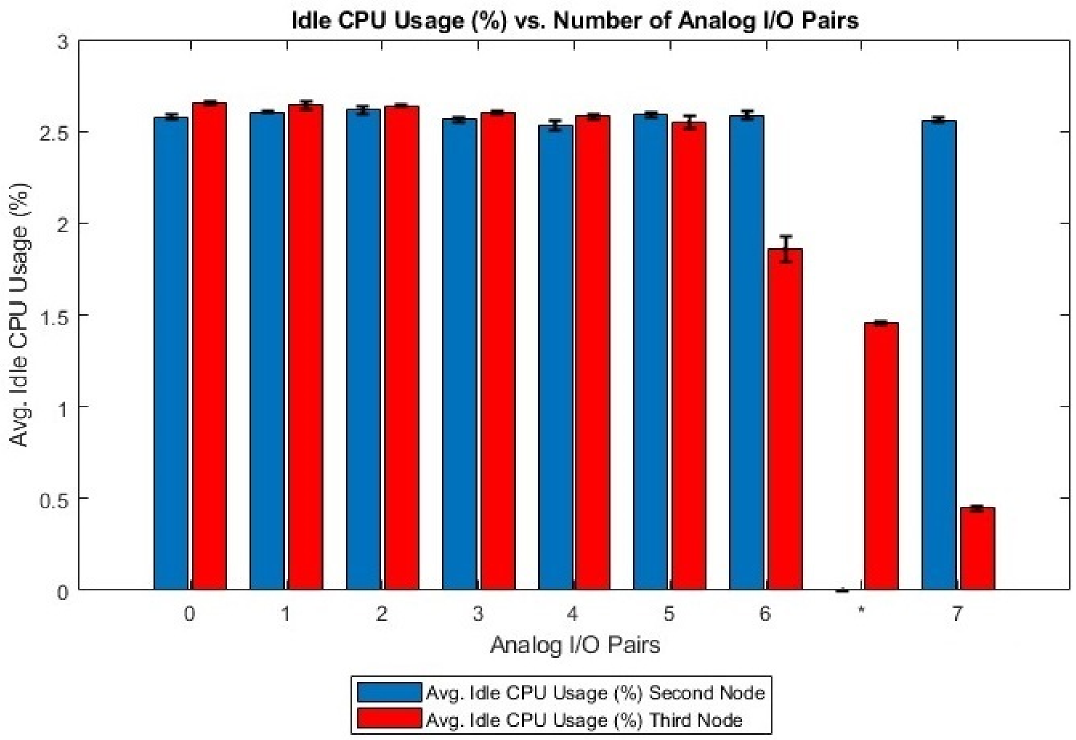


Figure 14: Non-stabilized setup, same as seen in Figure 13, but instead of removing devices measurements were taken for each **added** device. Same color scheme as Figure 13, where blue represents no change, green represents an increase, and red represents a decrease in CPU load (%) compared to previous measurement.

**Unbalanced idle CPU usage (%)**

Results from the unbalanced setup as seen in Figure 8. The measurements (see Figure 15) appears to almost reach stagnation up until the sixth module pair where a sudden change can be seen for the third node (red bars). However, upon a closer look at the bottom graph shows an interesting observation. Here it appears as if the measurements for the second node fluctuates, while the measurements for the third node steadily decreases. Suggesting that the system might be more heavily strained when the load was placed at the third node.

Average change to idle usage (%) is calculated in Table 5. Meanwhile an additional single module (denoted *) was added at the sixth I/O pair and the following measurements are presented in Table 6. The different behaviors when placing a load at the second node (blue bars) compared to the third node (red bars) further suggests that the system reacts differently to balanced/unbalanced configurations. The results also suggests that more data should be acquired in regards to how nodes behaves to acquire an improved and more accurate model.



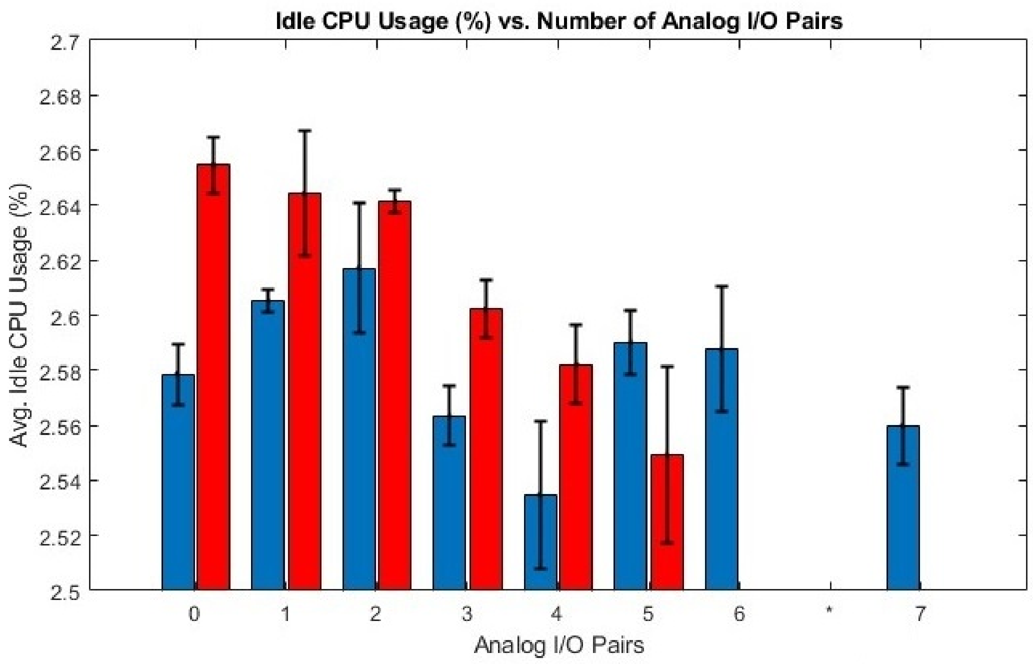


Figure 15: Idle CPU usage (%) for seven analog I/O pairs attached to an unbalanced setup. The graph below is a close up of the graph above. Blue bars represent two nodes with the first node being empty, and the second node having seven I/O pairs. Red bars represent the addition of a third node with the two first being empty, and the third node having seven I/O pairs. The "*" symbol is representing only a single analog I/O module added. Measurements from the first node are omitted due to insignificant difference to the second node.

| Node/Pair: | Diff. idle (%): | Node/Pair: | Diff. idle (%): |
|---|---|---|---|
| 2/1 | 0.0266 | 3/1 | -0.0102 |
| 2/2 | 0.0120 | 3/2 | -0.0030 |
| 2/3 | -0.0538 | 3/3 | -0.0392 |
| 2/4 | -0.0286 | 3/4 | -0.0200 |
| 2/5 | 0.0554 | 3/5 | -0.0328 |
| 2/6 | -0.0026 | - | - |
| **Avg. Diff (%):** | 0.0015 | **Avg. Diff (%):** | -0.0210 |

Table 5: Change in idle usage (%) for unbalanced setup with two and three nodes. The left side of the table represents two nodes, with the first node empty and increasing number of attached analog I/O pairs. The right side of the table represents three nodes with the two first being empty and an increasing number of attached analog I/O pairs.

| Node/Pair: | Diff. idle (%): |
|---|---|
| 3/6 | -0.6916 |
| 3/* | -0.4012 |
| 3/7 | -1.0080 |
| Avg. Diff idle (%) | -0.7003 |

Table 6: Differences in idle CPU usage (%) between I/O pairs for the third node. The " *" symbol represent the addition of a single analog input module to the 6th I/O pair on the third node. Negative values for idle usage (%), due to the decreased usage (%) when adding a device.

## V.II Network

This section covers results related to the network analysis.

### Injecting UDP-packets

No discernible disturbances were found using Wireshark. The altered UDP-packets used for flooding did not affect neither cyclic nor idle usage (%).

### Injecting PLK-packets

No discernible disturbances were found using Wireshark. Neither the altered PLK-packets, nor the unaltered ARP-packets and unaltered multicast-packets affected the cyclic or idle usage (%).

### Measuring output signal

The measured output signal (seen in Figure 16) using the oscilloscope had the same amplitude and frequency before and after the injected UDP and PLK-packets.

