Masters thesis

A game engine based digital twin framework enabling next-gen manufacturing

Davis, W.

A GAME ENGINE BASED DIGITAL TWIN FRAMEWORK ENABLING NEXT-GEN MANUFACTURING

A thesis submitted to Middlesex University in partial fulfilment of the requirements for the degree of Master of Science by Research

William Davis
M00581005

School of Science and Technology
Middlesex University

July, 2023

Acknowledgment

Many thanks to my supervisors Dr Huan Nguyen, Dr Ramona Trestian and Prof Mehmet Karamanoglu for inspiration, guidance and support throughout my research. I would also like to thank the London Digital Twin Research centre and the UK-India Education and Research Initiative for their time and support.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

Notation	Description
AGV	Automated Guided Vehicle
AI	Artificial Intelligence
BIM	Building Information Modelling
CAD	Computer Aided Design
CPS	Cyber-Physical System
DS	Digital Shadow
DT	Digital Twin
FMP	Festo Mobile Phone
GVL	Global Variable List
HMI	Human Machine Interface
IaaS	Infrastructure-as-a-service
IoT	Internet of Things
IP	Internet Protocol
LiDaR	Light Detection and Ranging
MES	Manufacturing Execution System
MPS	Manufacturing Production System
NAS	Network Attached Storage
NIC	Network Interface Controller
OPC	Open Platform Communications
OS	Operating System
PaaS	Platform-as-a-service
PCB	Printed Circuit Board
PLC	Programmable Logic Controller
RFID	Radio Frequency Identification

Notation	Description
ROS	Robot Operating System
RPM	Rotations per minute
SaaS	Software-as-a-service
STP	Sensor Trigger Program
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Abstract

Research into digital twins technology for automation and infrastructure is a new and prosperous field that is at the cutting-edge of current technology. Big data analytics, cloud computing, augmented reality and Internet of Things (IoT) have been around for a couple of years, and digital twin technology is what ties it all together. The aim for this research is to create a digital twin framework without ties to professional proprietary software. This should create a modular base for anyone to build a digital twin onto quickly, designed and developed on the Cyber-Physical System (CPS). Can digital twin technology be applied to monitoring and automation in a smart factory? Is it possible to apply a live tracking algorithm to a multi-stage smart factory? Is it possible to do this in a framework format to be applicable to other facilities? Use of the Festo CPS at Middlesex University will be a case study. The journey to realising these goals starts at forming a digital representation of a physical system, incorporating live tracking between the systems, offline simulation and overall being in a modular framework structure. This allows the system to be applied to multiple case studies. The outcome of this project was to not necessarily create a faster system, but to create a system less likely to have unscheduled downtime. The live tracking algorithm developed was successful in tracking multiple carriers, however, less successful when running into certain edge cases. Simulation mode works well and outputs data of the same shape as the physical CPS.

Chapter 1
Introduction

1.1 Cyber-Physical System

The CPS at Middlesex University is an Industry 4.0 standard mini-factory production line, featuring a customisable ordering suite. The factory consists of two independent islands and a Robotino Automated Guided Vehicle (AGV) that travels between them, receiving and delivering carriers. Carriers are small transport vessels that run on a guided conveyor system running between the production stations on each of the islands. The product(s) produced by this system is called a Festo Mobile Phone (FMP). These are made up of three major components: lower and upper casings, and a Printed Circuit Board (PCB). The PCB has two fuse holders mounted towards one side. This FMP has four significant variants and several different production properties that can be changed according to customer requirements. Variants are defined as: zero fuses, one fuse positioned to the left or right, or two fuses. Other properties that can be changed are heating temperature and time and pressing force and time. The CPS is split into two islands, each identical in size: 1.8m x 1.8m x 1.7m. The distance between the two islands is 2m. Each conveyor belt length is 0.7m. Each island is separated into four stations: three process driven stations, contributing to the FMP production, one handling the courier system between islands. The process driven stations are each controlled by a Siemens ET200sp Programmable Logic Controller (PLC), the station controlled by a Festo Codesys PLC. The production line's main ordering software is provided from Festo called Festo Manufacturing Execution System (MES), this handles production control. The Robotino Operating System (OS) is Linux Ubuntu running an early version of Robot Operating System (ROS), with physical dimensions: 0.4m diameter x 1.2m.

Figure 1.1: Physical CPS

1.1.1 CPS Stations

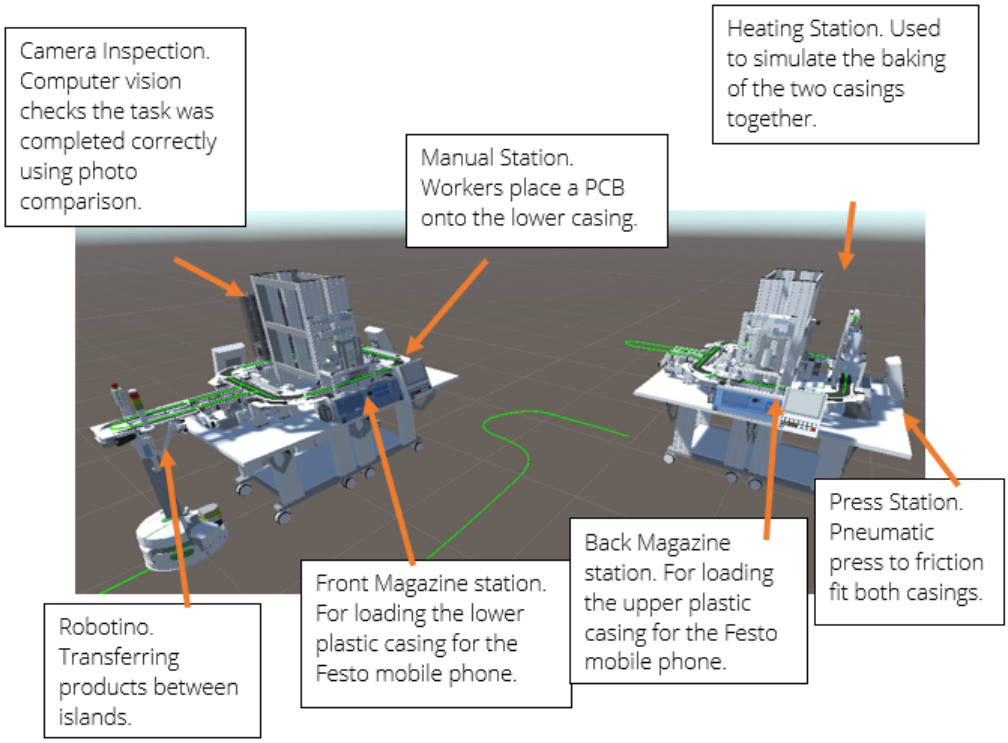


Figure 1.2: Overview of Virtual Digital Twin

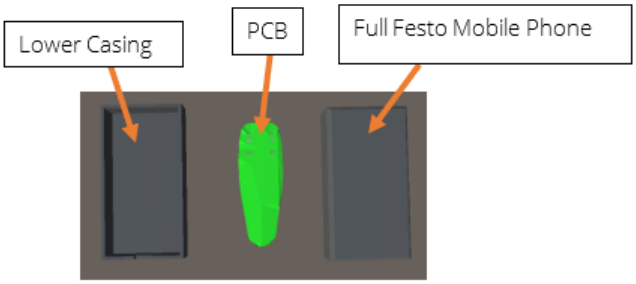


Figure 1.3: Components of a Festo Mobile Phone

First Island Stations:

- Magazine Front Station - this lowers the lower casing onto the next available carrier.
- Manual Station - a worker places the required PCB configuration onto the lower casing.

