Otto-von-Guericke University Magdeburg

Faculty of Computer Science
Institute for Intelligent Cooperating Systems

Master Thesis

Monitoring and Maintenance of Industry 4.0 Learning Laboratory using Digital Twin and Simultaneous OTA Updates

submitted: November 8, 2023

by: Sri Girish Tangirala, 230061

Advisers:

Supervisor
M. Sc. Vasu Dev Mukku
Institute of Logistics and Material Handling Systems
Otto-von-Guericke University
Universitätsplatz 2
39106 Magdeburg, Germany

Responsible Professor
Prof. Dr. Mesut Günes
Institute for Intelligent Cooperating Systems - Faculty of Computer Science
Otto-von-Guericke University
Universitätsplatz 2
39106 Magdeburg, Germany

Responsible Professor
Dr.-Ing. Tobias Reggelin
Institute of Logistics and Material Handling Systems
Otto-von-Guericke University
Universitätsplatz 2
39106 Magdeburg, Germany

Tangirala, Sri Girish:
Monitoring and Maintenance of Industry 4.0 Learning Laboratory using Digital Twin and Simultaneous OTA Updates
Master Thesis, Otto-von-Guericke University
Magdeburg, 2023.

Abstract

Cyber-Physical Systems (CPS) are computer systems that use computer-based algorithms to control a mechanism. CPS are an integral part of Industry 4.0, using the Internet of Things (IoT) to communicate with each other. One of the major functionalities of the CPS is the ability to update their firmware remotely. This concept is called Over-The-Air (OTA) Firmware Update. Safety-Aware Over-The-Air firmware updates have been implemented and tested in the Industry 4.0 Learning Laboratory [1], built by the ILM department at OvGU Magdeburg, which comprises of factory modules in a static layout. To keep up with the growing diversity of demand, factories need to constantly update their layout to create new manufacturing paths, which leads to a dynamic factory layout. These changes in the factory layout need to be detected remotely, due to a possible unavailability of control engineers at the factory site. The control logic of these dynamic layouts may also need regular updates, making it a time-consuming and tedious process if each factory module must be updated individually. This thesis aims to tackle this problem by introducing simultaneous OTA updates, assisted by a digital twin. Unity Engine is used to develop the digital twin and the physical system is continuously monitored by tracking the manufacturing process using CoAP messages and updated using the SUIT module.

Acknowledgements

Firstly, I would like to thank all those who supported and guided me in completing my master thesis.

I would like to express my profound gratitude to my professors, Prof. Dr. Mesut Günes and Dr.-Ing. Tobias Reggelin for their advice, guidance and addressing all my queries quickly and promptly.

I would also like to thank M.Sc. Vasu Dev Mukku for providing me with this topic, and allowing me to conduct this work at the Industry 4.0 Learning Laboratory at ILM, OvGU. I am thankful to him for his invaluable advice, guidance and suggestions throughout the course of this work.

Last, but not the least, I am grateful to my parents and my brother for their love, support and encouragement, which enabled me to complete this work.

Upon submitting this thesis, my long term association with Otto-von-Guericke University Magdeburg will come to an end.

Contents

Abstract

1 Introduction and Motivation
1.1 Motivation . . . 8
1.2 Aim of this thesis . . . 8
1.3 Structure of this thesis . . . 9

2 State of the Art
2.1 Background Technologies . . . 11
2.1.1 RIOT OS . . . 11
2.1.2 CoAP . . . 13
2.1.3 Over-The-Air Updates . . . 14
2.1.4 Digital Twin . . . 16
2.1.5 Factory Planing Laboratory . . . 17
2.2 Related Work . . . 18
2.2.1 OTA Updates . . . 18
2.2.2 Digital Twin . . . 20
2.2.3 Factory Planning Laboratory . . . 21
2.2.4 Digital-Physical System Pair . . . 22

3 Thesis Contribution
3.1 Conceptual Model . . . 24
3.2 Implementation . . . 25
3.2.1 Hardware Setup . . . 25
3.2.2 Software Setup . . . 27
3.2.3 Implementation in Factory Planning Laboratory . . . 30

4 Thesis Evaluation
4.1 Experiment 1: Measurement of Simultaneous Firmware Update Efficiency . . . 35
4.2 Experiment 2: Measurement of System Efficiency . . . 38

5 Conclusions and Future Work
5.1 Summary . . . 42
5.2 Future Work . . . 43

A List of Figures

B List of Tables

C Listings

D Bibliography

Acronyms

BLE Bluetooth Low Energy. 12, 22, 23

CoAP Constrained Application Protocol. 13–16, 22–24, 27, 32, 35, 43, 45

CPS Cyber-Physical Sytems. 7, 18

CPU Central Processing Unit. 11

ILM Institute of Logistics and Material Handling. 25, 30, 42

IoT Internet of Things. 7, 11, 13, 15, 18–20, 22, 23

Li-Fi Light Fidelity. 22, 26, 30

MCU Microcontroller Units. 8, 9, 12, 26

OTA Over-the-Air. 4, 7–9, 14, 15, 18–20, 22, 27, 35, 42

SUIT Software Updates for Internet of Things. 15, 16, 35

Wi-Fi Wireless Fidelity. 12, 35

1 Introduction and Motivation

The industrial revolution which started in the late 18th century was the cornerstone of the automation of manufacturing processes. Going through many changes, the current generation of production processes has arrived at what is termed the fourth industrial revolution, better known as Industry 4.0. The aim of Industry 4.0 is to digitize the entire manufacturing process with minimum human intervention [1]. The rapid change in technological growth and smart automation has given rise to heavily interconnected networks of production layouts that can be simulated using Cyber-Physical Sytems (CPS). CPS are computer systems that use computer-based algorithms to control a mechanism. CPS are an integral part of Industry 4.0, using the Internet of Things (IoT) to communicate with each other.

One of the major functionalities of the CPS is the ability to update their firmware remotely. This concept is called Over-the-Air (OTA) Firmware Update. Safety-Aware Over-the-Air firmware updates have been implemented and evaluated in the Industry 4.0 Learning Laboratory built by the ILM department at OvGU Magdeburg, which is comprised of factory modules in a static layout [2]. This thesis aims to build on the work carried out by them.

To keep up with the growing diversity of demand, factories need to constantly update their layout to create new manufacturing paths, which leads to a dynamic factory layout. These changes in the factory layout need to be detected remotely, due to a possible unavailability of control engineers at the factory site. This calls for continuous monitoring of the factory modules remotely, and representation of the factory in a virtual environ-

ment. The control logic of these dynamic layouts may also need regular updates, making it a time-consuming and tedious process if each factory module is updated individually. This thesis aims to tackle these problems by developing a digital twin and introducing simultaneous OTA updates.

1.1 Motivation

Simultaneous OTA update for multiple devices is still an area of research in many industries. It is imperative for those with tens or hundreds of deployed Microcontroller Units (MCU) to explore this technology, as it can help save a large amount of time and resources.

A digital twin can assist this process by constantly being connected to the factory layout and working in tandem with it so that any changes in the manufacturing process can immediately be detected and updated.

The combination of a digital twin and simultaneous OTA firmware updates is an asset to any industry as it supports both the surveillance of the system and its maintenance remotely. Many such systems have already been implemented in various industries such as automobiles, healthcare devices, and agricultural sectors. Current research in Industry 4.0 within the context of factory processes is majorly limited to either tracking or maintenance, as described in the upcoming sections. The process of continuous monitoring of a factory layout has been theorized but there is a definite lack of a comprehensive demonstration of the monitoring, maintenance, and update of a factory layout.

1.2 Aim of this thesis

Manufacturing processes are constantly being changed to meet the demands of the consumer market. To meet the requirements of the ever-changing methods, processes, and products, the industry needs to constantly monitor the ongoing processes and update them whenever necessary.

The aim of this thesis work is to provide the proof of concept for the above-described theory, by developing a physical-digital system pair and incorporating the concepts of Industry 4.0 such as real-time process updates and Over-The-Air Firmware Updates. The factory planning laboratory in the Institute of Logistics and Material Handling is utilized for implementation. Initially, a digital twin is developed using the available factory modules as the reference. Each factory module is equipped with a Nucleo-F767ZI MCU board connected to an ESP32 for wireless connectivity, and a combined layout of conveyors, sliders, and turntables makes up the physical system. A CoAP server is used for status updates and storing the firmware images and is hosted on the same network as the digital twin. The physical system sends updates to this server over the network using CoAP messages. Each module is given the provision for an OTA firmware update, and all modules can be simultaneously updated through WiFi using the functionality provided by the Digital Twin.

The system is evaluated based on simultaneous firmware update efficiency, speed, scalability, and process efficiency. Initially, each module is individually updated to calculate the update time for a single module. Following this, the entire system is updated using simultaneous updates through the digital twin to demonstrate the scalability of the system and measure the proportional increase in update speed. Finally, the process efficiency of the system is measured by tracking the uptime of each module and comparing it with the total uptime of the system.

1.3 Structure of this thesis

This thesis is structured as follows.

- Chapter 1 gives an introduction to the proposed work and the motivation behind it, as well as the aim of the thesis.
- Chapter 2 provides the theoretical background of the different concepts put together in this thesis and outlines some of the related work that has been done in this field.

- Chapter 3 describes the methodology of the thesis and provides a description of the different hardware and software setup utilized in this work.
- Chapter 4 presents the evaluation and provides a proof of concept for the work described in this thesis.
- Chapter 5 summarizes the work and delves into some concepts that can be explored in the future.

2

State of the Art

This chapter provides background on the various technologies used in the development of the proposed physical-digital system pair and describes the recent developments related to the Industry 4.0 features that have been provided in this system.

2.1 Background Technologies

2.1.1 RIOT OS

RIOT is a real-time operating system (OS) capable of multi-threaded operation, which supports a wide range of memory-constrained devices such as 8-bit and 16-bit microcontrollers and focuses on low-power wireless IoT systems. It is an open-source OS and is well-documented [3]. It is based on a microkernel architecture and the software programming is generally implemented in C and C++. The microkernel architecture is well-suited for devices with restricted availability of non-volatile memory, as it provides nothing more than the minimum amount of software required for the functioning of the operating system.