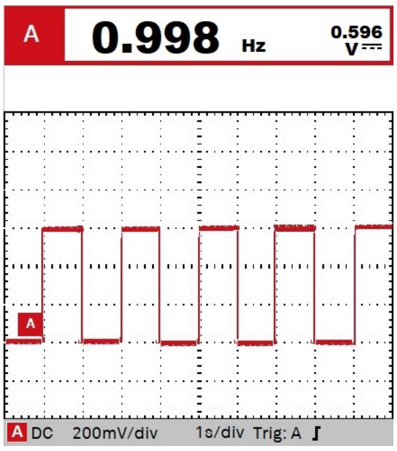


Figure 16: Oscilloscope reading for all measured signals, no difference was noted between any of them.

## V.III Program optimization

This section covers results regarding the program optimization.

### Memory allocation for FBs

The three programs were executed at three different points after the system had been disconnected for a few minutes and using clean projects. From Table 7 the average amount of allocated memory (bytes) per instantiated variable can be calculated using measurements A, B, and C. The average difference in allocated memory between the first and second instantiation of the FB (in the code *LCPID*) was 122 bytes, with a standard deviation of 0.8 bytes. After instantiating twelve of the same FB the average allocated memory dropped to 51 bytes, with a standard deviation of 1.2 bytes. The average amount of allocated memory for a single variable lowers from 1470 bytes down to roughly 51 bytes when 12 variables of the same FB is instantiated.

| No. of instantiated FBs: | Measurement A (bytes): | Measurement B (bytes): | Measurement C (bytes): | Avg./Avg. Diff. (bytes): |
|---|---|---|---|---|
| No variable | - | - | - | - |
| 1 variable of same FB | -1 464 | -1 468 | -1 478 | -1470 / 0 |
| 2 variables of same FB | -1 584 | -1 590 | -1 602 | -1592 / -122 |
| 12 variables of same FB | -2 094 | -2 090 | -2 112 | -2098.667/ -50.667 |

Table 7: DRAM heap memory allocation after instantiating new variables of the same FB. Average differences (Avg. Diff) calculated by subtracting the measured average and dividing by the number of variables. As an example, the average difference of 12 variables is $\sum(A+B+C)/3 = 2098.667$ bytes, subtracting the average for 2 variables (1592 bytes) gives us 506.667 bytes, and finally dividing by the difference in variables (12-2) results in 50.6 bytes.

### Verifying padding bytes

The two structures “myBadDatatype” and “myGoodDatatype” presented in Table.8 shows that there are indeed inserted padded bytes for “myBadDatatype” structure.

| UDTs: | Datatype1: | Datatype 2: | Datatype 3: | Datatype 4: | Datatype 5: | SIZEOF() value (bytes) |
|---|---|---|---|---|---|---|
| myBadDatatype | BOOL | UDINT | BOOL | BOOL | BOOL | 12 |
| myGoodDatatype | UDINT | BOOL | BOOL | BOOL | BOOL | 8 |

Table 8: The data structure of the different UDTs and the associated measured allocated memory (bytes).

### Impact of padding bytes on performance

As seen in Table 9, there is no noticeable impact on performance when comparing the two structures “myBadDatatype” and “myGoodDatatype”.

| Data Structure: | Avg. cyclic CPU usage (%): | Avg. idle CPU usage (%): |
|---|---|---|
| myBadDatatype | 10.297 | 83.874 |
| myGoodDatatype | 10.209 | 83.868 |

Table 9: Average cyclic and idle usage (%) for the “bad” and “good” data structure when executing the two programs in Figure 10.

**Model calculations**

**The following assumptions are made:**

- Values presented below for I/O pairs are converted to per module by simply dividing the values. The assumption is that there is no discernible difference between input and output modules.
- Positive values presented below for idle CPU usage (%) and negative values for cyclic CPU usage (%) are disregarded.

**The following values were observed:**

From Figure 11 the conclusion is made that there is no discernible difference between analog and digital modules.

From Table 3 the average impact from a module on idle and cyclic CPU usage (%). Assuming there is no discernible difference in impact on CPU usage (%) between input and output modules:

$$module_{idle} = -0.03730 \quad \text{and} \quad module_{cyclic} = 0.02625 \tag{1}$$

From Table 5 the average impact from a module on idle CPU usage (%). Positive idle usage (%) should be disregarded:

$$module_{idle} = 0.0015\,(2^{nd}\text{ node}) \quad \text{and} \quad module_{idle} = -0.0105\,(3^{rd}\text{ node}) \tag{2}$$

From Table 6 the unbalanced setup had the average impact on idle CPU usage (%):

$$module_{idle_{High}} = -0.7003 \tag{3}$$

From Table 2 an empty node impacts the idle and cyclic CPU usage (%):

$$node_{idle} = -0.0711 \quad \text{and} \quad node_{cyclic} = 0.0313 \tag{4}$$

From Table 4 an empty node impacts the idle and cyclic CPU usage (%):

$$node_{idle} = -0.0383 \quad \text{and} \quad node_{cyclic} = 0.008225 \tag{5}$$

**Linear relationship:** Captured data regarding modules and nodes are used to separate the impact of each device. Using $node_{cyclic} = 0.008225$ and $module_{cyclic} = 0.02625$ for cyclic usage (%) values, and $node_{idle} = -0.0383$ and $module_{idle} = -0.0105$ for idle usage (%) values. The values are chosen by trial and error for best result. Each node is predicted using values seen in Table 10. The nodes are, as for every previous setup, connected in hub/linear topology. The following three setups were evaluated: PLC-N1-N2-N3-N4-N5 at "medium" load (see Figure 17), PLC-N4-N1-N5-N2-N3 at "medium" load (see Figure 18) and "high" load (see Figure 19) as well as "very high" load (see Figure 20), and finally PLC-N3-N1-N5-N2-N4 at "very high" load (see Figure 21).

An interesting observation can be made in Figure 17, at medium load similar fluctuating behavior appear in both the stable (right) and unstable system (left). Further pinpointing the difficulties with measuring the CPU load of the PLC without proper knowledge of exactly which tasks are being executed. Furthermore, it appears as if the cyclic usage (%) is much more difficult to predict, compare measured values (blue) with predicted values (red).

Comparing the "medium" (Figure 18), "high" (Figure 19), and "very high" (Figure 20) load of PLC-N4-N1-N5-N2-N3 it seems as if the cyclic usage (%) continues to be more difficult to predict. The linear approximation is far better at predicting the idle usage (%). The best predictions were made at "high" load, while "medium" and "very high" load were equally difficult to predict for idle usage (%). Finally, the setup PLC-N3-N1-N5-N2-N4 at "very high" load (Figure 21) concludes that the cyclic usage (%) is very difficult to predict using the linear approximation. Except the sudden spike at PLC-N3-N1, the idle usage (%) was suddenly very easily predicted for "very high" load compared to the setup PLC-N4-N1-N5-N2-N3.

| Node: | Modules: | Predicted (%): |
|---|---|---|
| N1 | AO4622 AI4622 DI9371 DO6322 AO4622 AI4622 DO6322 DI9371 | -0.1223 (idle) and 0.2345 (cyclic) |
| N2 | DI9371 DO6322 AI4622 AO4622 AI4622 AO4622 DI9371 DO6322 AI4622 AO4622 | -0.1433 (idle) and 0.2870 (cyclic) |
| N3 | AI4622 AO4622 AI4622 AO4622 AI4622 AO4622 DI9371 DO6322 DI9371 DO6322 | Same as N2 |
| N4 | AO4622 AI4622 AO4622 AI4622 | -0.0803 (idle) and 0.1295 (cyclic) |
| N5 | Empty | -0.0383 (idle) and 0.0245 (cyclic) |

Table 10: The modules connected to each node in their respective order.