- Camera Inspection Station - this quality checks the worker's PCB placement and specification.
- Transfer Handling Station - this handles requests to and from the AGV.

Second Island Stations:

- Magazine Back Station - lowers the upper casing onto the next carrier that requires a casing.
- Pressing Station - friction fits the upper casing onto the lower casing using a pneumatic press.
- Heating Station - heats the product up to simulate baking. Heated to the temperature and duration requested.
- Transfer Handling Station - handles requests to and from the AGV.

1.2 Digital Twin Definition

Digital Twin (DT) technology is very new, therefore has the issue of a non-standardised definition. [5] gives a description that is gaining traction among recent papers and has similar requirements to this project: "Digital twin is an integrated multi-physics, multi-scale, probabilistic simulation of a complex product and uses the best available physical models, sensor updates, etc., to mirror the life of its corresponding twin".

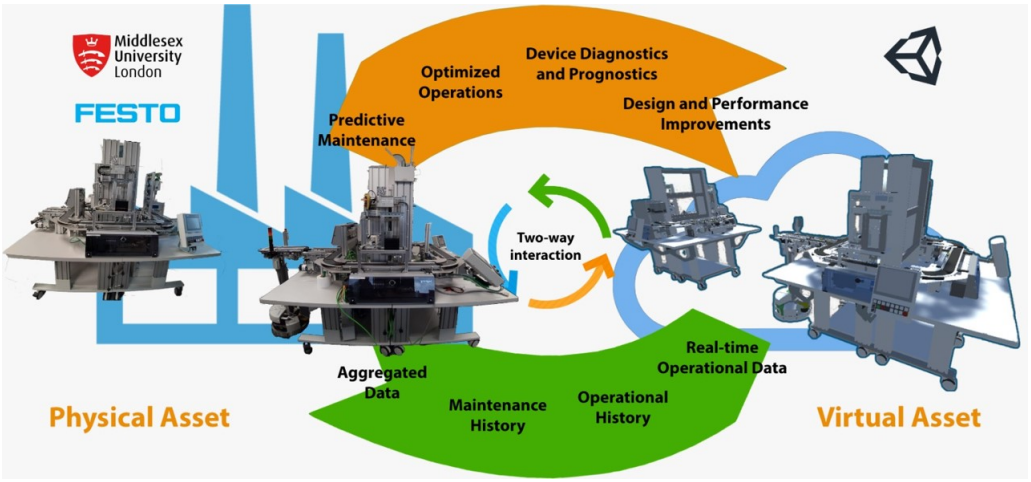


Figure 1.4: physical asset vs virtual asset

In this research, the definition should be taken as a digital representation of the physical system with these properties:

- Live mirroring
  - Physical system being a master.
  - Physical system being a slave.
- Simulation
  - Running new system programs and tests on a simulation before trying on the physical system to check it works.
  - Running real-time simulations while the physical system is running by collecting data, analysing, and creating new predictions for when maintenance is required.
- Predictions
  - A machine learning algorithm will be implemented with a bank of historical, classified data on which to base its assumptions. As $t \rightarrow \infty$, that data bank will grow.

Additionally, the definition of a Digital Shadow (DS) should be taken as having the simulation and prediction characterises of a DT, but lacking the two-way communication aspects - communication from the simulated system back to the physical.

1.3 Research question/problem

This research is trying to address the following questions: Can digital twin technology be applied to monitoring and automation in a smart factory? Is it possible to apply a live tracking algorithm to a multi-stage smart factory? Is it possible to do this in a framework format to be applicable to other facilities?

1.4 Research Aim

The aim of this project is to investigate and develop a digital twin framework that replicates and monitors a production line in real-time with the case study focusing on the Festo CPS without limitations of proprietary software.

1.5 Motivation

Unplanned downtime has affected 82% of companies and can cost a company up to £209,000 per hour. Of this 82%, the average length of downtime for each incident is 4 hours, costing an average of £836,000 [6]. As the maintenance is unplanned, the root cause of the problem is likely to be unknown; meaning time will have to be spent locating the issue in addition to time to repair. Knowing how a system's components wear over time and how they act in different conditions is the key to reducing unplanned downtime. However, while most automation components do ship with information on wear, the key issue is knowing when a component has got to the end of its life. With Big Data analytics, it is possible to understand the conditions surrounding the system to understand better when a system will fail, or a component within the system. Conditions that differ from ones that are planned can either make a component life last longer or shorter. This can cause a company to change a component too early, or overestimate how long one will last, resulting in unplanned or avoidable downtime [7].

1.6 Objectives

To design, implement and deploy a digital twin framework for the CPS with the following objectives:

- Develop a data interface to capture real-time information from the sensors in the smart factory to live stream / archive.
- Investigate a game engine approach and create a framework, for the 3D modelling of a digital twin system.
- A real-time monitoring framework for the operation of smart factory.

1.7 Outcomes

The research work presented in this report will provide the following contributions to the advancement of current practice:

- A digital twin framework that is easily applicable to a new system, making this technology more accessible to companies, and less dependent on proprietary software.