The kernel is hardware-independent and contains the basic functionality of the OS such as memory management and communication between various internal processes. The CPU and board-specific configuration files are stored in separate directories, with corresponding sub-directories for each individual CPU and board.

The RIOT architecture can be better described using Figure 2.1.

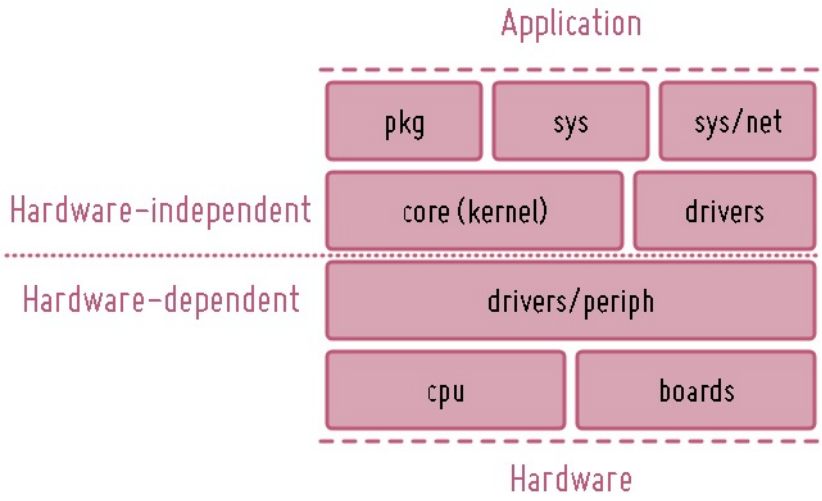

Figure 2.1: RIOT Structure [3]

The applications built on RIOT OS for this thesis are compiled by GNU Make. It is typically used to compile source code into programs or libraries. The main element of the Make utility is the Makefile. The Makefile defines the rules and targets that Make uses to compile the specified source code. The action which is carried out by Make is called a recipe.

RIOT OS is a modular system, in which all features are implemented in separate entities, conveniently named as modules. These modules are built on top of the kernel and can be added individually in user applications. The modules are generally compiled using Make, but other compilers such as Valgrind are also supported.

There are several modules already developed and maintained by the RIOT community. Two such modules utilized in this thesis are SUIT, which provides functionality for Over-The-Air firmware updates, and GNRC Border Router, which can provide an interface between an IPv6 network such as the Internet and a mote device such as an MCU. The SUIT module, combined with the GNRC border router, can be configured to update the device firmware over wireless interfaces such as Bluetooth Low Energy (BLE), IEEE 802.15 or IEEE 802.11 (Wi-Fi).

2.1.2 CoAP

The Constrained Application Protocol (CoAP) is a lightweight, machine-to-machine communication protocol, defined in RFC 7252 of the IETF [4]. It was developed and standardized by the Constrained RESTful Environments (CoRE) Working Group of the IETF, and expanded by adding various new functions in order to make the protocol suitable for IoT and M2M applications. It uses datagram-based transport protocol such as UDP, which makes it lightweight and desirable for low-power applications and networks such as 6LOWPAN.

CoAP protocol is a client-server-based model that follows a request-response method, very similar to the widely used HTTP protocol. However, unlike HTTP, machine-to-machine interactions in CoAP can lead to both machines taking up the role of a client and a server based on the use case. A CoAP request is equivalent to that of HTTP and is sent by a client to request an action (using a Method Code) on a resource (identified by a URI) on a server.

Four message types are available in CoAP: acknowledgment, reset, non-confirmable, and confirmable. This protocol includes message codes that identify the type of communication sent. CoAP responses are typically accompanied by Acknowledgment messages, while CoAP requests are typically sent as Confirmable or Non-confirmable messages. Safety-critical messages are transmitted using confirmable (CON) messages to guarantee dependability. They are sent with a message ID, and until the recipient sends an Acknowledgment message (ACK) with the same Message ID, they are retransmitted with a default timeout and exponential back-off between retransmissions. For non-critical communications, such a single reading from a continuous sensor data measurement, non-confirmable (NON) messages can be utilized. NON messages do not carry an acknowledgment (ACK), but they do have a message ID that can be used to identify duplicates. Retransmission can be enabled by a recipient sending a Reset (RST) message in the event that it is unable to process a request message.

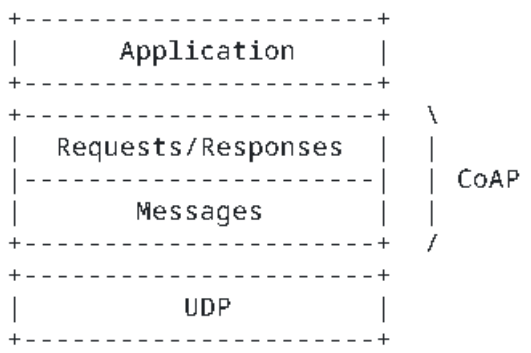

Figure 2.2: Abstract Layering of CoAP Protocol [4]

As shown in Figure 2.2, CoAP can be rationally described as a two-layer protocol, with the message codes serving as the data layer and UDP serving as the asynchronous messaging layer. On the other hand, CoAP is a single protocol that just includes the message and the Method or Response code in its message header.

2.1.3 Over-The-Air Updates

Over-the-Air (OTA) firmware updating is the process of providing wireless updates to the system of a terminal device, typically carried out over the internet. OTA updates for the firmware of the microcontrollers allow for remote maintenance and secure operation of the MCU.

OTA updates have become increasingly common in recent years, as they offer a convenient way for device manufacturers and developers to deliver new features and fix bugs without requiring users to physically connect their devices to a computer. This can be particularly useful for devices that are not easily accessible, such as those that are embedded in vehicles or other equipment.

OTA updates are typically delivered in the form of a package that contains the update files and instructions for installing the update. The device's operating system or a dedicated update application is responsible for downloading and installing the update. Some OTA updates may require the device to be restarted in order to complete the installation process.

There are several advantages to using OTA updates [5]. For device manufacturers, they can reduce the cost of distributing updates, as they do not have to rely on physical media or require users to bring their devices to a service center. For users, OTA updates can be more convenient, as they do not have to go through the process of connecting their device to a computer in order to receive the update. Additionally, OTA updates can ensure that devices are kept up to date with the latest security patches and other important updates.

However, there are also some potential drawbacks to OTA updates. If the update process is interrupted, it can result in a failed update, which may cause the device to become unstable or even inoperable. Additionally, some OTA updates may require a significant amount of data to be downloaded, which can result in higher data usage and longer update duration.

Overall, OTA updates offer a convenient and efficient way for device manufacturers and developers to deliver updates to their products and have become an important part of the modern device ecosystem.

SUIT, short for Software Updates for Internet of Things, is a firmware update infrastructure developed for RIOT OS by the IETF Working Group [6]. The SUIT specification is a standard for securely delivering updates to IoT devices, and it is designed to ensure that updates are authenticated, integrity-protected, and delivered in a manner that is resistant to attacks. It enables secure firmware updates for low-power IoT devices using the Constrained Application Protocol (CoAP).

The SUIT module in RIOT allows developers to build devices that are capable of securely receiving and installing updates Over-the-Air (OTA), without the need for a physical connection to a computer. This can be particularly useful for IoT devices, as it allows them to be updated remotely and without the need for user intervention.

The SUIT module in RIOT is implemented using a combination of cryptographic techniques, such as digital signatures and hashing, to ensure the authenticity and integrity of updates. It also includes mechanisms to

allow devices to verify the identity of the update server and to detect any tampering or modification of the update package.

The SUIT module in the RIOT codebase resides in the sys/suit subdirectory in which the storage of the firmware image on the flash memory of the board and transport of the image file through CoAP is implemented.

2.1.4 Digital Twin

A digital twin is a virtual replica of a physical system, designed to monitor the physical system using various sensors and interact with it using different communication protocols. The concept of a digital twin is a high-level construct, with many interpretations depending upon the area of application and the industry in question. As such, a digital twin can be anything from a simple block description of a process to a complex virtual simulation of an industry.

In the context of Industry 4.0, a digital twin can be used to monitor a manufacturing process by obtaining sensor data regarding various aspects of the process such as conveyor speed, position, occupation, and other such parameters. This data can then be relayed to the software model to run simulations or conduct various tests for performance improvements, all of which can then be used to manipulate the physical system to improve productivity.

There are multiple softwares available for out-of-the-box applications of a digital twin, that is, they can be installed and can be immediately put to use by creating virtual models of the physical modules and developing the required communication paths between them. After careful consideration of the available softwares and the required features, Unity3D has been selected for the purpose of developing a digital twin for this thesis.

One of the main features of Unity is its ability to import and work with assets from a variety of programs, such as 3D modeling software, audio editing software, and image editing software. This allows developers to easily incorporate a wide range of media into their projects. In addition to its asset import capabilities, Unity also has a robust set of tools for designing

and building virtual environments, including a visual scripting system and a physics engine. These tools allow developers to create complex and interactive simulations. The 3D models for the factory modules used in this thesis have been designed in an open-source 3D modeling software called Blender, and the asset import feature of Unity has been used to incorporate these models in a simulation. The various object manipulation tools in Unity have allowed for motion simulation of the modules to reciprocate the motion of the physical system.

Unity also has a large and active community of developers, with a wealth of resources and support available online. This includes a comprehensive manual, tutorials, forums, and assets.

2.1.5 Factory Planing Laboratory

The Factory Planning Laboratory built by the Institute of Logistics and Material Handling at OVGU Magdeburg is a prototypical implementation of the logistics of a manufacturing unit [7]. The laboratory consists of factory module prototypes which can be used to simulate manufacturing paths and processes such as production, storage, and dispatch of a product. The modules that are available in the laboratory can be categorized into four types, which are, conveyors, turntables, sliders, and pushers.

Conveyors can be used for lateral movement in one dimension without any rotation. Turntables can provide rotation and provide lateral movement in one dimension. Sliders can provide two-dimensional lateral movement. Pushers can be used to ensure the proper path of the product or discard unnecessary products by pushing them into or out of the plane of the modules.