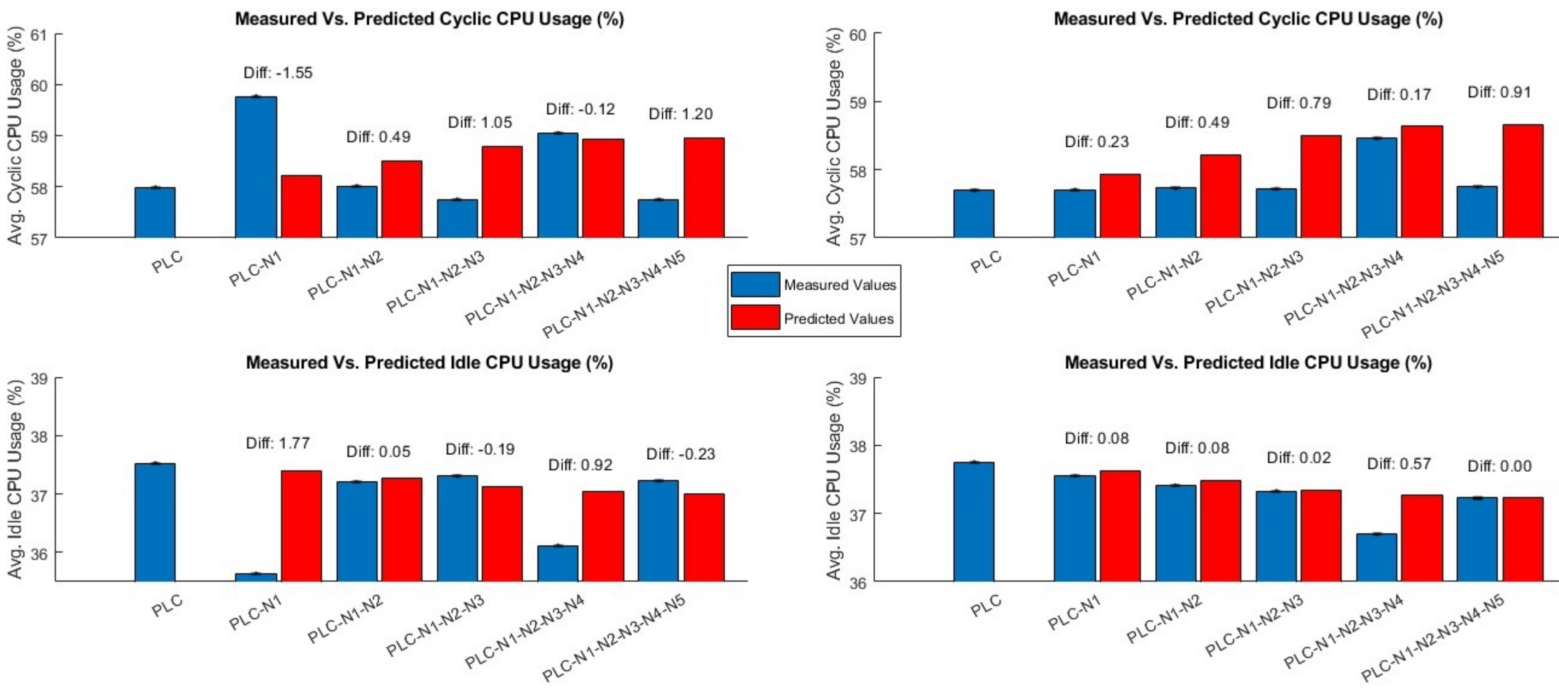


Figure 17: Linear relationship for setup PLC-N1-N2-N3-N4-N5 at "medium" load. To the left in the figure are measurements when adding devices, then removing devices to the right.

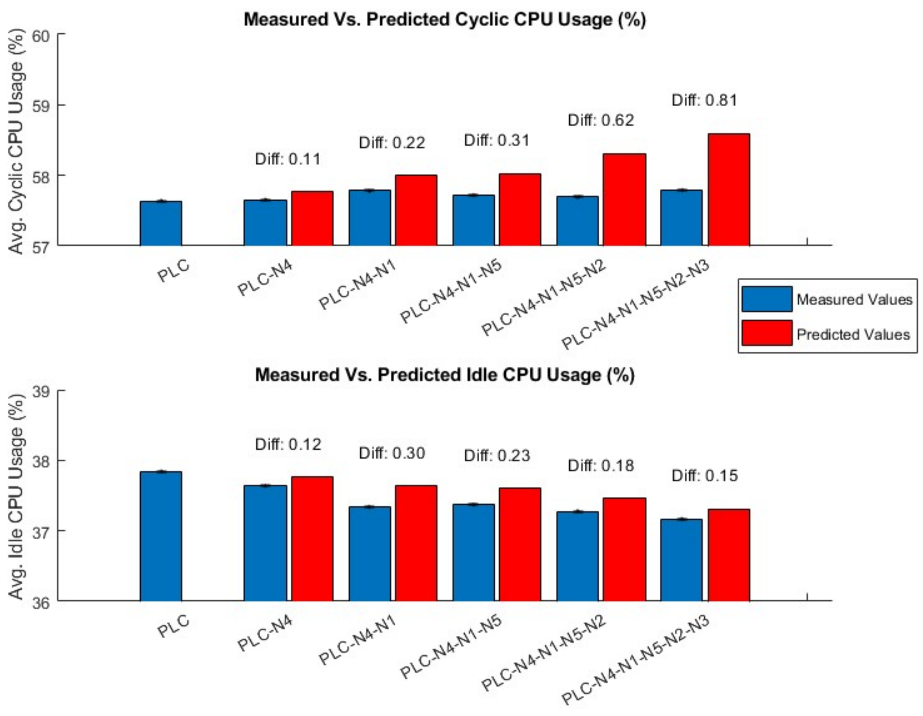


Figure 18: Linear relationship for setup PLC-N4-N1-N5-N2-N3 for “medium” load.

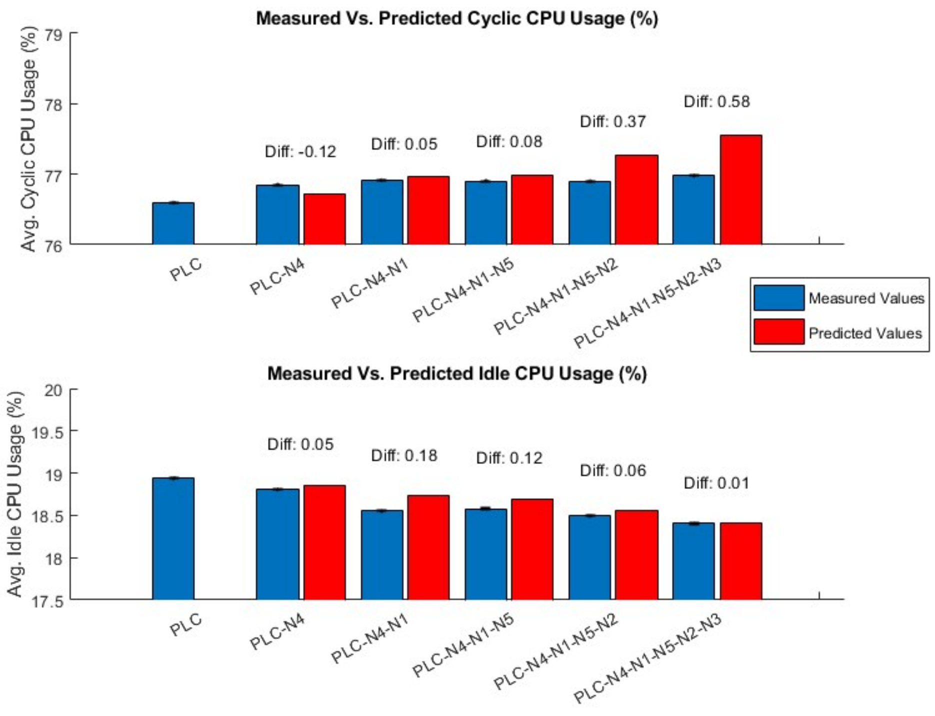


Figure 19: Linear relationship for setup PLC-N4-N1-N5-N2-N3 at “high” load.