- A two-way software communication and control protocol with the physical system.
- Remote access to historical data in the cloud for accurate simulation and processing from anywhere, including data sharing with partner systems over the cloud for increasing the historical database

1.8 Publications

- M. Raza, P. M. Kumar, D. V. Hung, W. Davis, H. Nguyen and R. Trestian, "A Digital Twin Framework for Industry 4.0 Enabling Next-Gen Manufacturing" – 2020. 9th International Conference on Industrial Technology and Management (ICITM), Oxford, United Kingdom
- S. V. Mihai, D. V. Hung, H. Nguyen, R. Trestian, W. Davis, "A Digital Twin Framework for Predictive Maintenance in Industry 4.0 Contact Author" – 2020. International Conference on High Performance Computing & Simulation (HPCS), Barcelona, Spain (Virtual)
- Davis, William; Yaqoob, Mahnoor; Bennett, Luke; Mihai, Stefan; Hung, Dang Viet; Trestian, Ramona; Karamanoglu, Mehmet; Barn, Balbir; Nguyen, Huan X, "An Innovative Blockchain-Based Traceability Framework for Industry 4.0 Cyber-Physical Factory" – 2022. 11th International Conference on Industrial Technology and Management (ICITM), Oxford, United Kingdom
- Mihai, M. Yaqoob, D.V. Hung, W. Davis, P. Towakel, M. Raza, M. Karamanoglu, B. Barn, D. Shetve, R. Prasad, H. Venkataraman, R. Trestian, and H. X. Nguyen, "Digital twins: A survey on enabling technologies, challenges, trends and future prospects," IEEE Communications Surveys and Tutorials, vol. 24, No. 4, pp. 2255-2291, 4th Quarter, 2022.
- H. Nguyen, W. Davis, and C. Sinclair, "Formula 1 is Leading the Digital Twin Technology", dt.mdx.ac.uk, 2021

1.9 Data Collection

Once the simulation side of the digital twin system is ready for analysis, data collection will start. The system will be run for a minimum of 100 orders with no errors. The

system will then be rerun as above, each time inducing a different failure or trend to classify common unplanned downtime scenarios. As this is a fully autonomous system, there will be no ethical issues surrounding the collection process.

Chapter 2
Literature Review

2.1 Introduction

The current Industry 4.0 standard cuts the cost of production, builds efficiency and gives companies an increasingly versatile approach to production. Factory production stations now communicate directly with one another, eliminating the need to communicate via a central processing controller. This decentralisation through modularisation and the Internet of Things increases flexibility, opportunity and efficiency. Rather than a centralised control unit delivering instructions to each machine to carry out linear sequential steps, individual machines now communicate directly enabling the partly finished product to be passed straight on to the next station. As everything is now processed locally, the production line is equipped to produce any number of unique products, not previously possible on single unit lines. By not having to communicate with a centralised unit, the production line can run more smoothly and efficiently. In addition to increased efficiency, the new security sensors built into the autonomous modular systems create a safe working environment for human operatives, ensuring robots halt if they encounter an obstruction [1]. This has the added benefit of workers being able to touch a robot to stop its motion without the need to activate an isolator. This was demonstrated by the car manufacturer Mercedes-Benz when unveiling their version of Industry 4.0 [8]. Digital Twin technology is a rapidly developing area of research and has a large selection of possible applications. Digital Twins have increased in popularity over recent years due to efficiency gains and a reduction of overall cost. This literature review will focus on digital twins with their application in the manufacturing industry as well as drawing inspiration from industries with complementary merits.

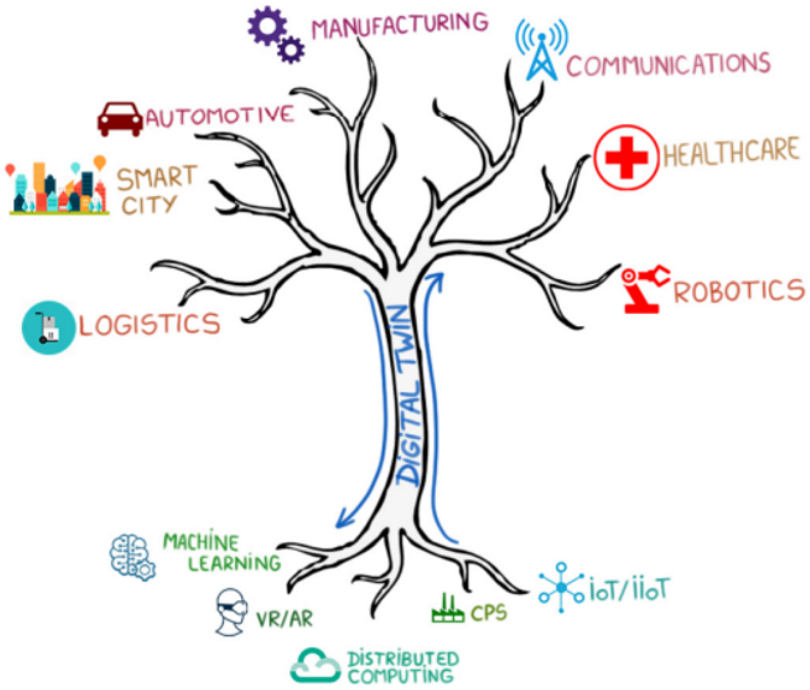


Figure 2.1: The many branches of digital twin technology [1]

2.2 Current wear prediction systems

Current non-digital twin systems just use standard real-time monitoring, so when something goes wrong, they can see what happened. In the case of bridge failure, using standard real-time monitoring will not effectively predict a failure, hence the need for a digital twin [9]. The only information companies currently use is a crude life expectancy for key parts, which suggests when technicians should swap out parts, whether they are good or bad. To get those figures, manufacturers use FEA and form predictions on new system data to predict the wear. By contrast, digital twins run that prediction continuously, producing a learning environment in which all parts are aging. In extreme conditions, this could mean the usual system-predicted component lifetimes may well be shortened. Using a digital twin will adjust for these variables, producing more realistic and accurate maintenance schedules.

2.3 Industries using digital twins

2.3.1 Formula 1

Every time a team tests and races its cars, it uses a digital twin to monitor and predict when problems are going to arise. Clearly this is essential due to the very high cost of each car. A typical race car has over 150 sensors for measuring temperatures, pressures, acceleration, forces and shaft speeds [10]. These data are collected every 0.001 seconds (over a billion numbers in a race), which means it is highly unlikely that they would miss any vital data collection when things go wrong. These data are also essential as teams may manufacture over 3000 new components each week. Clearly, there is potential for signification cost savings by ensuring components are not designed to the wrong specification due to incorrect data. Digital twins have made motor racing a more cost-efficient sport, as well as accelerating the development of new technology.
Consider the key aspects of Formula 1 racing (F1).