The modules in the Factory Planning Laboratory are designed by Fischertechnik [8] and can be assembled manually from parts provided by them, which makes it easier to develop a modular factory model with several options for customization. Initially, Arduino Mega 2560 was used as the controller board for these modules in the laboratory, but for the

purpose of network connectivity and multi-threading in this thesis, it was replaced by Nucleo F767ZI board.

2.2 Related Work

This section describes the current research in the literature regarding each individual facet of this thesis and then describes the current state of the art of the Digital-Physical system pair.

2.2.1 OTA Updates

Updates for CPS are relatively rare, especially when compared to personal computers or mobile devices. For example, in automotive systems, the frequency of updates is very low and mostly available only through dealerships. However, observations of recent trends suggest growth in automotive OTA updates.

Rapid advancement in Internet of Things (IoT) technology gave rise to increased research on the methods and processes of Over-the-Air firmware updates for IoT devices in the past few years. A few relevant systematic analyses are presented here for context. Firstly, an overview of the necessity of OTA updates is presented. Then, some methods of OTA updates are discussed and compared.

Bauwens et.al. presented an excellent approach to defining some key requirements of OTA updates [9]. In their study, an analysis is performed concerning the distribution of software development efforts in different parts of widely used IoT operating systems. The software modules are divided into four blocks, applications, network, core OS and platform. Memory usage and Git commit history for each block are calculated for different IoT operating systems. These calculations lead to the conclusion that application software for most IoT systems occupies much less memory and requires much less update frequency when compared to the network protocol stack. It is calculated that 60% to 84% of the firmware updates are related to the network protocol stack. Hence, the authors have concluded

that without network protocol updates, it is not possible to guarantee the optimal performance of an IoT device during its operation lifetime.

After analyzing the need for OTA updates, some OTA update methods are presented here. In a journal article published by Halder et. al., research has been conducted on OTA updates in the automotive industry, focusing mainly from a security perspective [10]. They discuss the key security issues, challenges, and requirements necessary for OTA updates and provide detailed background and comparison on various secure OTA update techniques available in the current works of literature. Though the majority of this study focuses on the automotive sector, the update techniques described in the study provide a solid foundation for further analysis. Some of the techniques are summarized below.

- Symmetric Key Encryption: A set of link keys are shared between the Software Supplier, OEM, and the device. When a software update is needed, the Software Supplier uses a link key as a symmetric key with the device, encrypts the software, and sends it to the device for updation [11].

- Hash Function: The binary file of the software to be updated is divided into multiple data fragments. Each fragment is encrypted using a pre-shared encryption key and transmitted to the device [12].

- Blockchain: All devices participating in the update process are grouped into a cluster, with a designated cluster head. The cluster heads are connected in an overlay network to form a hierarchically connected system without a central controller. The Software Distributor triggers an update process by sending a store request to the cloud server. After receiving an acknowledgment from the cloud, the Software Distributor uploads the new software to the cloud, creates a transaction in a Blockchain block indicating the location of the latest software in the cloud, and broadcasts it to all the devices in the cluster[13].

- Hardware Security Module: A Hardware Security Module is used to store a cryptographic key and perform encryption during a software update. In this method, a gateway device downloads the software update from a remote server and validates it. If the validation is successful, it transfers the software to the target device. The advantage of using this technique is that it supports multiple encryption methods such as AES, RSA, SHA, etc. [14].

A novel method of OTA update has been proposed by Guissouma et.al. where they proposed an approach covering the entire Software Development Life Cycle (SDLC) of an IoT system [15]. An important part of their work is their special attention given to variant and configuration management. Initially, three types of updates are defined

- Adaptive Updates: Updates that are performed to introduce new functionality into the system.
- Corrective Updates: Updates that are performed for error corrections and bug fixes.
- Perfective Updates: Updates that are performed for optimization of processes.

The method used is called a contract-based design, where a contract is a mathematical construct defined as a tuple $C = (A,G)$. The assumption A stands for a condition on the environment of a component M and the guarantee G describes a property that the component guarantees to satisfy if A holds. The OTA update process is triggered by the violation of a contract.

2.2.2 Digital Twin

Current developments of digital twins in the industry have been described by Tao et al.[16]. They reviewed the most relevant theories in their paper, which can be divided into four parts as Modeling, simulation, verification, validation, and accreditation (VV&A); data fusion; interaction and collaboration; and service. They have further summarised the industrial

applications of Digital Twins, as being used majorly in Product Lifecycle, for patents and applications by industry leaders.

The dimensions of digital twin applications have been summarised by Enders and Hoßbach[17]. They described six dimensions which are the industrial sector, purpose, physical reference object, completeness, creation time, and connection. In their paper, they stated that the manufacturing industry is one of the major fields where digital twins are applied, and the three main purposes of the digital twin are stated as simulation, monitoring, and control. These dimensions are the cornerstone for the implementation of the digital twin in this thesis.

Vijayakumar et al. developed a Digital Twin for factory system simulation in [18]. The aim of this work is to develop a new methodology for reducing the amount of time and cost of keeping the manufacturing facility updated. They have used Plant Simulation for the development of the simulation model and Autodesk Navisworks for the visualization of the factory. The results are promising, with a 96.6% reduction in the monitoring process. However, this system is unidirectional, in essence, only uses the Digital Twin to monitor the system, but does not provide a firmware update feature.

Wahab et al. [19] proposed a project to design and develop an AR Aid as an alternative learning tool training guidance for the operational setup of an electrical discharge machine. The key feature of their work is the use of Unity software to develop the Digital Twin and Blender software to develop the animations, both of which are used in this thesis,

2.2.3 Factory Planning Laboratory

The Factory Planning Laboratory at OVGU Magdeburg has already been utilized for experimentation of various factory environment simulations over the course of two years. Initially, the storage of products in a storage matrix is simulated in [20], where a High-Bay Shelf was assembled and utilized for experimentation. Later, a complete factory layout with separate production, storage, and dispatch zones was developed to provide proof of concept for different wireless communication methods between the

factory modules. Different wireless communication protocols such as Li-Fi, Bluetooth, UART, and RF are implemented to demonstrate inter-protocol connectivity and modularity of the system. [20]

2.2.4 Digital-Physical System Pair

Grznár et al. [1] discussed the future of manufacturing systems with regard to technological innovation and factory planning. They described that digitization and the application of exponential technologies are extremely important to CPS. The manufacturing systems in the future will be designed as modular, reconfigurable and independent systems. To keep up with such a system, it is absolutely essential to develop a Digital Twin in constant communication with the physical system, which is termed here as a Digital-Physical System Pair.

Within the context of manufacturing, there have not been many instances of a combination of a digital twin with OTA firmware updates in the literature. The closest resemblance to the work proposed in this thesis is conducted in the BASE MoVE research project by J. Akelbein et al [21]. This project provides a modular base platform for IoT applications. A proof-of-concept is provided by implementing the platform on a home automation system.

The BASE MoVE project used RIOT OS for its modularity, debug support, and support of an open compiler to create a sensor node hardware platform with support for OTA updates using the SUIT module.

This project has also been equipped with multiprotocol support, where IPv6 protocols such as 6LoWPAN and BLE are included. The native 6LoWPAN stack in RIOT OS was utilized. BLE, which is included in different smart home products and almost all smartphones, was implemented using the open-source stack called nimBLE which is currently integrated into RIOT OS. Additionally, the Thread protocol was also included in this group, where they used the OpenThread implementation due to its portability to RIOT OS. The IP transport protocol that was used in this project is the Constrained Application Protocol (CoAP), as it is relatively lightweight and uses UDP. Initially, they used the CoAP implementation from RIOT OS to

manually map the sensor data provided by their devices to a REST structure. Some limitations such as compatibility and interoperability led to the development of a second approach using a prototypical implementation of a subset of Dotdot, which is a universal language of IoT to enable devices to work together on a network. Due to limitations of documentation and the unavailability of open-source specifications, this implementation was only restricted to sensor polling over CoAP and 6LoWPAN.

The platform was evaluated by developing a home automation system, where a house was retrofitted. The sensors were attached to the windows and doors to provide data of the state, either open or closed. Two scenarios S1 and S2, respectively named Ambient Assisted Living scenario and Smart Home scenario were evaluated within multiple rooms, using various protocols mentioned before.

Finally, J. Akelbein et al concluded that their approach for an IoT solution is feasible. They reported that RIOT OS has served their purpose well, with clear signs of improvements in the future. After careful consideration of the conclusions provided in their paper, the use of OpenThread protocol for this thesis was rejected as they reported memory leaks and energy consumption issues. However, their documentation on wireless implementation is not satisfactory. Although they mentioned the use of BLE initially, they did not include it in their final evaluation due to a lack of support. There was also no consideration of a WiFi-based implementation, although the present generation of home automation systems almost always includes WiFi.

To summarize, the necessary background of the different technologies used in this thesis is provided and recent developments in those technologies are outlined. In light of the described works and their drawbacks, this thesis aims to accommodate the features that proved to be proper additions to a factory layout and omit the remaining features. The following chapter provides a conceptual model of the proposed work and details the implementation of the factory layout.

3

Thesis Contribution

3.1 Conceptual Model

Figure 3.1 shows the conceptual model of the system in the form of a block diagram.