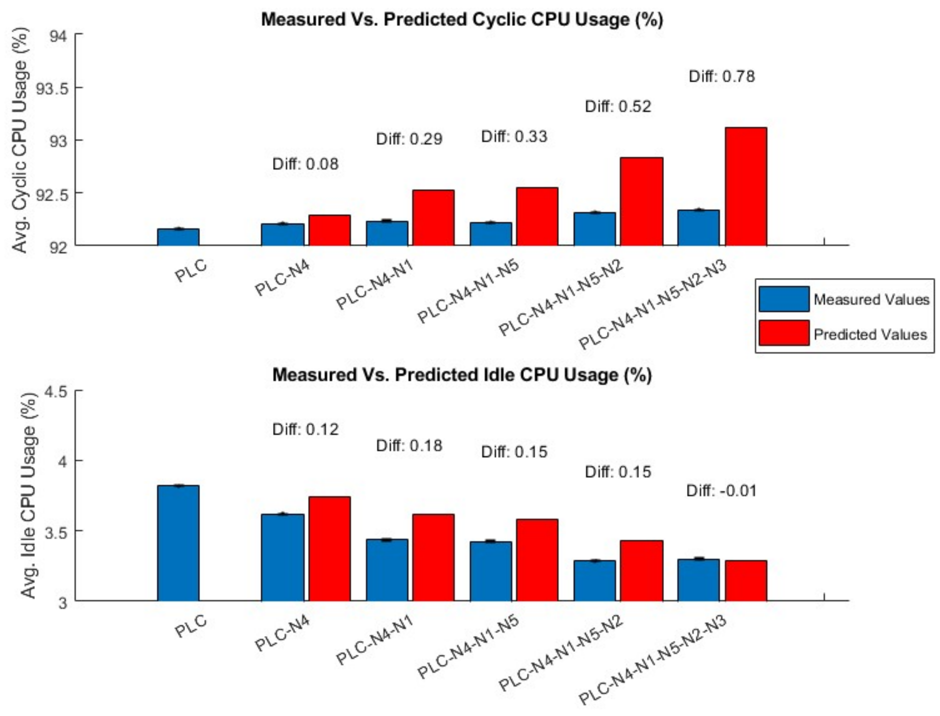


Figure 20: Linear relationship for setup PLC-N4-N1-N5-N2-N3 at “very high” load.

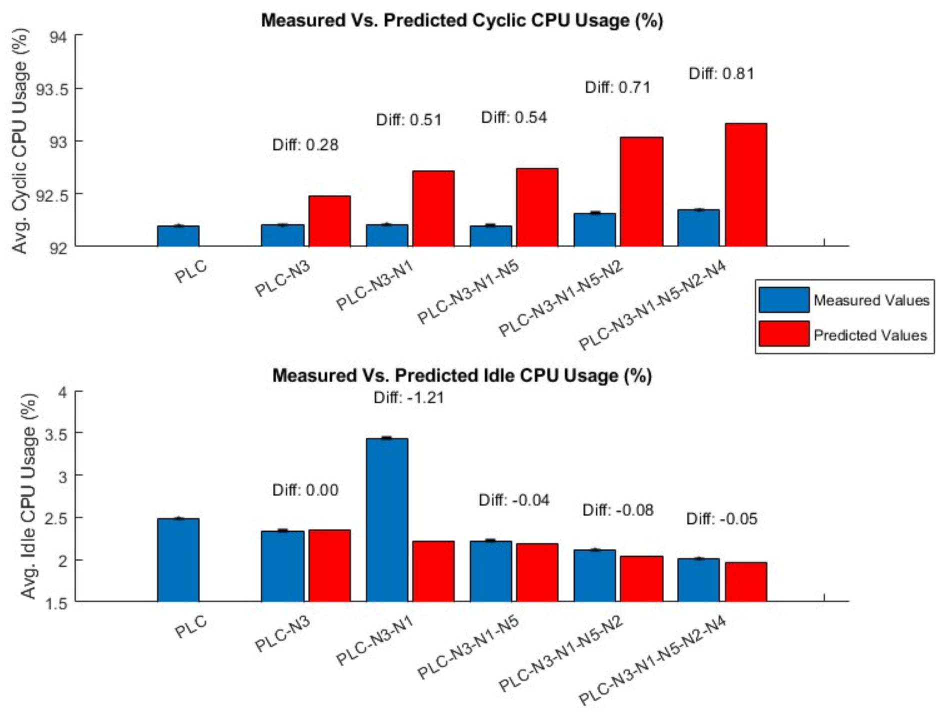


Figure 21: Linear relationship for setup PLC-N3-N1-N5-N2-N4 at “very high” load.

**Ridge regression:**
Medians of data from *IV.II Balanced cyclic vs. idle CPU usage (%)* are split into two groups: modules and nodes. The measurements on which to apply ridge regression are the same as in *V.III Linear relationship*. Starting from the PLC and towards the last device, an array contains '0's or '1's depending on if the device is a node or a module. Ridge regression is applied with either data from modules or nodes using said array. The ridge regression is effectively polynomial regression, with the addition of a regularization parameter $\lambda$ which penalizes large coefficients. The ridge regression coefficients also have individual weighted coefficients.

The predictions made in Figure 22 captures the measured values for the cyclic usage (%) far better than i Figure 21. However, some deviations can be seen such as the dip in CPU usage (%) at device count 20-25. Further suggesting that more data regarding node behavior is required.

Similarly, in Figure 23 it is quite apparent that predictions using ridge regression does not fully capture the fluctuating CPU usage (%) when the system is under "medium" load. However, the linear approximation of the idle usage (%) also struggled here.

At last, in Figure 24 predictions are made for the very same setup as in Figure 19. However, the inaccurate predictions made here suggests that the baseline (measured CPU usage (%) of only the PLC) might have been off. Inserting an offset shows that the predicted behavior of the cyclic usage (%) is still better than the linear approximation, hence the ridge regression is the better choice for predicting cyclic usage (%).

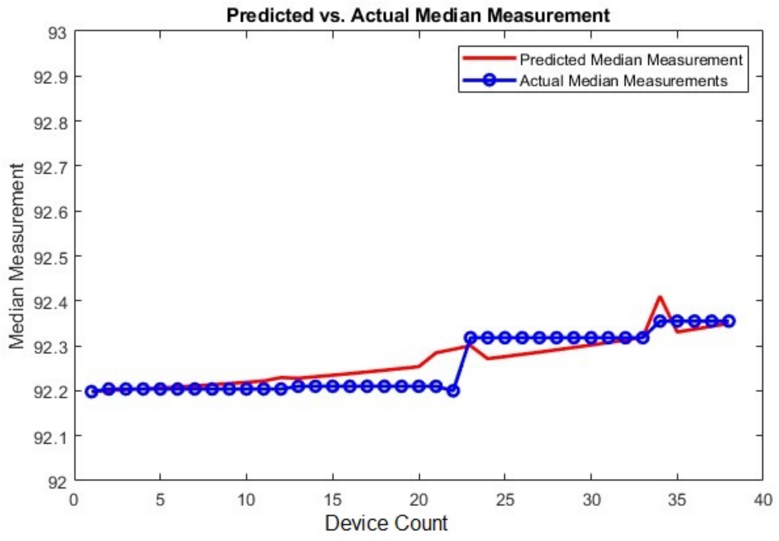


Figure 22: Ridge regression of $2^{nd}$ degree with high penalty factor and weighted coefficients. Same setup as in Figure 21