Car Handling	Track types Car design Starting position on grid Driving styles Driver emotional state Driver error Race interruptions Weather
Dynamic Environment	Live race (all above variables from other drivers)
Cost	Lives Money Constrained preparation time Reputation Sponsorship

Table 2.1: Volatile variables when considering modelling f1 cars

Edward Lorenz presented Chaos Theory as 'The study of apparently random or unpredictable behaviour in systems governed by deterministic laws' [11]. Initially at least, chaotic systems appear predictable, but quickly begin to appear random. Consequently, certainty in forecasting reduces exponentially with time. In the competitive area of F1, the art of race strategy is to stay at least one step ahead of the initial predictability and

increase the certainty of the forecasting. This is something of an art, having three phases that can be described as:

- Simulation
- Performance
- Feedback

2.3.1.1 Simulation

Preparation for performance. In F1, this is the design and production of a digital twin of the car, tailored to a driver's needs, style and upcoming racetrack. As Patrick Lane-Nott, former member of the McLaren Applied Technologies team states in his article 'How a Formula 1 mindset can revolutionise digital twins in construction' [12]: 'Countless alternative car set-ups are evaluated (fine-tuning variables which affect the trade-off between straight-line speed and cornering performance) to ensure that when the car first goes on track, a strong start is made to the weekend's continuous process of improvement'. Drivers then complete thousands of laps of the circuit in the digital twin to fine-tune the team's decision making.

2.3.1.2 Performance

In F1, the car feeds live performance data to its digital twin from hundreds or thousands of sensors. These allow the team (a collaboration between the engineers and their AI to ask questions such as: How's it going? Are there any issues? Is anything about to break? Statistically sound decisions can then be made almost instantly.

2.3.1.3 Feedback

Performance review for sustainability and predictive maintenance. In the heat of the race, there isn't time, of course, to collate and use all the massed data for decision making. While AI and machine learning will help prioritise what the team acts upon during the performance, the data set is also used for post-race feedback and review: the comparison of performance against expectation. This is termed 'validation' in the context of programming. While the terms 'feedback' and 'review' a neutral, a key aim of these data is to produce sustainability by identifying the root-cause issues behind any indicators of under performance.

2.3.2 Construction

While this literature review is not intended to be limited to the history of the McLaren Advanced Engineering team, now known as McLaren Applied, its work is an exemplar for the spread of the Digital Twin strategy. Initially a part of McLaren Automotive, the company realised that its approach to marginal gains was a general rubric for performance improvement and could be used by any sector. Examples include solar panels for the Beagle 2 Mars lander, the London Olympics rowing, sailing and cycling teams, and the development of a 5G active antenna for high-speed transport. A specific example of this application of a rubric can be seen in the above quoted article by Lane-Nott (now Director of Engineering at hyperTunnel): How a Formula 1 mindset can revolutionise digital twins in construction, in which he transfers the skills and understanding of F1 Digital Twins to tunnelling. The broad challenges Lane-Nott poses are that 'contractors are still largely building to drawings manually, with no other input' and that 'in safety-critical environments, there is simply no room for personal judgement'. A provocative starting point for the construction industry. Let's explore its setting.

Multiple variables	Geology of the site Choice of construction method Sequencing of construction Overall project management Contractor management Surveyor accuracy Construction worker skills Construction worker accuracy Machine accuracy Machinery reliability Supply chain interruptions Weather
Dynamic Environment	Active construction (All above variables)
Cost	Lives Money Multiple consequences of late completion Reputation and future contracts

Table 2.2: Volatile variables when considering modelling in construction

2.3.2.1 Simulation

Lane-Nott discusses how using a digital twin can answer central questions such as the choice of construction method, the sequencing of tasks, and strategies to mitigate interruptions to the supply chain. However, he goes on to say that 'The digital twin can even be used to model more extreme scenarios like what happens in the tunnel during temperature or wind extremes or when the tunnel has to withstand that once-in-a-century storm, landslide or earthquake'.

2.3.2.2 Performance

'Time is money' might be a cliché, but is nonetheless as true in construction as elsewhere. Live data, therefore, is essential for effective project management. A crucial difference between Building Information Modelling (BIM) and digital twins is the nature of the data. In BIM, the data is static, but just as the F1 car feeds masses of data back to its digital twin for immediate decision making, so too should the construction site, through its own array of sensors: live data for real-time decisions. The revolution in tunnelling is in the combination of existing technologies, such as horizontal directional drilling (HDD), which is the robotic 3D printing of the tunnel in a context of live data. New high-definition surveying and modelling techniques provide live data back to the twin, allowing it to manipulate both aspects of the automation – the drilling and the 3D printing. Lane-Nott says the impact of this approach is that 'it becomes possible to avoid unpleasant surprises and to bring an end to the wastefully expensive habit of over-specifying tunnelling machinery 'just in case.' [13].

2.3.2.3 Summary

The examples above show how the McLaren Advanced team and Lane-Nott have taken the underlying digital twin strategy and applied it to different settings. This ambition is also central to this Game Engine-based Digital Twin Framework. Rather than create a digital twin and associated programming on a platform bespoke to the Siemens and Festo equipment, the aim has been to use a ubiquitous game engine (Unity) so that the work can be easily transferred to novel settings. As this report will go on to show, this has not been without its challenges.

2.3.3 Healthcare

By using the F1 model as an example of our baseline understanding of digital twins, we can begin to see major challenges in other sectors, such as healthcare, for instance. In this setting, the real-time feedback to a digital twin would ideally be about the effectiveness of a chosen therapy. Lane-Nott makes an important note about an F1 car's digital twin, that it: 'Entails bringing together numerous digital twins, each representing a discrete section of the vehicle, such as the engine, chassis and bodywork, and each usually developed by a different department or supplier'. So, now let's use this lens to explore healthcare. Clearly, the term 'healthcare' is an umbrella term for digital-twin possibilities as disparate as organisation management, virtual organs and genomic medicine. While noting that this is a rapidly changing landscape, there is one factor that seems consistent across the majority, if not all, of the current wide range of applications. If a true digital twin enables us to look at: Simulation, Performance, and Review, perhaps a useful distinction for a twin that only looks at Simulation might be to term this a Digital Shadow. For the Performance elements discussed above, there must be a live feed direct from the real world to the digital twin, which in turn enables a technology-enabled decision to be made about the next step. Healthcare's use of the term digital twin is, therefore, unhelpful, though clearly expressing an aspiration. Let's take one example: Genomic medicine. According to a study by Danuta R. Gawel et al, the efficacy of medication to treat common diseases can range from around 60% down to just 25%. A key variable is the patient's personal genome. To express the multiple variables in this example, I'll defer to Gawel et al's study [14]: The aim of this work is to create a so-called digital twin of the patient – a test environment based on their personal genome – and to test the thousands of cells using 'single-cell RNA sequencing' in order to determine the most effective medication to prescribe. This is clearly an ingenious version of the Simulation phase. Following the administering of the drug (Performance), there will obviously be a Review of effectiveness.

2.3.4 Bridges - London

A different setting again, is the development of 'smart bridges'. The SmartBridge project is in London, designed by the James Fisher construction company in collaboration with Brunel University, London. This bridge has sensors mounted inside to continually monitor its health, providing vital information, such as the amount of load, traffic volumes and weather. This project aims to provide vital maintenance data as well as a complete

data picture that can be analysed following any local or wholesale failures of the structure. This analysis would then contribute vital case studies to future bridge engineers, making production time quicker and bridges more reliable [9].