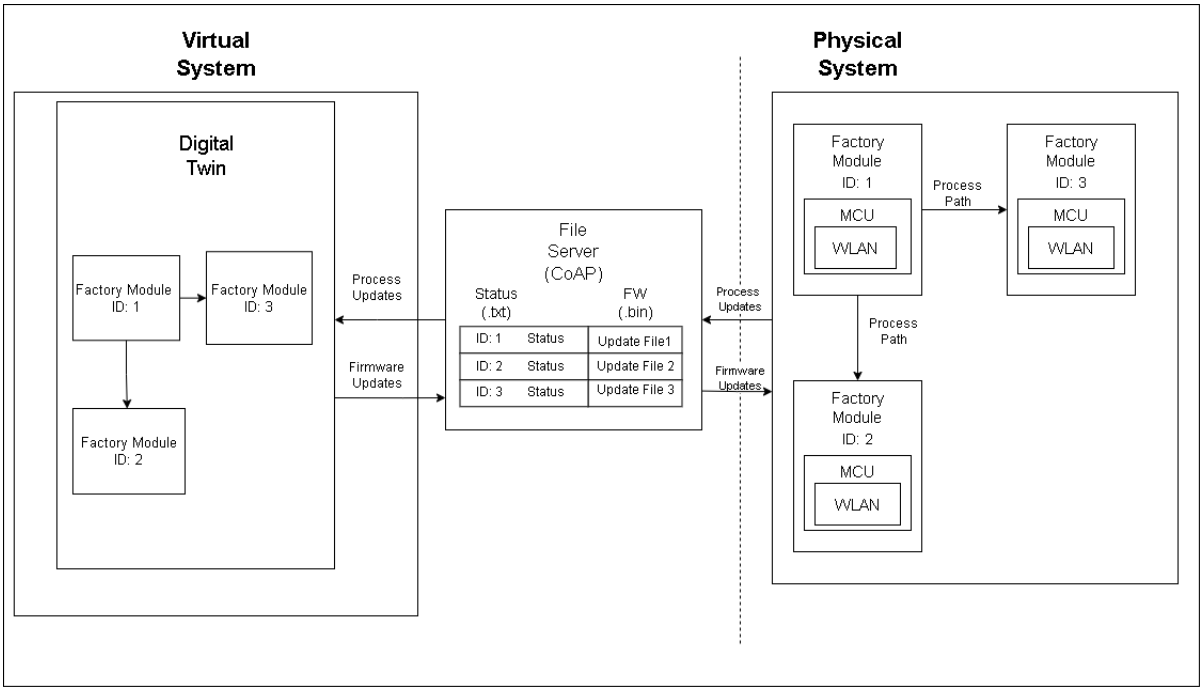

Figure 3.1: Conceptual Model

The system is divided into a physical and a virtual system, which is linked by the Linux PC hosting the CoAP fileserver. The physical system is replicated in the virtual system and tracked by reading the process updates from the

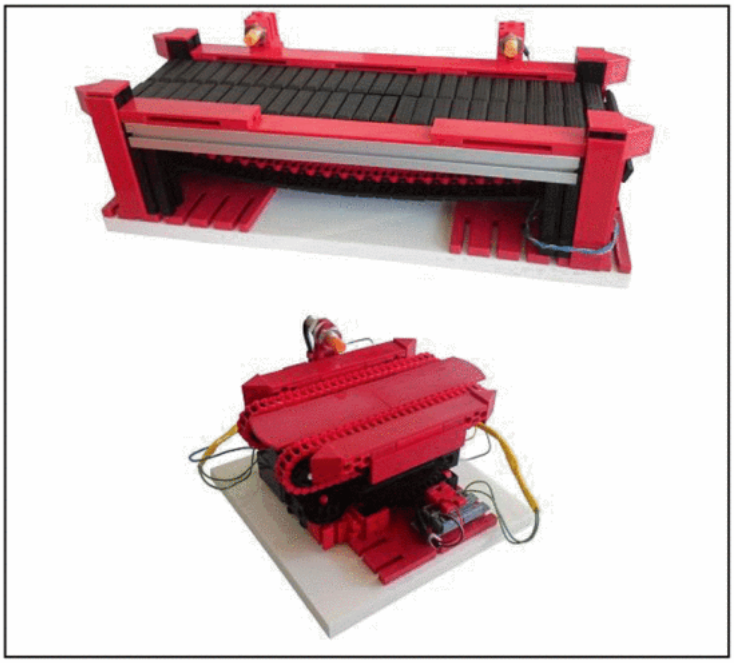

Figure 3.2: Conveyor and Turntable [2]

fileserver. Similarly, the physical system is updated by accessing the update files from the fileserver and flashing onto each module simultaneously.

3.2 Implementation

This section elaborates on the hardware system, the software configuration, and the interfaces between them to provide a detailed description of the system which has been set up as a proof of concept for the theory described in Chapter 1.

3.2.1 Hardware Setup

Factory modules such as conveyors (Figure 3.2), sliders (Figure 3.3), and turntables (Figure 3.2) which are available in the Factory Planning Laboratory of the Institute of Logistics and Material Handling (ILM) department at Otto-von-Guericke University, are utilized for experimentation purposes of this thesis.

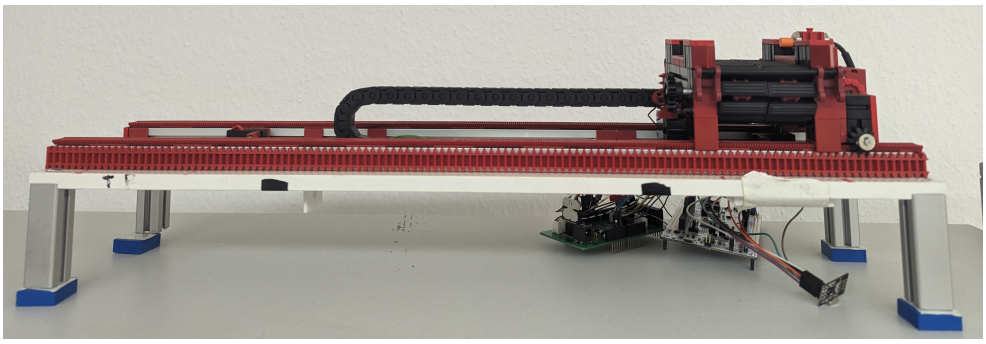

Figure 3.3: Slider

After careful research on different MCU based on the requirements of this work, ARM-based STM32F767ZI Nucleo-144 mbed-enabled development boards (Figure 3.4) are selected as the controllers for these factory modules. Some of the features of these boards include a 32-bit ARM-based STM32 controller, up to 512kBytes of SRAM, up to 216MHz of operating frequency, and a 144-pin layout with multiple application-specific hardware add-on capabilities. The STM32 boards boast an impressive number of peripherals when compared to boards like the PIC32, and also can be much more reliable when running power-demanding applications.

One of the drawbacks of the Nucleo F767ZI board is the lack of wireless connectivity, as there is only an ethernet port to connect to a network. The use of an ESP32 board can cover this shortcoming by running a border router application on it and connecting it to the Nucleo board using UART, as shown in Figure 3.5.

The modules communicate with each other using Li-Fi and RF communication. Li-Fi communication is achieved using the Li-Fi protocol[23], with each module containing an LED and an LDR as shown in Figure 3.6. The LED can emit light pulses at a preset frequency while the LDR can convert these pulses to an analog value, which can be later converted using an ADC to a digital value to be processed by the microcontroller. Meanwhile, RF communication is achieved by nRF24L01+ transceiver modules which can be seen in Figure 3.7. Each nRF24L01+ can be configured either as a transmitter or a receiver. Each module has 6 communication channels, so

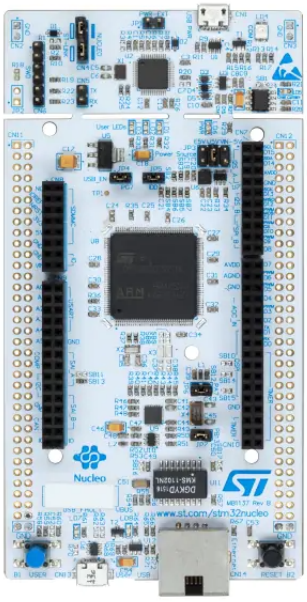

Figure 3.4: Nucleo-F767ZI Microcontroller Board [22]

each transmitter module can send data simultaneously to 6 different receiver modules. RIOT OS contains well-documented drivers for nRF24L01+ including test applications, and hence it makes a perfect addition to this system.

3.2.2 Software Setup

RIOT OS is used as the operating system for its advantages in multi-threaded operation, well-defined OTA update mechanism, and real-time updates to the server. CoAP is used as the communication protocol as it has many advantages such as being lightweight, UDP-based, and thoroughly implemented and documented in the RIOT OS architecture. C programming language is used to program the factory modules on top of the RIOT kernel.

Unity3D is used as the platform to develop a visual interface for the digital twin. The 3D models of the factory modules have been designed in an open-source modeling software called Blender. The background code is