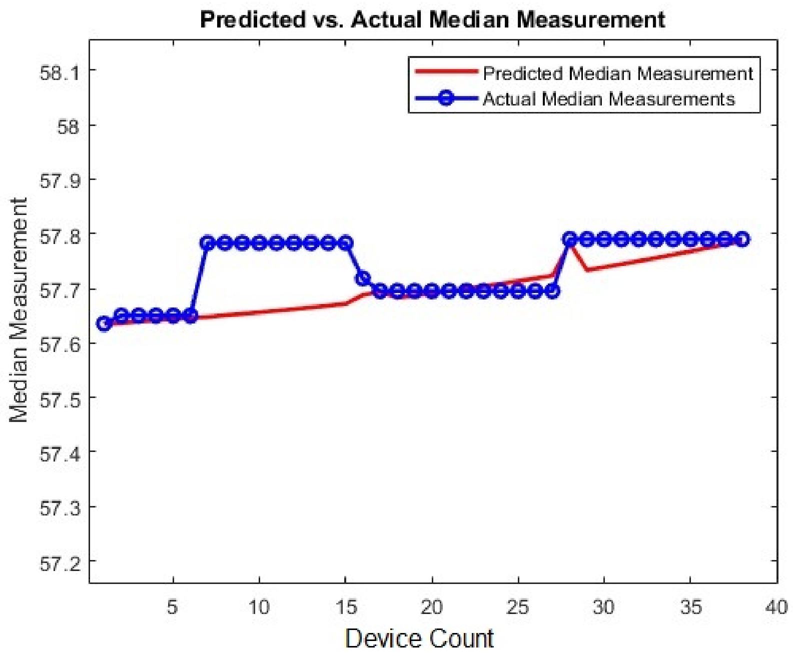
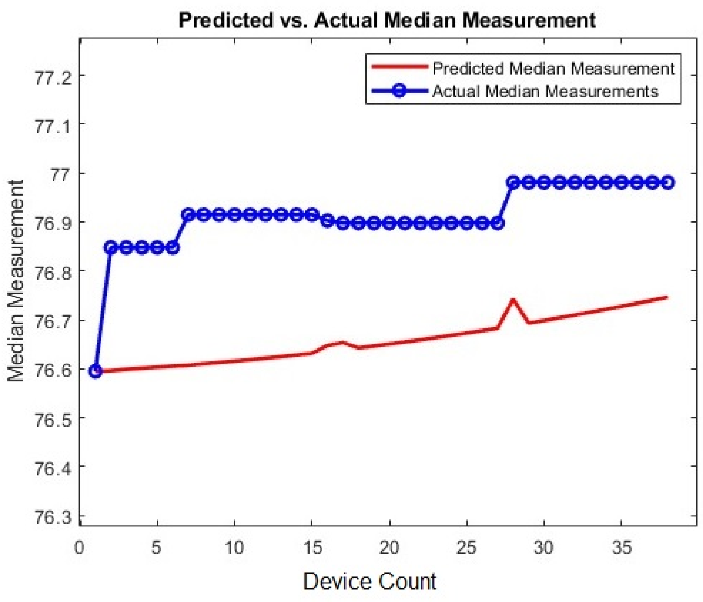


Figure 23: Ridge regression of $2^{nd}$ degree with high penalty factor and weighted coefficients. Same setup as in Figure 18



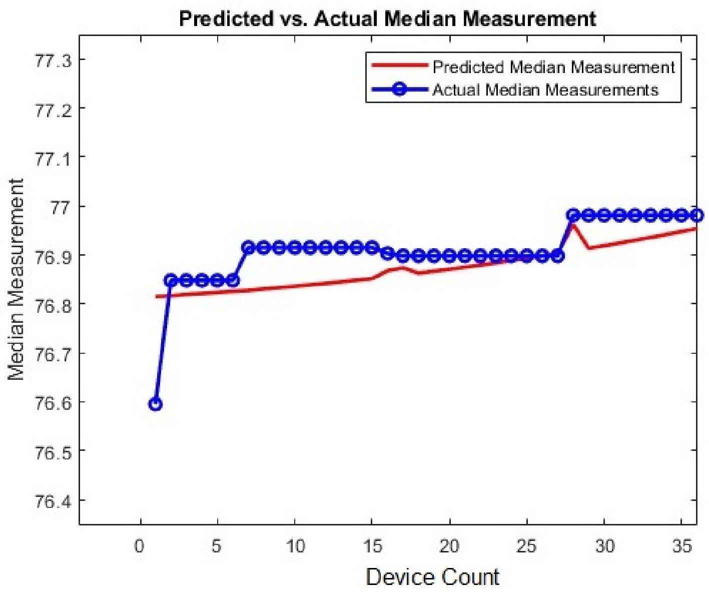


Figure 24: Ridge regression of $2^{nd}$ degree with high penalty factor and weighted coefficients. Same setup as in Figure 19. The figure shows the impact on predictions when the base (PLC) cyclic CPU usage (%) measurement is inaccurate. However, adjusting the baseline and the predictions still hold.

# VI Discussion

This chapter ties the theory to the results and provides the reader with additional context surrounding the model and the work in general.

## VI.I General

While certain real-time controllers can be extended with additional memory or even have the CPU replaced for better performance, PLC systems are limited by their hardware due to hard real-time restrictions. As such, it becomes important to household with the CPU and memory resources to ensure that the system works as intended. However, PLC manufacturers tend not to disclose performance numbers related to the impact of a certain type of device. For the rare case when they do provide benchmarks, it might not be applicable to the consumers intended use. Without these types of guidelines provided by the manufacturer, it is difficult to translate how many more devices that can be incorporated into the system without exhausting hardware resources. Including third-party devices into the system further complicates the matter.

To perform a proper evaluation of a real-time system requires detailed knowledge about the hardware and software, which might not be readily available for off-the-shelves PLC systems. As such, a commercially available PLC system could be approached as a "grey-box", where the inner workings are mostly unknown. The method presented in this thesis consists of measuring incremental changes to the idle and cyclic CPU load (%) when adding a device. The working principle of the method is to exploit the finite amount of time in which the scan cycle operates. If a large amount of CPU time is dedicated to a computationally heavy control program, tasks directly associated with the newly added device will be more prioritized and can potentially become more easily measured.

The developed mathematical model used to predict the performance changes must be applicable to any modular PLC system. The ultimate goal is to allow the consumer to estimate the margin of expansion with regards to the number of devices within the PLC system at some CPU load. A problem with regards to the margin of expansion is the fact that modern PLC systems can often be re-configured to increase performance, although often with some negative side effect. One such example is to make use of multicast packets, which increases communication speed but only allows small amounts of data to be sent. The model used in this work does not make use of system specific performance gains, but instead measures the system at default and maintains a fixed configuration throughout testing. There are disadvantages with this approach, such as the risk of gathering too conservative measurements, resulting in a skewed model.

Both the cyclic and idle CPU usage (%) are common performance indicators provided by CPU performance measurements, i.e., Intel's PMU. The profiling tool provided by B&R measures several other tasks, but without the knowledge of the system's inner workings it is difficult to differentiate tasks tied to the addition of a new device. An interesting observation related to this can be made in Figure 14, where idle usage (%) drops severely, while the cyclic usage (%) remains almost stagnant.

In short, it is apparent that the sum of negative idle change (%) and positive cyclic change (%) does not equal zero. This means that the change in idle usage (%) reflects other tasks not covered by cyclic usage (%) measurements, and vice versa. The idle usage (%) indirectly becomes a measurement for the addition of a device since the scheduler must prioritize between tasks to keep the total allocated CPU time within the scan cycle. Exactly which tasks are prioritized by the scheduler are unknown at this time. An improved model could be derived if one could pinpoint those precise tasks. Meanwhile, the cyclic usage (%) is more well defined as it increases due to changes to the I/O image table, or mapped variables within the control program.

Another aspect to take into consideration when evaluating the impact of a device is the chosen node topology. The method presented in this thesis exclusively uses hub/linear topology which presumably would result in an increase in delay for devices at the end of the line. However, one could argue that delays are inevitable for very large systems, and therefore better represent the real-life impact of a device. Topologies similar to hub/linear are also unavoidably present within larger setups. But it is advised that the method should be tailored towards the topology which the user intends to use.

To be clear, there are many ways to design a method for evaluating the impact of an added device. The method presented in the thesis starts by evaluating the difference between analog and digital modules, and tries to isolate the impact of a single empty node. Starting with the former, the comparison between analog and digital modules is done by cross-coupling the same number of channels on the analog and digital modules. The modules were connected serially to the PLC using the X2X-link communication. While there is a documented difference in delay of 50 µs between the two modules, it is too small to notice when serially connected to the PLC. Hence the outcome might differ if the modules were connected via Powerlink to a node instead. The problem with adding nodes to the system is how to differentiate between time delays related to the node, and the newly attached module itself.

One could also argue that the chosen number of channels used was too restrictive and did not evaluate the device's full potential. The reasoning behind the restrictive use of channels was due to the analog modules only having four channels, and there is arguably little to gain from comparing an unequal number of channels of each signal type. Analog modules were chosen for the remainder of tests primarily to be able to do further comparisons with digital modules in the future, but also since analog modules were specifically required for network testing.