2.3.5 Industry 4 - Smart Factories

Finally, relating back to industry 4 and the era of smart factories. In [15], the paper talks about the use of big data analytics, machine learning and cloud computing to create a virtual representation of a system. The end goal of a virtual representation of a system is to encapsulate all types of data a non-DT system may look at without taking others into account. To create a new, higher level of abstraction from the former, decreasing the likely-hood of data misinterpretation, therefore, decreasing the risk of unplanned downtime. In [15], their definition of DT fits nicely into the definition of a Digital Shadow. One way communication from the factory, into software. The only change from a non-DT system would then be a digital representation, since other systems still perform these predictive maintenance tasks, albeit not quite as successfully as a digital shadow system [16].

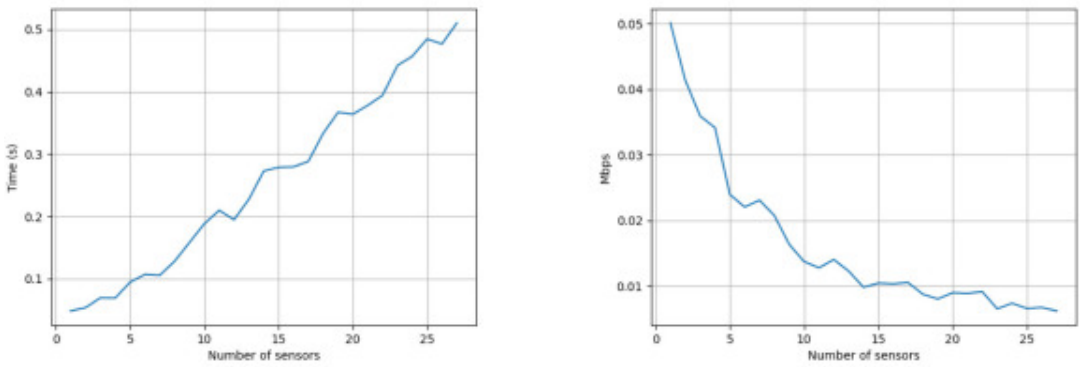
2.3.5.1 Low-cost CPS upgrades

In [17], the authors show how successful low-cost microcontrollers can be at interfacing with a digital twin network over the same industrial protocols as PLCs. The research was applied to a Manufacturing Production System (MPS) that forms V-bends in metal by pressing two dies from opposite sides. [17] split their twin definition into three parts: the phyiscal system, a digital thread, and a ”cyber MPS demonstrator”. Where digital thread was OPC-UA communication and the demonstrator the data fushion captured and amalgormated into a visual dashboard. As technology is moving fast, it is difficult for smaller companies to ultilise the benefits of Industry 4 and soon DTs. Implementing microcontrollers to proved the system could be viewed as a more cost-effective choice.

2.3.5.2 DT framework

Leading on from [17] low-cost microcontrollers, creating an open source DT framework, not tied to specific manufacturers with proprietary software would further advance this notion that technology should be accessible to all [2]. The more data gathering systems, or sensors, a CPS has, the greater the computational power needed for the DT to fuse all this data, which can increase the latency in generating a reliable and coherent virtual

representation. In certain CPS applications, like autonomous vehicles and industrial control systems, low-latency responses are imperative, necessitating a careful balance between the number of sensors and latency requirements. See Figure 2.2.



(a) Latency (b) Throughput

Figure 2.2: Latency vs Throughput [2]

The authors of [3] took it a step further, closer to the aim of this paper - high-speed live tracking. Using TCP for a higher speed protocol providing real-time data. They explored this using a simple user defined temperature controlled chamber. OpenPLC was used as the physical system controller, communicating with the DT node - a Raspberry Pi. They created a mathematical prediction of how the temperature curve would look if the temperature was progressively increased. Various open-source softwares were used for the prediction and computational models. A system architecture is shown in Figure 2.3.

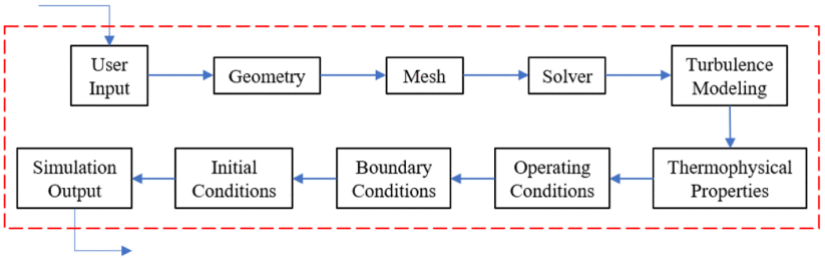


Figure 2.3: System architecture with open-source software [3]

2.4 Current Internet of Things Analytics platforms

Considering technologies that interface with non-Big-Data devices.

2.4.1 Thingspeak

Thingspeak is an Internet of Things (IoT) analytics platform that manages data uploaded to it from an IoT system. Designed by MathWorks, Thingspeak is inter-operable with Simulink and MATLAB giving it distinct advantages over other stand-alone software. MATLAB code can be executed directly inside Thingspeak, increasing the possibilities for data processing and visualisations over other IoT analytics platforms with pre-set graphing functions. The cloud-based software allows for data visualisation, aggregation and live data streams. Thingspeak has additional features such as interfacing with popular APIs such as Twitter and Twilio [18].

2.4.2 Mindsphere

Mindsphere from Siemens is an IoT analytics tool focusing specifically on industrial IoT applications rather than general IoT applications. The Mendix-based application platform uses advanced analytics tools with AI to track complex trends and data. Mindsphere works well with Siemens PLCs which now have prebuild libraries and functions to send data at a high frequency. Siemens claim 300 data points per second by uploading from a single PLC using a MindConnect Nano [19].

2.4.3 The Cloud

The Cloud is becoming the backbone of society and fundamental to new industry building around IoT. For new innovations to function properly, wireless networking and data storage/control from the cloud is essential. The three types of cloud computing are:

- Software-as-a-Service (SaaS) - A business subscribes to an application over the internet.
- Platform-as-a-Service (PaaS) - A business creates its own bespoke application for use only in the business. These are generally a more secure option.
- Infrastructure-as-a-Service (IaaS) - A business of significant size rents out space on its powerful servers to other businesses [20].

Large manufacturing companies will often use all three service-types, each with its advantages and disadvantages. Some cloud servers are also set up as highspeed Network Attached Storage (NAS) file servers. In these instances, all factory processes are logged,

making it relatively simply and efficient to deploy new designs or distribute files to machines instantaneously [21].