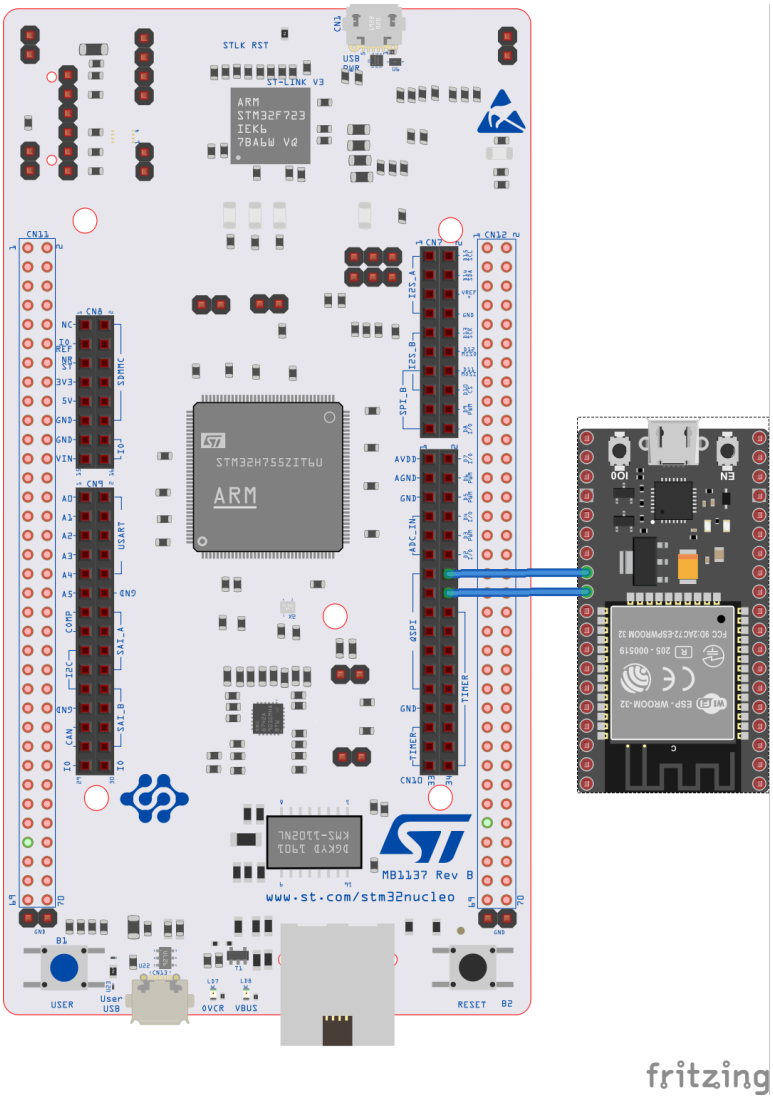

Figure 3.5: Nucleo-F767ZI - ESP32 Connections

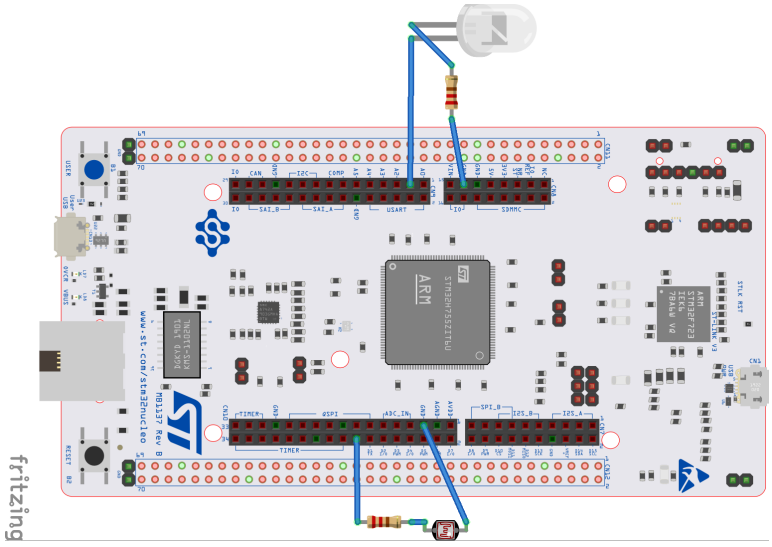

Figure 3.6: Nucleo-F767ZI - LiFi Connections

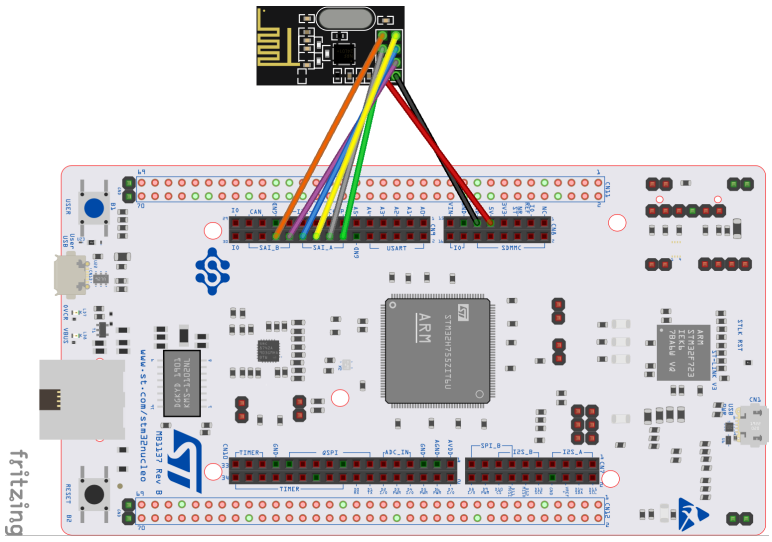

Figure 3.7: Nucleo-F767ZI - NRF24L01+ Transceiver Connections

written in C# language which includes the programs to create a network, create the necessary interfaces for communicating with the hardware, and publish the firmware image files to the server.

The circuit schematics of the Nucleo-F767ZI and its various connections are designed using Fritzing software [24]. The conceptual model of the system is sketched using the Draw online tool [25]. The Sequence diagram and the Activity diagram in Chapter 4 are drawn using the Lucidchart online tool [26], and the graphs are plotted using Matlab software [27].

3.2.3 Implementation in Factory Planning Laboratory

As mentioned, the Factory Planning Laboratory in the ILM Department at OvGU Magdeburg is utilized for the implementation of the system. The entire system is divided into two parts, namely Production Zone and Dispatch Zone. The Fischertechnik factory modules are each equipped with a Nucleo-F767ZI board for motor, sensor, and switch control. Each module in the Production Zone can communicate with the neighboring module using Li-Fi communication, and each module in the Dispatch Zone can communicate with its neighboring module using RF communication.

Two types of networks are set up and experimented separately. For the wired network, a local IPv6 network is set up in the Linux host PC using the "setup_network" shell script pre-written in the tools/ethos subdirectory of the RIOT codebase. By running the following terminal command from the RIOT base directory, a TAP interface named "prodA" is created and the prefix "2001:db8::/64" is setup for the local network.

Listing 3.1: Linux Command for Wired Network Setup

```
sudo dist/tools/ethos/setup_network.sh prodA 2001:db8::/64
```

After the network setup, each factory module is added to the network by creating a separate TAP interface for each module and adding a route to the network through the module's link local IPv6 address, using the following commands.

Listing 3.2: Linux Commands for ethos setup for each module

```
sudo ip tuntap add prodB mode tap user ${USER}
sudo sysctl -w net.ipv6.conf.prodB.forwarding=1
sudo sysctl -w net.ipv6.conf.prodB.accept_ra=0
sudo ip link set prodB up
sudo ip a a fe80::1/64 dev prodB
sudo ip a a fd00:dead:beef::1/128 dev lo
sudo ip address add 2001:db8::1/128 dev prodB
sudo ip route add "2001:db8::/64" via fe80::2 dev "prodB"
```

The commands are similar for creating each TAP interface, except that the name "prodB" is replaced with the corresponding name of the factory module. These commands are automated and run when the Digital Twin application is executed for the first time.

The Factory Planning Laboratory is not connected to a network providing an IPv6 address on its WiFi interface. During the course of this work, several attempts were made to provide a workaround as seen in [28]. Since there is no DHCPv6 server providing dynamic IPv6 addresses on the network, a static address needs to be provided for each of the modules on the network, which can be provided automatically to each module via the ESP32 router through SLIP, based on the physical address of the module. Due to inadequate documentation of the wireless setup in the SUIT module of the RIOT codebase, two bugs were found in the source code during this setup and solved as seen in [29] and [30]. After each module is provided with a unique static IPv6 address, it must be routed to the WiFi interface of the Linux PC that hosts the file server. A successful connection to the host can be confirmed with a reply to a ping command from the host to the node.

Two layouts of factory modules are planned for experimentation in this work, as seen in Figure 3.8 and Figure 3.9. The layouts are physically assembled in the laboratory and designed virtually in the Digital Twin. The modules are programmed to transport workpieces through a pre-defined path unique to each layout and tracked through the simulated model. The Unity model reads the process updates from the file server and replicates it in the simulation. The Key Performance Indicator (KPI) for this setup is the uptime of the module which is displayed in the simulated model for each

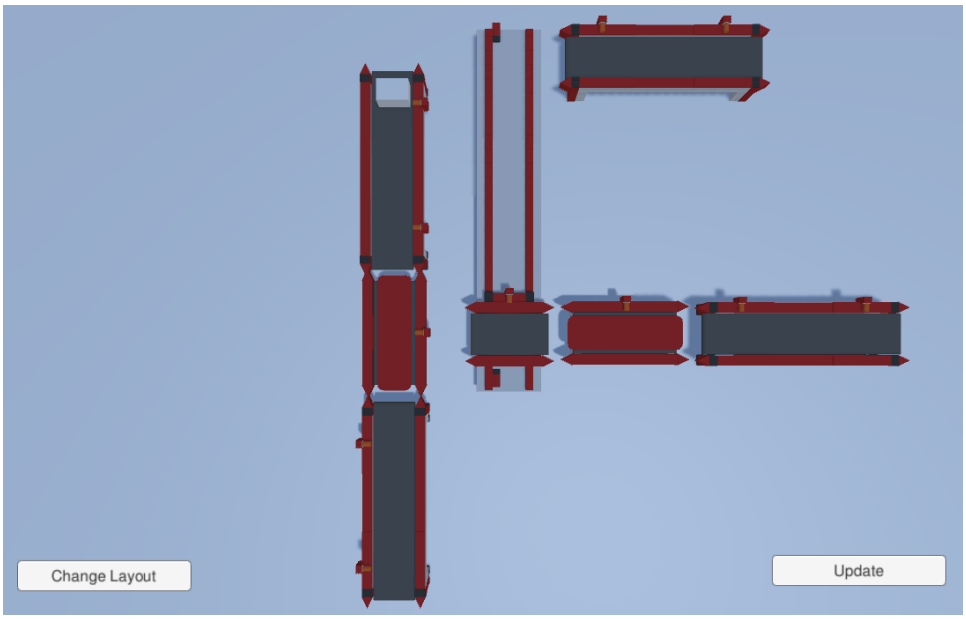

Figure 3.8: Simulation of Factory Layout 1

individual module, and the total active time of the process is recorded and displayed.

Three individual threads run simultaneously in each module: a CoAP server thread with the highest priority to handle incoming and outgoing CoAP messages which are used for tracking and firmware updates, a module-motion thread to read the sensor data, switch states, and control the pre-defined motion of the modules, and a communication thread to send and receive data from neighboring modules. As shown in Listing 3.1, the threads have different memory sizes and are increased or decreased depending on the changing layouts and their process requirements.