Another aspect which might confuse the reader is the choice of using I/O pairs instead of measuring individual input and output modules. While it may have been an oversight, it entirely stems from the practical aspect of easily allowing input modules to receive an input signal, which is required for the input module to function properly.

Returning to the latter point of evaluation, the impact of a single empty node was done by simply connecting six nodes in linear/hub-topology to the PLC through Powerlink. Admittedly a fault occurred here, a proper evaluation would have been to measure the change in CPU usage (%) for every added node. Instead, several measurements for one node and six nodes were made. It would have been interesting to see the resulting change in delay for every node, which might have shed some light on anomalies in CPU usage (%) such as differences in cyclic usage (%) seen in Table 4.

The next step in the method was to evaluate how nodes and modules together impact the system's performance. As previously stated, there are endless variations to how nodes and modules can be connected. The term "unbalanced" and "balanced" node and module configurations were coined to describe setups with a different number of modules per node, and a close to equal number of modules per node, respectively. The reason for this distinction, instead of measuring setups with a varying number of modules per node, was to simultaneously investigate the impact on network communication. The hypothesis was that nodes which transmit vastly more data related to the unbalance in modules, would perhaps impact the system differently than nodes with an equal number of modules and data. The results in Figure 15 point to the existence of some unbalance in the load between placing the modules at the second or third node. However, it is difficult to draw any conclusions from this single observation as more data on node behavior is needed to assess the previously stated hypothesis.

Another phenomenon observed during testing was the difference between measuring a complete setup in which a device has been removed, compared to measuring a setup in which a device has been added. Compare Figure 14, in which there are large fluctuations in CPU usage (%) when measuring after a device has been added with Figure 13, where the very same setup is measured after a device has been removed. A hypothesis is that the system stabilizes more quickly when removing devices compared to when devices are added. The difference might be due to the lack of time between measurements.

### The results

Looking at section *V.III Model Calculations*, the reader can more easily grasp the conclusions drawn from the results in section *V Results.* Starting with differences between analog and digital modules. In Figure 11 it appears as if the documented difference of 50 µs between analog and digital modules is too small to impact the system in a significant way. Although, it must be noted that the load from the control program on the CPU was moderate. As such, the measured difference might have been greater from a more resource intensive control program. However, it is not the incremental change between additional devices we are interested in, but instead the difference between the two module types. Hence, any significant difference should also be noticeable at moderate load.

Comparing the "balanced" setup in Equation 1 with the "unbalanced" setup in Equation 2, a few conclusions can be drawn. Starting with the "balanced" setup, it appears as if there is a greater change in idle usage (%) compared to cyclic usage (%). This follows the reasoning done in section *III.II RTOS* regarding task priority. The idle usage (%) is the allocated CPU time executing the lowest priority task. Any change to either cyclic usage (%) or other tasks result in a decrease in idle usage (%). If the CPU is heavily strained by a resource intensive control program, the resulting change in idle usage (%) might better reflect the true system impact from an additional device. In addition, idle usage (%) measurements were heavily prioritized over cyclic usage (%) measurements since the measurements remained more consistent.

Both the idle and cyclic usage (%) measurements in Table 3 and Figure 13 seem promising since many of the bars in the cyclic usage (%) appear green (increasing) and conversely in the idle usage (%) where many bars appear red (decreasing). As such, both measurements from the "balanced" setup are valid candidates to use in the model due to consistent results. Make note of the division of the values used in section *V.III Model Calculations* which were done to convert average CPU (%) difference from *per I/O pair* to *per device* instead.

The "unbalanced" setup found in Table 5 tell a different story compared to the balanced setup. While the red bars (third node) in Figure 15 steadily decrease, the blue bars (second node) remain almost stagnant for the same amount of attached modules. However, if we zoom in on the blue bars we can clearly see that they fluctuate, while the red bars continue to steadily decrease. Looking at the error bars the same reasoning seems to hold, thereby strengthening the hypothesis that more accurate results regarding the impact of a device are produced by straining the CPU.

Continuing in section *V.III Model Calculations*, the results for the impact of a single node are derived in Equation 4. The measurements presented here originate from Figure 12. Continuing, the measurements in Equation 5 are from testing nodes and modules together. Notice that the idle usage (%) steadily decreases for each node, while the cyclic usage (%) only increases for every other node. To fully understand this requires more data on the behavior on nodes, further discussed in *VI.II Future Work.* In comparison, the values in Equation 4 appear to have a higher average decrease in idle usage (%) for each added node, but not in cyclic usage (%). This might point to the fact that there is a higher increase in the number of tasks associated with the addition of a node than a module.

In summary, there are discrepancies between the "unbalanced" and "balanced" setup which might be linked to unknown behavior of nodes. The results are also skewed in favor of idle usage (%) due to more consistent measurements, as previously mentioned. Many measurements in which there were anomalies, such as a sudden increase in cyclic or idle usage (%), should perhaps be cautiously used in the model. However, the cyclic usage (%) measurements for the impact of a node, see Table 4, in which an increase could only be seen for every other node was later used in the model. The reason behind that decision was that the cyclic (%) predictions originally heavily overshot, even more so than in Figure 19-21. Hence, the smallest observable values were used for the impact on cyclic usage (%). In section *VI.II Future work* it is further explained how task isolation might help shed some light on why these anomalies occur.

The lack of data on cyclic behavior might be more apparent in section *V.III Linear Relationship*, where idle usage (%) can be more accurately predicted than cyclic usage (%). A test rig with five nodes was used, see Table 1, the CPU usage (%) was measured for each node. The predicted values were calculated using a simple linear relationship, see Table 10. As previously mentioned, the predicted value for cyclic usage (%) heavily overshot the measured value for the test rig. Meanwhile, the linear approximation predicts the change to idle usage (%) very well, as seen in Figure 20-21. It must be said though, that the predictions are only accurate for when the control program heavily loads the CPU, see the right part in Figure 17, where it is clear how neither cyclic (%) nor idle usage (%) are accurately predicted.

An attempt at predicting the cyclic usage (%) can be seen in section *V.III Ridge regression* where ridge regression is used. Based on the samples gathered in *V.I Balanced idle. vs. cyclic CPU usage (%)*, the data was split in two groups; modules and nodes. The ridge regression use a $2^{nd}$ degree polynomial with differently weighted coefficients for the two data groups. The resulting model creates good estimates but fails to capture the entire behavior in CPU usage (%), see Figure 24. As with the Linear Approximation, further data on the node behavior needs to be gathered and used in each model for better idle and cyclic usage (%) predictions.

Turning the attention to other aspects surrounding scalability and predictability, namely the the network and program optimization. In section *V.II Network* it appears as if neither injecting PLK- nor UDP-packets had an impact on the performance of the PLC. The captured packets from Wireshark could unfortunately not be shown in either results due to security reasons. However, no change was detected when observing communication in Wireshark or on CPU usage (%) in Profiler. Most likely the reverse engineering of the PLK communication failed, requiring some parameter change in the payload of the PLK-packet to work as intended. Injecting UDP-packets to the Ethernet port might also have been more successful if it could be verified that the PLC was reconfigured to be actively listening on said port.

The last objective with regards to network analysis was to see if a measured signal output from a module in the Powerlink network would be affected by strain put on the Ethernet network. A sudden increase, known as flooding, in communication on the Ethernet network would be simulated using UDP-packet injection. The idea being that if the CPU could not handle the increase in communication it would point towards there being a direct connection between the Ethernet and Powerlink network. However, no change on the signal from the module could be measured during UDP-flooding. Hence, Powerlink and Ethernet communication appears to be disassociated from each other.

PLK-flooding was also performed while measuring the same signal from the module in an attempt to see if the PLC would be affected in any way. But without proper knowledge of any potential weaknesses in the Powerlink communication the attempt was more speculative and lacked any foundation, and no change on the signal could be measured. In summary it appears as if the network is stable and most likely will handle the mentioned capacity limit of 240 nodes.

Regarding program optimization, a vast topic which should have had more attention in this thesis. Starting with repeatably instantiating a FB, it appears as if it requires smaller and smaller pieces of allocated memory in the heap. This could also have been verified using more precise methods such as the SIZEOF() function. However, the chosen approach also delivered satisfactory results as the allocated heap size consistently changed for every new instance.