2.5 Cyber Security

In the third industrial revolution, PLCs were basic and the cyber security for these units was in its infancy. Systems were vulnerable to attack by hackers. While cyber security has improved significantly, it is also more fundamental to business due to Industry 4.0 centring on the Internet of Things principle. Now, any individual device with a Wi-Fi or Bluetooth connection is vulnerable to attack. These cyber-attacks fall broadly into two camps:

- Accidental attacks, where viruses find their way onto a company's systems, but were not necessarily targeting specific PLCs
- Intentional attacks, the attacks designed to target and take out a specific controller [22].

A famous example of a cyber-attack on PLCs was the use of the virus Stuxnet, in June 2010. Allegedly created by the USA, Stuxnet was designed to disable Iran's suspected nuclear installations and alleged nuclear weapons development. The worm virus could manoeuvre itself through a range of different systems and platforms to reach its eventual target – the Siemens S7-300 PLCs running Iran's centrifuges [23].

2.6 A review of communication protocols fit for digital twins technology

2.6.1 OPC UA Communication

Open Platform Communications United Architecture (OPC UA) communication has been widely adopted by most Industry 4.0 systems and is becoming a standard. OPC UA is popular in the automation industry because of its reliability, multiplatform support and security and is designed for data collection and control [24]. It works with a server-client connection. In an industrial setting, the server runs on the PLC. This communication enables the creation of variables within an object on the OPC UA server, and allows a client to request variable values, or to subscribe to variables; the client being

notified when a value is changed [25]. At Middlesex University, this became accessible via the CPS through an update to its systems. The main flaw with this communication in the S7-1500 PLCs is the frequency at which the controller would publish updated variable data. The need for a large amount of data being sent at a low latency cannot be satisfied by this protocol.

2.6.2 Snap7 Communication

This library enables the developer to pole the PLC I/O list and individually get one sensor state at a time. Experimenting with the Snap7 library to get multiple variables at once was found to be unreliable. The main flaw with this system was its slow update time. Incorporating threading into the system helped spread the demand across CPS stations; however, the data publishing delay for one station was particularly slow: 160ms. A major issue for this project was that Snap7 is specifically designed for Siemens PLCs and would not allow any other controller to work with the system [26].

2.6.3 Socket server: TCP vs UDP

Using socket connections is fundamental to how devices communicate with each other over a network. Two early protocols used in communication are:

- Transmission Control Protocol (TCP)
- User Datagram Protocol (UDP)

TCP/IP communication was invented in the early 1970s, where TCP collects and assembles data [27], the Internet Protocol (IP) makes sure the data was sent to the correct location. Once confirmed, an acknowledgement packet is sent back [28], ensuring all packets are received and acted upon. If a packet is not acknowledged, it is resent. UDP does not require this acknowledgment packet, meaning UDP can be sent at a higher frequency as it is a connectionless protocol [29]. Depending on the importance of the information sent, the lack-of-reliability trade-off for speed can be good or bad. In live tracking an industry 4.0 production line, many sensors are binary and are only on for a fraction of a second. If sensor information is only sent when there is new data, (for example, a state change) that information must be received by the live twin. In conclusion, TCP communication is a more viable protocol for a digital twins application because of the packet acknowledgments and the connection-oriented format vs connectionless format [29].

2.7 Potential Digital Twin Environments

2.7.1 Unity

This is a popular game engine that can solve complex mechanical problems and give games more realism as Unity's physics closely resembles real-world physics. By making use of this engine, complex simulations can be run for robotics applications, such as mobile robots with path-finding algorithms [30].

Pros	Accurate physics engine. Easy to interface other libraries. Extensive documentation. Free software, not specific to any manufacturer.
Cons	Declaring sensor positions and path routing is more time consuming. Cannot build a 3D model in the software. Additional software required.

Table 2.3: Unity pros and cons

2.7.2 Siemens NX

This is Siemens' version of what they call a digital twin. It allows the user to create the 3D representation directly in the software, run simulations, and optimise it ready for the physical system to be built correctly, first time. Siemens also offers Mindsphere which is cloud-based software to analyse the data coming from the digital twin to make predictive maintenance calculations [31].

Pros	- It has a catalogue of all Festo products with an easy drag/drop function to quickly build a digital twin of a current system. (The physical system, however, must be Festo.) - Can produce the 3D model directly in the software. - Accurate physics engine.
Cons	- Specific to Festo/Siemens equipment. - Increased speed optimisation in larger systems is needed, as a high level of processing power is required. - No real-time tracking capabilities.

Table 2.4: Siemens NX pros and cons

The framework selected for this project is the Unity game engine. It is free software that has an accurate physics engine, documentation and a flexible API based in C# to make

interfacing with anything fast and reliable. Festo provided this to-scale 3D model and was imported into Unity.

2.8 Digital Twin Benefits

- The digital twin of a product can be created early in its design stage. This allows designers to simulate and validate a product's desired properties. [19].
- Digital Twins can be used in the physical plain as well as in software. For example, the PLC code can be written and virtually commissioned. This can be run faster than in the physical system so data sets can be much quicker to harvest [19].
- Companies who adopt this technology can see around 30% improvement in cycle times [32].
- Telecommunications and New Media (BITKOM) estimate that companies who invest in the technology could have an economic potential of over £69 billion by 2025 [32].

Chapter 3
Main system design and function

3.1 System Outline

As a result of this research, the software created for this project utilises the Unity game engine to create a digital twin framework, which has been applied to the CPS based at Middlesex University. In the Unity software, a scale model of the CPS is linked with the physical system. The system has two main settings: Physical Mode and Simulation Mode. Physical Mode is a live link to the physical system. This uses a high frequency two-way communication protocol to accurately visualise the current state of the physical system. Any changes to the physical system are reflected in the simulation. Simulation Mode runs independently of the physical system. The software is written in such a way that the same code is used for both physical and simulated functions. The data collection and output, therefore, is identical. Irrespective of either mode being selected, the option to upload select data to the cloud is available and tests were carried out with both Thingspeak and Mindsphere. The exportation of these data is carried out via python receiving a dump of data and bulk uploading. A diagram of this is shown in Figure 3.1.

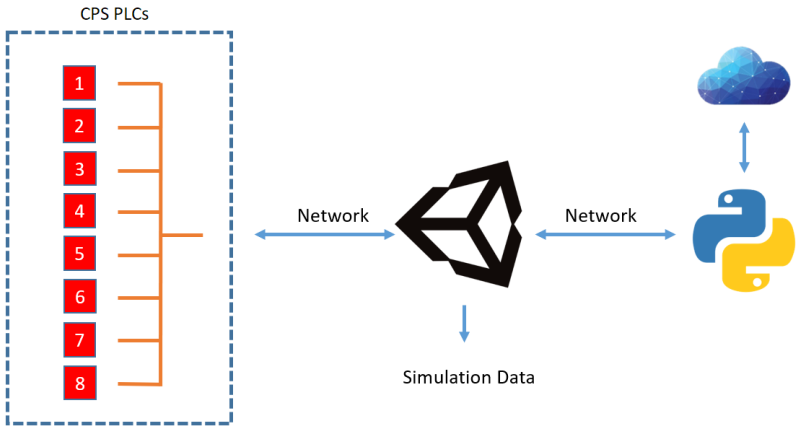


Figure 3.1: Showing the outline of the DT framework.