Listing 3.3: Memory Allocation for Threads

```
static char _nanocoap_server_stack[THREAD_STACKSIZE_DEFAULT +
    THREAD_EXTRA_STACKSIZE_PRINTF];
char module_motion_thread_stack[THREAD_STACKSIZE_MAIN];
char rx_handler_stack[THREAD_STACKSIZE_MAIN];
```

The working of the system can be depicted by the sequence diagram as shown in Figure 3.10. As shown in the figure, the network is setup and the

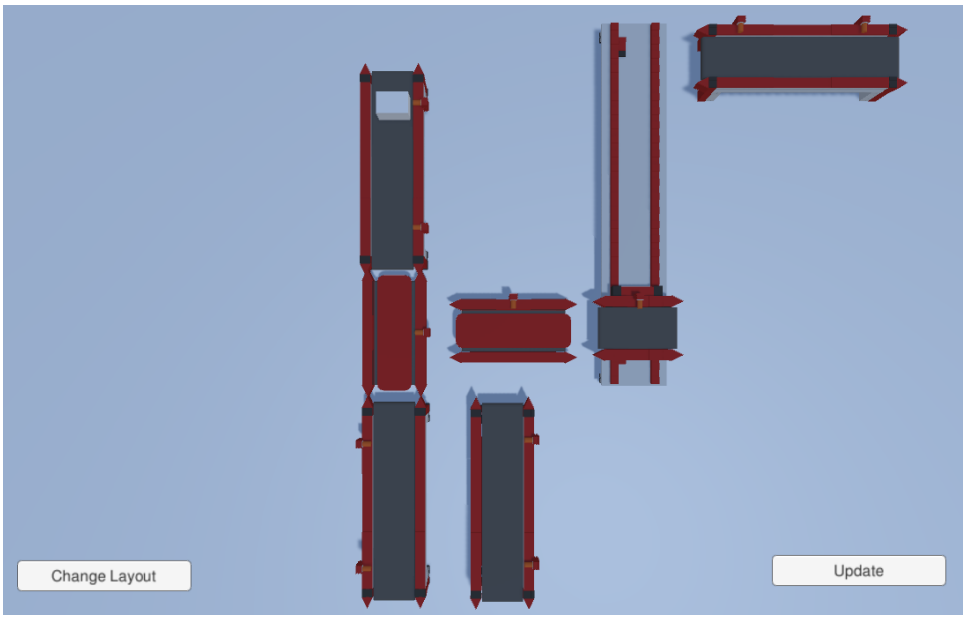

Figure 3.9: Simulation of Factory Layout 2

modules are routed to the network through the TAP interfaces. The Digital Twin is run and the process starts by placing a workpiece on the first sensor of one of the conveyors in the Production Zone, namely Conveyor A or Conveyor B. The workpiece moves across the Production Zone through the Turntable. In Layout 1 (Figure 3.8), the workpiece then moves through the Slider which transports it to Dispatch Conveyor A through the Dispatch Turntable, or to Dispatch Conveyor B directly. In Layout 2 (Figure 3.9), the workpiece moves to the Turntable which then transports it to Dispatch Conveyor A directly, or to Dispatch Conveyor B through the Slider. These decisions are made by sending a message from each module to its neighboring module regarding the source of the workpiece. Each module also sends tracking information to the file server which is replicated in the Digital Twin. At any random time, if there is new firmware to be updated to each module, an update can be triggered by the user to the Digital Twin, which in turn triggers each module to notify the update.

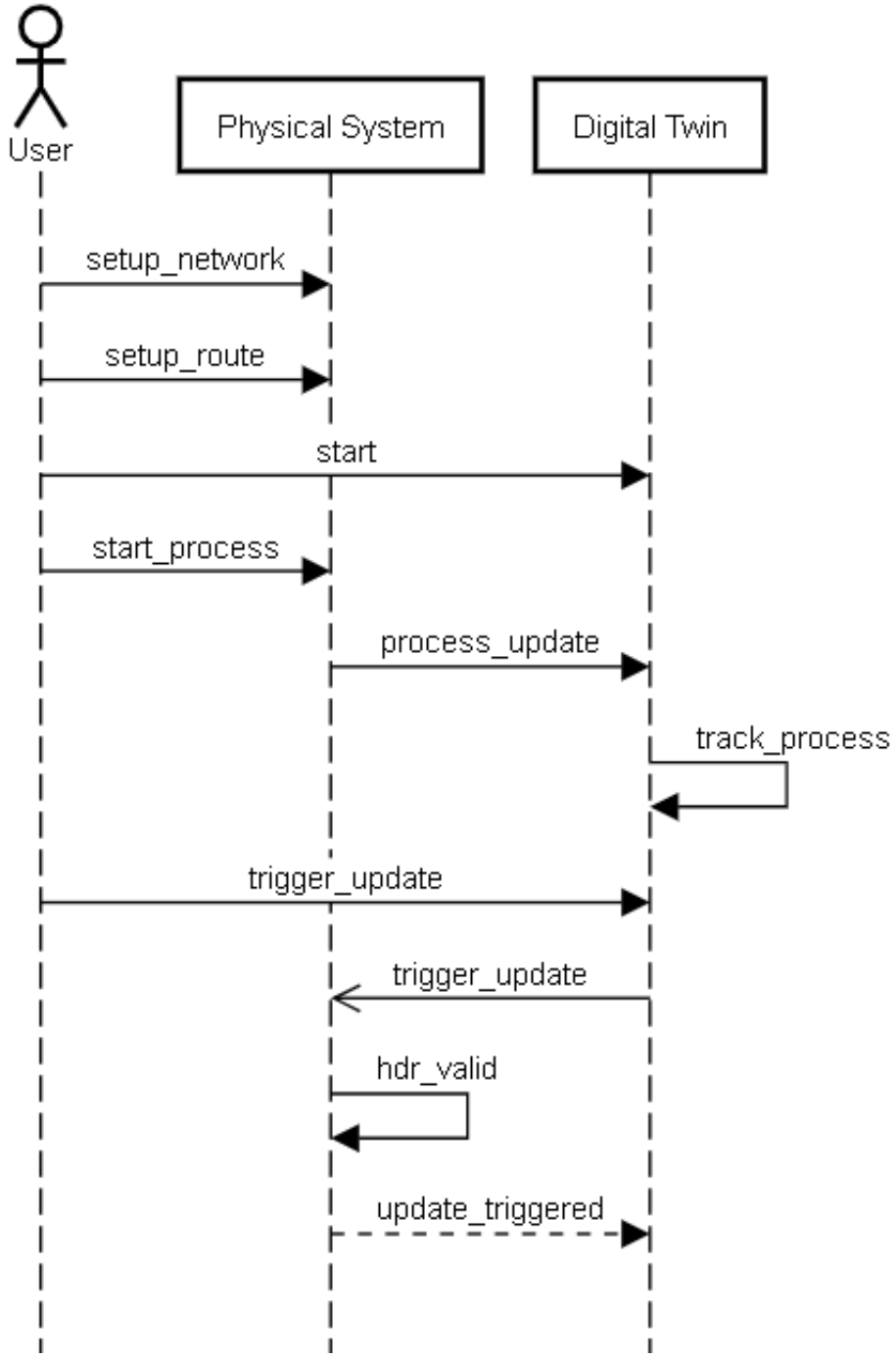

Figure 3.10: Sequence Diagram depicting the working of the system

4

Thesis Evaluation

This section provides the details of the experiments conducted on the system described in Chapter 3 and the analysis of the results. Two experiments are conducted on the system with a focus on obtaining information on the increase in the speed of the firmware update and efficiency of the system using uptime and downtime of the modules. The Digital Twin is used extensively for this purpose, as it displays the majority of the KPIs needed for this analysis.

4.1 Experiment 1: Measurement of Simultaneous Firmware Update Efficiency

The SUIT framework in RIOT OS allows the MCU to be updated using multiple methods such as Ethos, SLIP, and Wi-Fi. In this experiment, each module is initially updated through the network individually, by running the commands in the Linux terminal. These commands publish the updated firmware to the CoAP fileserver and notify the module through the network interface about the new firmware to be installed. The process is elaborated through an Activity Diagram depicted in Figure 4.1.

This experiment is performed both using a wired network connection with Ethos running over the serial link of the modules to the Linux host PC, and a wireless network connection (OTA) using Wi-Fi. The results are very similar with regard to speed of update and hence are tabulated together.

Each module is updated individually for 50 iterations. The update time in seconds is measured and tabulated. Each individual module is then