Moving on to padding of UDTs, it appears as if it does not significantly impact the CPU load. It instead leads to an increase in packet payload, which might cause unnecessary increase in data traffic in the network. Both avoiding padding bytes and reusing FBs are good practices which can optimize larger systems.

## VI.II Future work

This section evaluates the results and builds on the discussion with suggestions to improve the method and derived model.

### The method

The validity of the model rests on the fact that it only predicts systems of similar size to the test rig. While the idle usage (%) was at time relatively accurate, the confidence in the model lowers for systems with more nodes. More time should have been spent investigating the impact of a single empty node. Specifically trying to find setups where anomalies such as sudden CPU usage (%) changes occur.

The CPU usage (%) does not tell us anything about exactly what is going on inside the system The CPU usage (%) might as well have gone up due to it being stalled waiting for memory access. As such, the model could be improved by isolating tasks running only in the presence of an additional node or module. Doing so might help us to understand which measurements more accurately portray the impact of the device. However, accessing low-level logic information regarding tasks, such as TCBs through the RTOS, or task execution using profiling tools such as Intel's PMU, is not entirely feasible for commercial systems. Hence, investigating specific tasks through trial and error using the profiling tool would be very time consuming, but greatly increase the model accuracy.

As mentioned in the theory, interrupt latency and context-switching time are identified as the largest contributors to increased CPU load due to additional devices. They are directly tied to how VxWorks behaves, and a proper model should include an investigation regarding type of device and how VxWorks interacts with said device.

Finally, some thoughts on the identified distributions mentioned at the beginning of chapter V, i.e., the general pareto and kernel distribution. As mentioned, the assumption is made that all measurements follow the normal distribution due to the low volume of gathered data samples. However, this assumption might be entirely false and should be thoroughly investigated.

Another method of testing how the device impacts the system would be to measure signal response times similar to in section *V.III Program optimization.* By cleverly structuring the control program to output a signal at certain instances in the code, some more insight into signal delays due to the addition of a device could be gained. This method could most likely also be used to investigate systems which do not provide proper profiling tools.

The validity of the model also depends on the chosen network topology. The measurements with hub/linear topology used in the model more or less represent a worst-case scenario. Measuring different topologies might both shed some light on previously seen CPU usage (%) spikes concerning nodes, but also potential performance gains.

While on the subject of network, there might be better solutions regarding both how to modify packets using Scapy, but also how to configure the PLC to be susceptible to UDP-flooding and PLK-packet injection. These types of investigations might pinpoint weaknesses in the network which could limit the device capacity during certain conditions.

It is suggested to properly investigate how the Powerlink messages are constructed, then use the knowledge to continue the attempt at faking additional device communication. The same suggestion can be done regarding UDP-flooding. Starting with investigating which libraries are best suited for leaving the PLC most vulnerable to the UDP-flooding attack.

There are many more aspects to program optimization which can be linked to scalability and predictability not mentioned in this thesis. Most hardware is more than capable of executing tasks and handling network delays within a reasonable scan cycle period. The true limitation often lies in a less than optimized program. It is an understatement to say that the most important future work lies in finding programming conventions that could improve the system's scalability and predictability. The only future work which directly ties to the thesis is continuing to investigate the impact on communication from padding bytes. Mainly because there might exist devices in the network which directly copies the structure of UDTs instead of re-arranging them to have a more optimized structure.

### The model

As briefly mentioned in section *III.VI Developing a predictability & scalability model*, the largest hindrance to an accurate model is quite obviously the number of collected samples. However, there might be corners which could be cut in order to use machine learning methods for the scalability model and probabilistic methods for predictability. For example, more samples could likely be gathered by inserting commands to the profiling tool into the PLC control program itself. This could streamline the gathering of many samples and essentially outsource all the work to the PLC itself.

While the intention of the model was to be as simplistic as possible to increase the applicability to other PLC systems, it must be noted that more time could have been spent investigating other ways of modelling cyclic usage (%). One aspect which was overlooked was to simply find a scaling factor which could perhaps decrease the predicted values for cyclic usage (%).

Given time, there exists excellent ways of predicting CPU usage (%) from machine learning techniques, see section *III.VI Developing a Scalability and Predictability Model.* Combining tricks of gathering large amounts of data from the profiling tool directly in the control program with said techniques could potentially result in better predictions.

## VI.III Difficulties

The largest time hurdles were connecting and reprogramming the PLC system for every test and then waiting for the profiling tool to gather measurements. Since the CPU usage (%) was intentionally set fairly high, it affected the download time of gathered measurements from the PLC to the PC. It required a few attempts to fine-tune the balance between the amount of gathered data to achieve good measurements and the assigned CPU load from the control program. Too few gathered data entries increased the risk of inconsistent measurements, while too many gathered data entries resulted in failure to download from the PLC to the PC due to communication error. In general it took two work days to configure and gather good measurements for every setup, and in some unfortunate cases the setup had to be rebuilt due to some mistake along the way. Another hurdle was the fact that there were no previous studies found which could be used as reference regarding the chosen method and model.

# VII Conclusions

Measuring the impact on performance (CPU usage (%)) when adding a device to a commercial PLC system is time consuming. Primarily due to ensuring that the system is stabilized and configured properly between measurements. The lost time and struggle to get consistent measurements resulted in small sample sizes. Small deviations in CPU load due to unknown hardware and software interactions are enough to pollute the measurements. On top of verifying samples by making the PLC system powerless between measurements, it also took considerable amounts of time to connect, configure, and download measurements from the PLC to the PC.

However, even though the gathered sample sizes were small, they still produced estimates which predict the impact on CPU usage (%) from the addition of a device with decent accuracy. Idle CPU usage (%) was best predicted using Linear Approximation while cyclic CPU usage (%) was best predicted using ridge regression with weighted coefficients. As such, a relationship between number/type of devices and their impact on performance (CPU usage (%)) was established. The mathematical models used could easily be applied to other modular PLC systems since they rely on commonly found CPU performance measurements, i.e., idle and cyclic usage (%).

To counteract measurement distortions the proposed solution is to try and isolate tasks which affect CPU usage (%) measurements negatively, and are disassociated with the addition of a device. Furthermore, the accuracy of the model only applies to systems of similar size connected in hub/linear topology. To improve the model, further measurements of systems with different topologies and greater size should be gathered. Great care should be put into investigating the behavior of nodes, as lack of data on this topic severely impacted the model.

Several assumptions were also made for the model, such as neglecting the differences between input and output modules. A more accurate model should include these types of differences and could also profit from including specific configurations reflecting the intended future use of the system. While the difference in impact on performance between analog and digital modules was deemed too small to impact systems of similar size to the test system, larger systems with more nodes might be affected.

The methods used for evaluating the network limitations did not produce results which pinpoint any weaknesses in neither the industrial Internet protocol (Powerlink) or the Ethernet network. From the results it appears as if the Powerlink protocol supports the specified number of nodes. Manipulating the Ethernet network does not seem to impact the Powerlink network. However, the conclusion is that the chosen methods are incomplete and could be improved to further investigate any weaknesses.

Two programming conventions could be utilized to optimize the memory usage. Datatypes such as FBs should be designed as multi-purpose to save allocated memory. Also, maintaining good structure of UDTs to prevent padding bytes could improve network traffic, but does not appear to improve the CPU usage (%).

## References

[1] E. Parr, *Industrial Control Handbook.* Newnes, 1998. [Online]. Available: https://books.google.se/books?id=JCQfAQAAIAAJ

[2] H. Patel, "Top 20 PLC manufacturers : PLC brands and ranking," *https://instrumentationblog.com/plc-manufacturers-plc-brands/*, 2024.

[3] R. Automation, *CompactLogix 5380 Controllers*, Accessed: June. 15, 2024. [Online]. Available: https://www.rockwellautomation.com/en-gb/products/hardware/allen-bradley/programmable-controllers/small-controllers/compactlogix-family/compactlogix-5380-controllers.html.

[4] S. Tierney, *The PDQ-II: Allen-Bradley's First PLC*, Accessed: June 23, 2024. [Online]. Available: https://theautomationblog.com/pdq-ii/.

[5] A. Group, *Säkerhetsprodukter från BR och ABB*, Accessed: June 18, 2024. [Online]. Available: https://new.abb.com/low-voltage/sv/produkter/maskinsakerhet/br-och-abb.