3.2 Festo CPS CAD

The Computer Aided Design (CAD) has been provided by the Festo sales team, therefore needed processing and slicing to isolate the individual components, allowing for software actuation of the models. The reprocessing and slicing was done in Blender, then exported into a .FBX file to be used in Unity.

3.3 Two-way communication protocol function

3.3.1 Data transfer protocol

To make this framework adaptable to any system, TCP sockets have been selected as the primary data transfer method. Devices that have a network interface controller (NIC) can communicate via TCP, therefore, any automated system will theoretically be able to communicate with Unity. When the software is launched, Unity starts an array of socket servers. The PLCs in the CPS are listening for the servers and automatically connect when they are available.

3.3.2 Data being sent

Tracking all the sensor data needs a high amount of processing, so data transfer efficiency is of a top priority. Processing all sensor values at every update cycle requires more handling, so being smart with what data to process is computationally more efficient. Since most of the data is Boolean, Unity can assume the condition of a sensor until it is updated. As an example, imagine the first packet of data sent is '1,1,1,1'. If there are no physical changes to the system, and the sensors being tracked still read '1,1,1,1', there is no need to send a new data packet. Communication, then, is 'event driven'. An example of this simple filtering can be seen in Figure A.1. Clearly, if there is a change in the physical system and the new sensor reading is, for example, '1,0,1,0', a new data packet will then be communicated to the software over TCP. Making assumptions such as this can open holes in the system where fast switching sensors can have their update missed if any packet loss were to occur. Using TCP allows the sender to know if a packet arrived or not, and if not, the packet will be sent again. As an added tracking layer of protection, the packets that are sent also contain an index number, so the number of lost packets can be tracked. Clearly, an alternative and quite common method of sending data over a network is to use UDP, broadcasting all data all the time at a high frequency. Lost packets tend to get overwritten quite quickly. For this specific application and processing power constraints, TCP is more reliable.

3.4 Live tracking

The system uses a multi-script structure. Whether in simulation mode or live mode, the code running is identical. An example of this is live tracking the physical carrier on

the conveyor and mapping it onto the virtual system. In the physical system, a carrier will activate capacitive sensors connected to the Siemens PLCs to trigger actions, such as start a key process in the manufacture of the Festo Mobile Phone. Sensor positions shown in Figure 3.2 are the same in the virtual system as well as the phyiscal system.

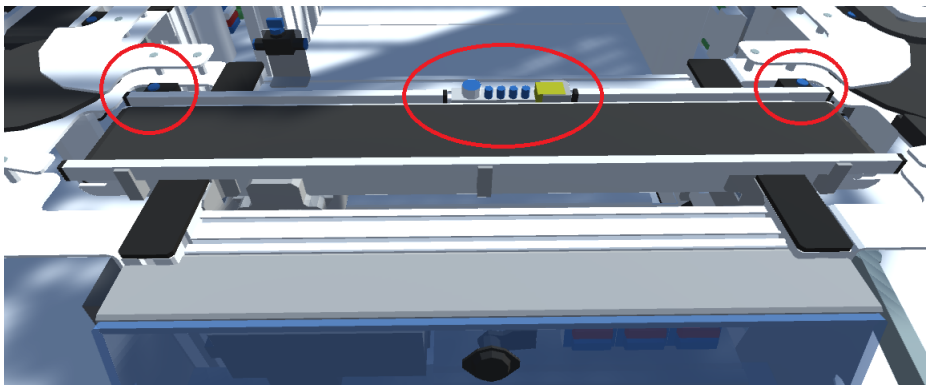


Figure 3.2: A single conveyor and its sensors

The Sensor Trigger Program (STP) is a modular program that acts as an attribute to all virtual sensors inside Unity; it is only active when Simulation Mode is set to True. The program works by detecting a collision event between two box colliders, checking the name of the object the script is attached to and setting the relevant variable to True in the Global Variable List (GVL). See Algorithm 1.

Algorithm 1 Virtual Sensor triggering algorithm

```
1:  if collisionEvent&simulationMode then
2:      caseValue = sensorName
3:  end if
4:  switch caseValue do
5:      case sensorName
6:          set GVL.sensorname = 1
7:      case sensorName1
8:          set GVL.sensorname1 = 1
9:      case sensorNameX
10:         set GVL.sensornameX = 1
```

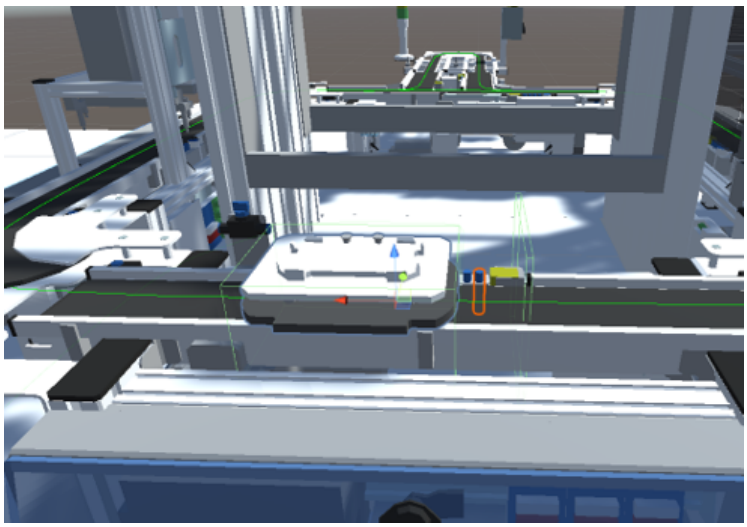


Figure 3.3: Carrier Box Collider vs Stopper Box Collider

The virtual part carrier has a virtual box surrounding the entire 3D model, shown in figure 3.3. This box, or box collider, shows if another body has collided with it or not. The same is true of the virtual sensors built into the CPS model. When a carrier is travelling round the conveyor belts, its box collider will collide with a sensor's.