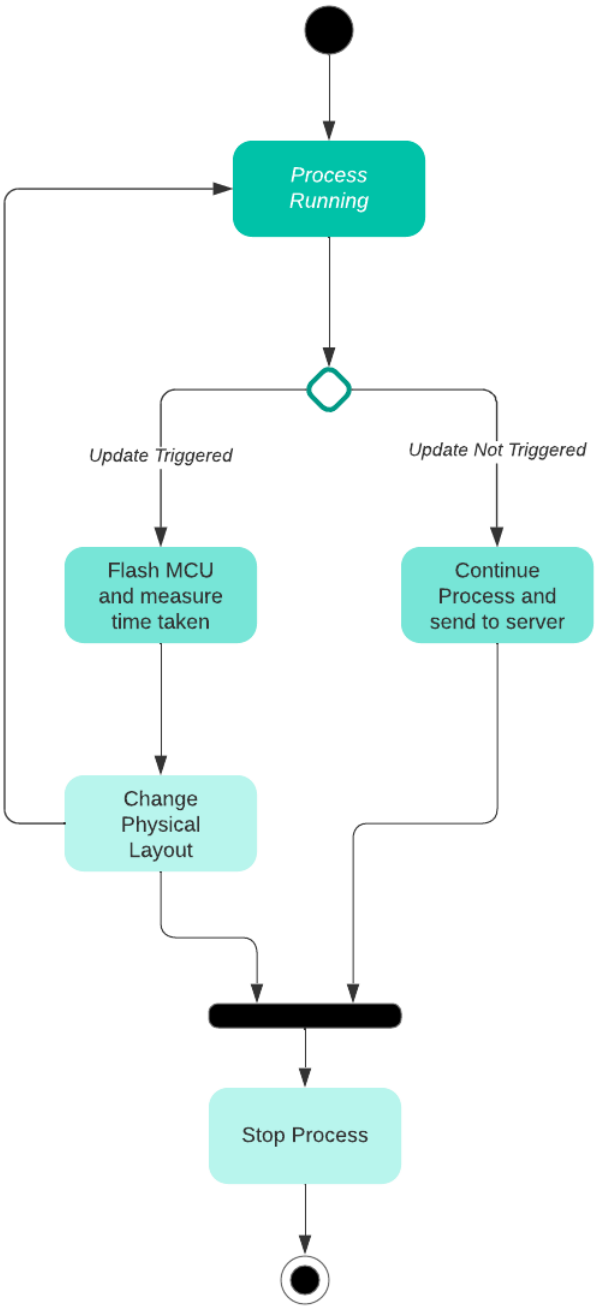

Figure 4.1: Procedure for Experiment 1

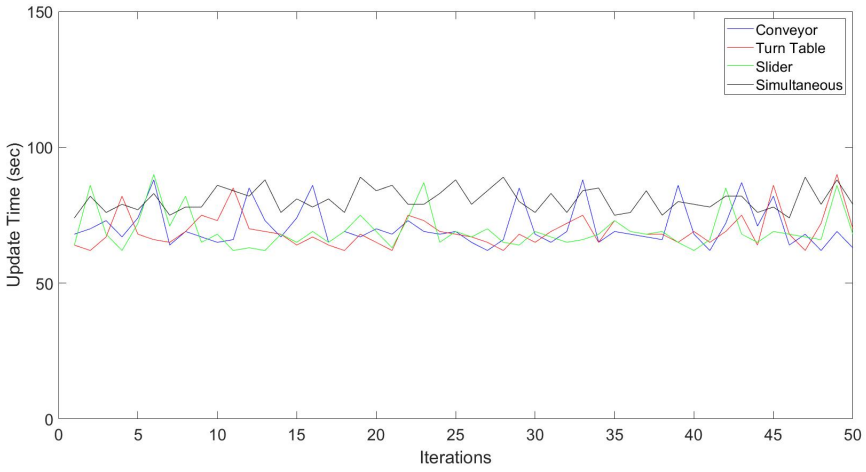

Figure 4.2: Plot of Update Time for 50 Iterations

added to the network by adding a route from the interface to the network address. For the wireless network, the physical address of the module can be routed to the network. For the wired network, a TAP interface is created for each of the modules and a separate route is added from each module to the network through the TAP interface. All modules are updated simultaneously through the Digital Twin and the update time is measured and tabulated. The results for both tabulations are plotted as shown in Figure 4.2.

The average time taken for the firmware update of each individual module, measured as an average of 50 iterations, is displayed in Table 4.1. The total update time for both the individual module update and the simultaneous update is noted as shown in the equation below. Since there are 7 modules in each layout, the equation for individual updates adds the time taken for 4 conveyors, 2 turntables, and 1 slider.

Module	**Time (sec)**
Conveyor	70.62
Turntable	68.92
Slider	69.22

Table 4.1: Factory Module Update Speeds

$$\textit{Time Taken for all modules by individual update } (t_{ind}) =$$
$$\textit{Sum of averages of update time for all modules } (t_{ind\_avg})$$
$$t_{ind} = \sum(t_{ind\_avg})$$
$$t_{ind} = 4 \times conveyor + 2 \times turntable + 1 \times slider$$
$$t_{ind} = 489.54 \text{ sec}$$
$$\textit{Time Taken for all modules by simultaneous update } (t_{sim}) =$$
$$\textit{Average of update time for simultaneous update } (t_{sim\_avg})$$
$$t_{sim} = 80.64 \text{ sec}$$
$$\textit{Increase in update speed due to simultaneous update} = \frac{t_{ind}}{t_{sim}}$$
$$\textit{Increase in update speed} = \frac{489.54}{80.64}$$
$$\textit{Increase in update speed} = 6.07$$

The results show that the simultaneous firmware update leads to a 6.07 times increase in update speed, while also proving that the simultaneous update speed increases proportionally with the increase in the number of factory modules added to the layout.

4.2 Experiment 2: Measurement of System Efficiency

The Digital Twin displays the Uptime of each factory module when it is in motion and the total running time of the entire system which can used to calculate the process efficiency of the system. This experiment is conducted by running the system continuously for 30 minutes, with each workpiece being transported individually. Each workpiece is produced as soon as its predecessor is dispatched to emulate continuous processing. The working procedure is depicted in the activity diagram in Figure 4.3.

The uptime of all the modules after 30 minutes of running the process can be observed in the Digital Twin as shown in Figure 4.4. The timings are tabulated in Table 4.2.

The total time taken by the workpiece to be transported from production to dispatch is called Cycle Time. The measured Cycle Time of the system

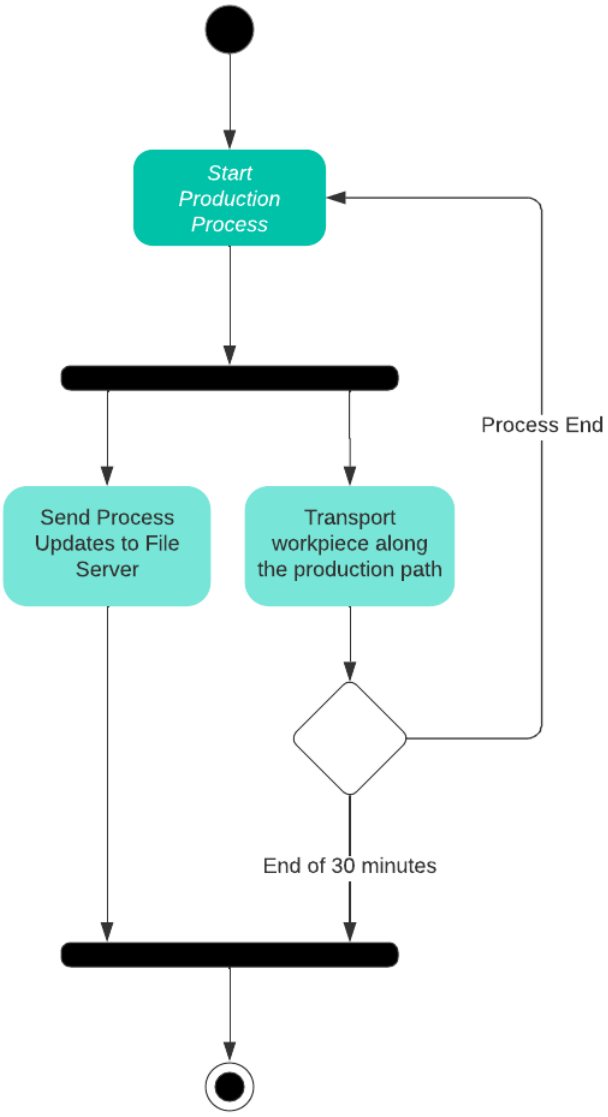

Figure 4.3: Procedure for Experiment 2

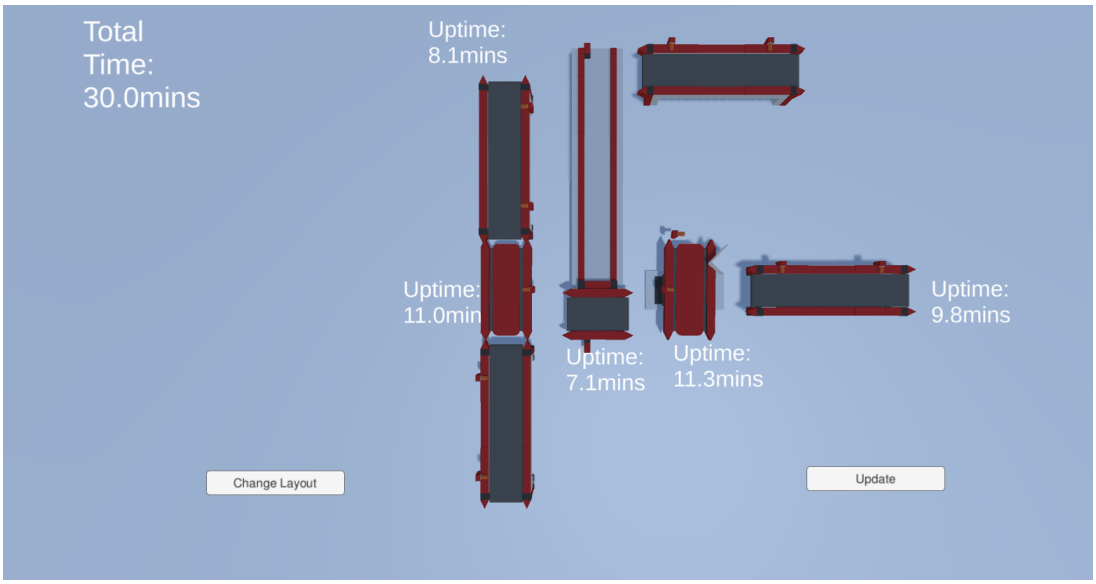

Figure 4.4: Runtime of individual modules

Module	**Time (min)**
Production Conveyor	8.1
Production Turntable	11.0
Slider	7.1
Dispatch Turntable	11.3
Dispatch Conveyor	9.8
Total Running Time	30.0

Table 4.2: Runtime of modules and total time

can be calculated by the sum of the runtime of all involved modules in the process. The ideal Cycle Time of the system, when the system runs at 100% efficiency, is the total runtime of the system displayed in the Digital Twin. The efficiency of the system can be measured using these values as shown in the equations below.

$$\textit{Measured Cycle Time of the System } (t_{sum}) = 47.3 \textit{ mins}$$

$$\textit{Ideal Cycle Time of the System } (t_{ideal}) = 30 \textit{ mins}$$

$$\textit{System Efficiency} = \frac{t_{ideal}}{t_{sum}} \times 100$$

$$\textit{System Efficiency} = \frac{30}{47} \times 100$$

System Efficiency = 63.42%

The efficiency of the system for the experimented layout (Layout 1) is calculated to be 63.42%. The moderate efficiency can be attributed to multiple factors such as the offset of the sensor placement on the modules and increased usage of the hardware over several years resulting in decreased motion speed of the factory modules. The efficiency can be improved by methods such as increasing the power input to the modules and conducting regular hardware maintenance or replacement.