[6] S. Iqbal, S. A. Khan, and Z. A. Khan, "Benchmarking industrial PLC & PAC: An approach to cost effective industrial automation," in *International Conference on Open Source Systems and Technologies*, Lahore, Pakistan, 16-18 December 2013, pp. 141–146.

[7] A. Canedo, H. Ludwig, and M. A. Al Faruque, "High communication throughput and low scan cycle time with multi/many-core programmable logic controllers," *IEEE Embedded Systems Letters*, vol. 6, no. 2, pp. 21–24, 2014.

[8] B. Vogel-Heuser, J. Fischer, S. Rösch, S. Feldmann, and S. Ulewicz, "Challenges for maintenance of PLC-software and its related hardware for automated production systems: Selected industrial case studies," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Bremen, Germany, 29 September - 1 October 2015, pp. 362–371.

[9] A. Ayub, W. Jo, S. A. Qasim, and I. Ahmed, "How are industrial control systems insecure by design? A deeper insight into real-world programmable logic controllers," *IEEE Security Privacy*, vol. 21, no. 4, pp. 10–19, 2023.

[10] H. Yang, L. Cheng, and M. C. Chuah, "Detecting payload attacks on programmable logic controllers (PLCs)," in *2018 IEEE Conference on Communications and Network Security (CNS)*, Beijing, China, 30 May - 1 June 2018, pp. 1–9.

[11] E. P. S. Group, *Communication Profile Specification, version 1.3.0 (2016)*, Accessed: February 28, 2024. [Online]. Available: https://www.ethernet-powerlink.org/fileadmin/user_upload/Dokumente/Downloads/TECHNICAL_DOCUMENTS/EPSG_DS_301_V-1-3-0__4_.pdf.

[12] BR-Automation, *Powerlink Manual V2.61*, Accessed: April 23, 2024. [Online]. Available: https://download.br-automation.com/BRP44400000000000000735497/b_Powerlink-ENG_V2.61.pdf?px-hash=9f85964db2f6999b8d874a0fc129937b&px-time=1715777851.

[13] BR-Automation', *X2X Manual V4.20*, Accessed: April 23, 2024. [Online]. Available: https://download.br-automation.com/BRP44400000000000000739501/MAX20-en_V4.20.pdf?px-hash=b64d1319d535020db166d42a37a73c1a&px-time=1715778107.

[14] T. D. Juhász, S. Pletl, and L. Molnar, "A method for designing and implementing a real-time operating system for industrial devices," in *2019 IEEE 13th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 2019, pp. 149–154.

[15] P. Hambarde, R. Varma, and S. Jha, "The survey of real time operating system: RTOS," in *International Conference on Electronic Systems, Signal Processing and Computing Technologies*, Nagpur, India, 09-11 January 2014, pp. 34–39.

[16] S. Nejati, S. Di Alesio, M. Sabetzadeh, and L. Briand, "Modeling and analysis of CPU usage in safety-critical embedded systems to support stress testing," in *ACM/IEEE 15th International Conference on Model Driven Engineering Languages & Systems (Models 12)*, Innsbruck, Austria, 30 September - 5 October 2012, p. 759–775.

[17] BR-Automation, *Training module handbook: Automation Runtime (English edition, V30)*, Accessed: April 30, 2024. [Online]. Available: https://www.br-automation.com/sv/produkter/tm213tre30-eng/.

[18] P. Zhang, H. Li, and Z. Gao, "PIL: A method to improve interrupt latency in real-time kernels," in *International Conference on Scalable Computing and Communications; $8^{th}$ International Conference on Embedded Computing*, Dalian, China, 25-27 September 2009, pp. 75–80.

[19] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, "Adaptive runtime response time control in PLC-based real-time systems using reinforcement learning," in *2018 IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Gothenburg, Sweden, 27 May - 3 June 2018, pp. 217–223.

[20] BR-Automation, *Real-time and performance*, Accessed: April 23, 2024. [Online]. Available: https://www.br-automation.com/sv/produkter/software/additional-information/real-time-and-performance/.

[21] J. C. Maeng, J.-H. Kim, and M. Ryu, "An RTOS API translator for model-driven embedded software development," in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*, Sydney, NSW, Australia, 16-18 August 2006, pp. 363–367.

[22] Intel, *Intel Performance Monitoring Units (PMUs)*, Accessed: April 30, 2024. [Online]. Available: https://perfmon-events.intel.com/#.

[23] E. Lee, J. Reineke, and M. Zimmer, "Abstract PRET machines," in *IEEE Real-Time Systems Symposium (RTSS)*, Paris, France, 5-8 December 2017, pp. 1–11.

[24] BR-Automation, *X20AI4622 Datasheet V3.40, p. 14*, Accessed: April 30, 2024. [Online]. Available: https://download.br-automation.com/BRP44400000000000000734063/X20AI4622-en_V3.40.pdf?px-hash=a07ec2991da5cefdbc936661eaaa40c0&px-time=1714479094.

[25] BR-Automation', *X20DI9371 Datasheet V3.40, p. 14*, Accessed: April 30, 2024. [Online]. Available: https://download.br-automation.com/BRP44400000000000000680057/X20DI9371-ENG_V3.23.pdf?px-hash=493509dddac4d9f0e4e66c7e722c8b19&px-time=1714479314.

[26] P. I. A. GmbH, *Introduction to POWERLINK*, Accessed: June 23, 2024. [Online]. Available: https://www.port.de/en/products/ethernet-powerlink.html.

[27] D. C. Plummer, *An Ethernet Address Resolution Protocol*, Accessed: June 23, 2024. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc826.

[28] M. S. Inst Tools, *User Defined Data Types (UDT) – Purpose, Need, Tutorial*, Accessed: June 18, 2024. [Online]. Available: https://instrumentationtools.com/user-defined-data-types-udt-purpose-need-tutorial/.

[29] J. R. IBM Developer, *Memory Access Granularity (2005)*, Accessed: April 24, 2024. [Online]. Available: https://developer.ibm.com/articles/pa-dalign/.

[30] BR-Automation, *Data sheet X20(c)CPx58x*, Accessed: June 18, 2024. [Online]. Available: https://www.br-automation.com/sv/downloads/control-and-io-systems/x20-system/cpus/x20cp3586/data-sheet-x20ccpx58x/.

[31] M. M. Y. Kuo, S. Andalam, and P. S. Roop, "Precision timed industrial automation systems," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, Dresden, Germany, 14-18 March 2016, pp. 1024–1025.

[32] BR-Automation, *Training module handbook: Working with Automation Studio (English edition, V30)*, Accessed: April 25, 2024. [Online]. Available: https://www.br-automation.com/sv/produkter/tm210tre30-eng/.

[33] M. Duggan, K. Mason, J. Duggan, E. Howley, and E. Barrett, "Predicting host cpu utilization in cloud computing using recurrent neural networks," in *12th International Conference for Internet Technology and Secured Transactions (ICITST)*, Cambridge, UK, 11-14 December 2017, pp. 67–72.

[34] H. Zhang, Y. Jiang, X. Jiao, X. Song, W. N. Hung, and M. Gu, "Reliability analysis of PLC systems by Bayesian network," in *IEEE 6$^{th}$ International Conference on Software Security and Reliability*, Gaithersburg, MD, USA, 20-22 June 2012, pp. 283–290.

[35] BR-Automation, *X20 system user's manual*, Accessed: June 18, 2024. [Online]. Available: https://download.br-automation.com/BRP44400000000000000739501/MAX20-en_V4.20.pdf?px-hash=5ef5f652e85e162dd508b378b66c72bd&px-time=1725618510.

[36] Fluke, *Fluke 120B Series ScopeMeter® Industrial Handheld Oscilloscopes*, Accessed: June 18, 2024. [Online]. Available: https://www.fluke.com/fr-ca/produit/test-electrique/oscilloscopes-portables/120b.

[37] W. Foundation, *Wireshark*, Accessed: April 23, 2024. [Online]. Available: https://www.wireshark.org/.

[38] G. Valadon, *Scapy*, Accessed: April 23, 2024. [Online]. Available: https://scapy.net/.