These experiments provide conclusive proof of the feasibility of developing such a system and its advantages in the context of manufacturing processes. Experiment 1 proves the efficiency of the simultaneous firmware updates by comparing individual module update time to simultaneous update time. In Experiment 2, the workload of each module is measured and efficiency is calculated which is useful to find out which modules are being overused or unused, and can be used to avoid bottlenecks and calculate potential alternatives for production paths.

5

Conclusions and Future Work

5.1 Summary

Rapid developments in Industry 4.0 led to the utilization of the Internet of Things for smart factories which resulted in the addition of several new features such as parallel threading, digital twins, and firmware updates. Firstly, these features were brought together in this work to provide a proof of concept for a Digital-Physical system pair. The concepts of Digital Twin and Firmware Updates for embedded systems were introduced.

Secondly, a thorough literature survey was conducted to understand the current state of the art for such systems, and the research gap was described, reaching the conclusion that individual firmware update methods are used by several industries, or the update files are broadcasted to an entire network of devices. However, there exists a research gap on simultaneous OTA firmware update methods. Furthermore, limited research exists on the concept of the Digital-Physical system pair introduced in this work, which can be extremely useful in many industries in general, and the manufacturing industry in particular.

Thirdly, the system described for this thesis was developed using the Factory Planning Laboratory at the Institute of Logistics and Material Handling at OvGU. The hardware and software components of the system, as well as the combination of the physical system with the digital twin to conduct monitoring and maintenance of the system, were described.

Finally, two experiments were conducted to provide proof of feasibility and measurability of the system. In Experiment 1, the increase in the firmware

update speed of the modules due to the simultaneous update method was measured. In Experiment 2, the efficiency of the system for Layout 1 was calculated by comparing the runtime of each module to the total time.

5.2 Future Work

The work carried out for the development of a physical system and its associated digital twin in this thesis has a few limitations that can be addressed in the future. One of the limitations is the lack of a true remote digital twin. The digital twin demonstrated here runs on the local Linux host running a CoAP file server. This can be improved by providing connectivity to the Internet using DHCPv6 prefix delegation to the ESP32 WiFi nodes.

The Digital Twin is only used for monitoring the system without any feedback to the physical modules. This is not possible using CoAP as it is a client-server based protocol. By implementing the LWM2M protocol in this system, each module can also be remotely controlled along with being remotely monitored.

A

List of Figures

2.1 RIOT Structure [3] 12

2.2 Abstract Layering of CoAP Protocol [4] 14

3.1 Conceptual Model 24

3.2 Conveyor and Turntable [2] 25

3.3 Slider 26

3.4 Nucleo-F767ZI Microcontroller Board [22] 27

3.5 Nucleo-F767ZI - ESP32 Connections 28

3.6 Nucleo-F767ZI - LiFi Connections 29

3.7 Nucleo-F767ZI - NRF24L01+ Transceiver Connections 29

3.8 Simulation of Factory Layout 1 32

3.9 Simulation of Factory Layout 2 33

3.10 Sequence Diagram depicting the working of the system 34

4.1 Procedure for Experiment 1 36

4.2 Plot of Update Time for 50 Iterations 37

4.3 Procedure for Experiment 2 39

4.4 Runtime of individual modules 40

B

List of Tables

4.1 Factory Module Update Speeds 37

4.2 Runtime of modules and total time 40

C

Listings

3.1 Linux Command for Wired Network Setup 30
3.2 Linux Commands for ethos setup for each module 30
3.3 Memory Allocation for Threads 32

D

Bibliography

[1] P. Grznár, M. Gregor, M. Krajčovič, Š. Mozol, M. Schickerle, V. Vavrík, L. Ďurica, M. Marschall, and T. Bielik, "Modeling and simulation of processes in a factory of the future," *Applied sciences*, vol. 10, no. 13, p. 4503, 2020.

[2] A. Karri, "Development and evaluation of safety-aware over-the-air (ota) updates in the factory planning laboratory," Nov 2021. http://comsys.ovgu.de/Thesis+Topics/Finished+Theses/Aravind+Karri+2021.html.

[3] E. Baccelli, C. Gündoğan, O. Hahm, P. Kietzmann, M. S. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählisch, "RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT," *IEEE Internet of Things Journal*, vol. 5, Mar. 2018. https://www.riot-os.org/assets/pdfs/riot-ieeeiotjournal-2018.pdf.

[4] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)." RFC 7252, June 2014. https://www.rfc-editor.org/info/rfc7252.

[5] D. K. Nilsson, L. Sun, and T. Nakajima, "A framework for self-verification of firmware updates over the air in vehicle ecus," in *2008 IEEE Globecom Workshops*, pp. 1–5, 2008.

[6] B. Moran, H. Tschofenig, H. Birkholz, K. Zandberg, and Øyvind Rønningstad, "A Concise Binary Object Representation (CBOR)-based

Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest," Internet-Draft draft-ietf-suit-manifest-23, Internet Engineering Task Force, Sept. 2023. https://datatracker.ietf.org/doc/draft-ietf-suit-manifest/23/.

[7] S. Lang, T. Reggelin, M. Jobran, and W. Hofmann, "Towards a modular, decentralized and digital industry 4.0 learning factory," in *2018 Sixth International Conference on Enterprise Systems (ES)*, pp. 123–128, 2018.

[8] Fischertechnik, "fischertechnik und simulation - eine optimale kombination!," 1965. https://www.fischertechnik.de/de-de/produkte/industrie-und-hochschulen.

[9] J. Bauwens, P. Ruckebusch, S. Giannoulis, I. Moerman, and E. De Poorter, "Over-the-air software updates in the internet of things: An overview of key principles," *IEEE Communications Magazine*, vol. 58, pp. 35–41, 02 2020.

[10] S. Halder, A. Ghosal, and M. Conti, "Secure over-the-air software updates in connected vehicles: A survey," *Computer Networks*, vol. 178, p. 107343, 06 2020.

[11] S. Mahmud, S. Shanker, and I. Hossain, "Secure software upload in an intelligent vehicle via wireless communication links," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, pp. 588–593, 2005.

[12] D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," in *ICC Workshops - 2008 IEEE International Conference on Communications Workshops*, pp. 380–384, 2008.

[13] M. Steger, A. Dorri, S. S. Kanhere, K. Römer, R. Jurdak, and M. Karner, "Secure wireless automotive software updates using blockchains: A proof of concept," in *Advanced Microsystems for Automotive Applications 2017*, pp. 137–149, Springer International Publishing, 2018.

[14] R. Petri, M. Springer, D. Zelle, I. McDonald, A. Fuchs, and C. Krauß, "Evaluation of lightweight tpms for automotive software updates over

the air," in *Proc. of 4th International Conference on Embedded Security in Car USA*, pp. 1–15, 2016.

[15] H. Guissouma, C. P. Hohl, H. Stoll, and E. Sax, "Variability-aware process extension for updating cyber physical systems over the air," in *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–8, 2020.

[16] F. Tao, Q. Qi, L. Wang, and A. Nee, "Digital twins and cyber-physical systems toward smart manufacturing and industry 4.0: Correlation and comparison," *Engineering*, vol. 5, no. 4, pp. 653–661, 2019.

[17] M. R. Enders and N. Hoßbach, "Dimensions of digital twin applications-a literature review," 2019.

[18] K. Vijayakumar, C. Dhanasekaran, R. Pugazhenthi, and S. Sivaganesan, "Digital twin for factory system simulation," *International Journal of Recent Technology and Engineering*, vol. 8, no. 1, pp. 63–68, 2019.

[19] R. M. Wahab, A. I. Taharudin, A. R. Hemdi, N. M. M. Noor, A. F. Zubair, and N. N. Azmi, "Interactive visualization to enhance learning experienced on machining process of edm wire cut through augmented reality (ar)," in *2023 IEEE 12th International Conference on Engineering Education (ICEED)*, pp. 1–6, 2023.

[20] C. Sankaramanchi, "I-4.0-learning-Laboratory-," June 2023. `https://github.com/skrishnachaitanya09/I-4.0-learning-Laboratory-`.

[21] J.-P. Akelbein, K. Beckmann, M. Hoss, S. Schneider, S. Seyfarth, and M. Thoss, "Base move - a basis for a future-proof iot sensor," in *INFORMATIK 2020* (R. H. Reussner, A. Koziolek, and R. Heinrich, eds.), pp. 931–938, Gesellschaft für Informatik, Bonn, 2021.

[22] STMicroelectronics, "Nucleo-f767zi," Feb 2017. `https://os.mbed.com/platforms/ST-Nucleo-F767ZI/`.

[23] V. D. Mukku, S. Lang, and T. Reggelin, "Integration of lifi technology in an industry 4.0 learning factory," *Procedia Manufacturing*, vol. 31, pp. 232–238, 2019.

[24] Fritzing, "Learning electronics with fritzing," 2016. https://fritzing.org/learning/.

[25] D. Benson, "Draw.io - free flowchart maker and diagrams online," 2015. https://app.diagrams.net/.

[26] B. Dilts and K. Sun, "Lucidchart," 2010. https://lucid.app/documents#/documents?folder_id=recent.

[27] J. Little and C. Moler, "Matlab," 1984. https://de.mathworks.com/products/matlab.html.

[28] B. Valentin and S. G. Tangirala, "Border router ota using wifi," May 2023. https://forum.riot-os.org/t/border-router-ota-using-wifi/3895/2.

[29] B. Valentin, "Cpu/esp_common: Esp-wifi: Drop assert(val) #19854 · riot-os/riot," Aug 2023. https://github.com/RIOT-OS/RIOT/pull/19854.

[30] B. Valentin, "Gnrc_static: Fix static pid assignment #19855 · riot-os/riot," Aug 2023. https://github.com/RIOT-OS/RIOT/pull/19855.

Declaration of Academic Integrity

I hereby declare that I have written the present work myself and did not use any sources or tools other than the ones indicated.

Datum: .....

(Signature)