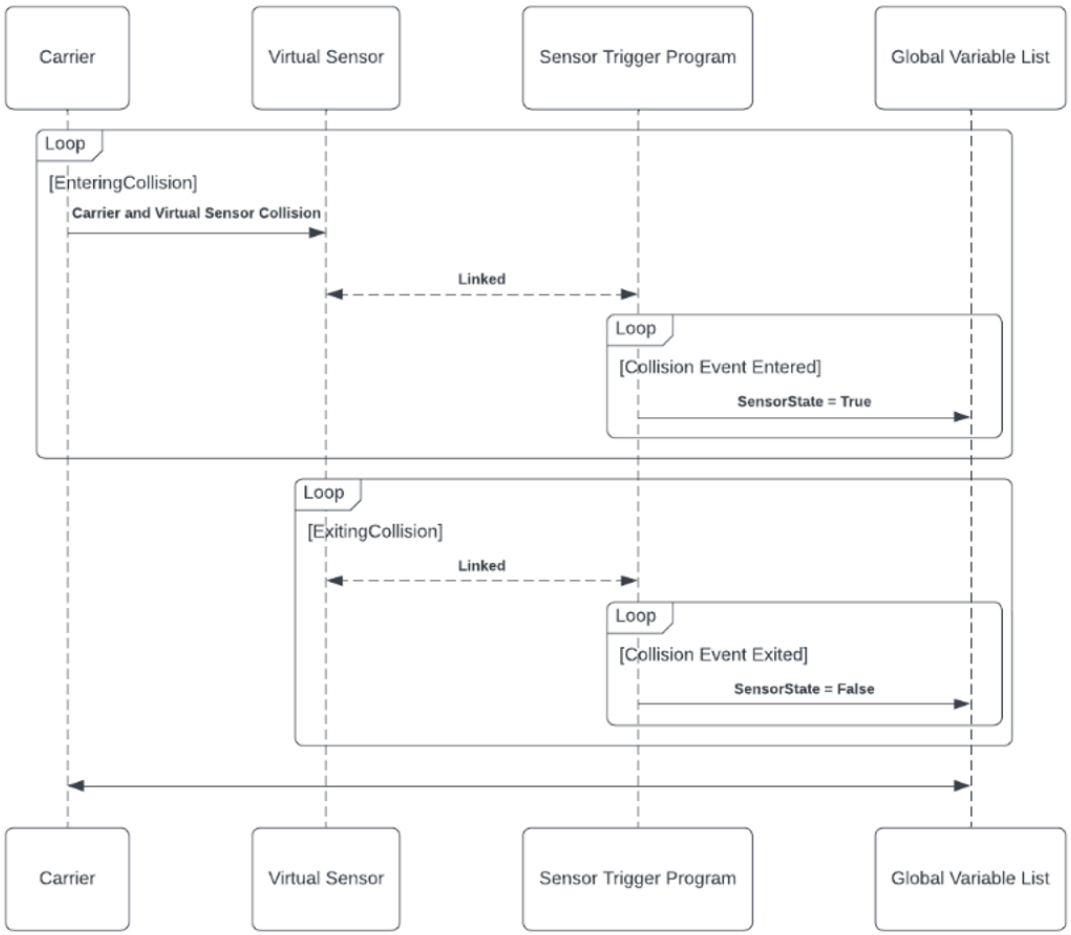


Figure 3.4: Showing virtual sensor trigger acknowledgement.

This triggers a collision event inside the STP which sets that sensor's state to True in the GVL. A block diagram example of this can be seen in Figure A.6. A secondary event is triggered when those box colliders are no longer in contact. This 'on-exit' event sets the sensor's state to False in the GVL, Figure A.7.

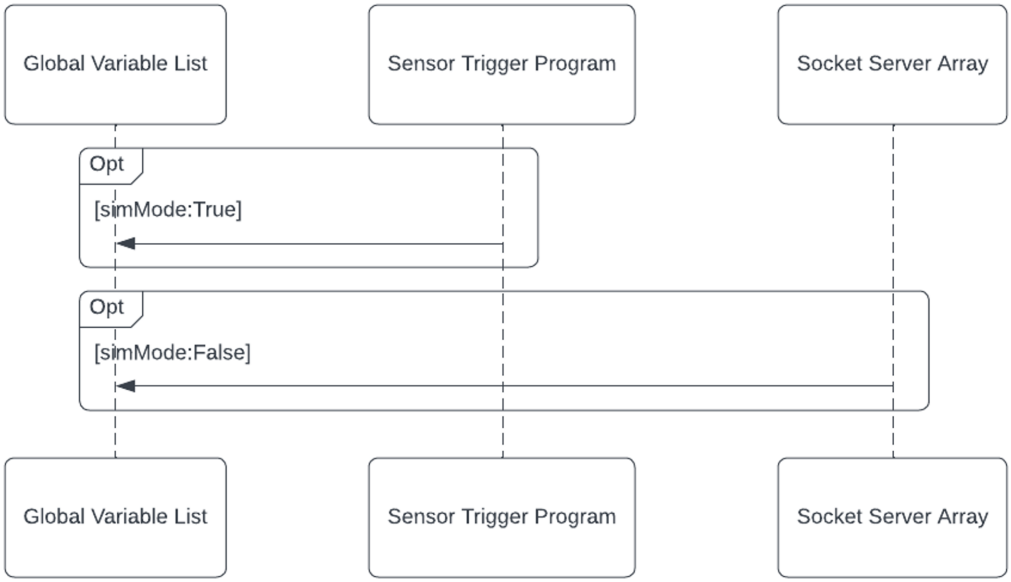


Figure 3.5: Shows that if simulation is active, the Sensor Trigger Program is also active, else, the socket server array is active.

Running the system in this way allows the GVL to remain the master and is always referred to, no matter if in Simulation Mode or not. Using this name-collection technique allows a singular script to be used for every binary sensor on the system, enabling the modularity of this digital twin framework. This creates a system that can output data that is the same shape as the physical system, allowing efficiency testing when an in-software unit of time is not equal to 1 second. All the other system processes have their own scripts that match the PLC code in function, written in C#, and ultimately talking to the master script's GVL. It is shown in Figure A.2 how the software checks which mode the system is in before calling the dataDist function. The function layout can be seen on Figure A.3. To convert this virtual standalone system into one that can live-track the physical system, the STP must have its write access changed:

- write access to the GVL is revoked
- socket server array's write access is enabled.

This ensures that the GVL receives fresh data via the socket server array from the physical sensors. All programs running inside the digital twin are not aware if they are receiving data from a simulated sensor or a physical one.

3.4.1 The tracking system

The ability to track a carrier from the physical space to the virtual space is a challenge since there are no outside tracking cameras. The physical conveyor belt has optical encoders mounted to each of the motors driving them. In the original PLC code, the encoder is read as a pulse, solely serving as a 'keep alive' signal, confirming the belt is turning. When pulses are not being received, a system alarm triggers on the HMI to warn of an issue with the conveyor belt. Knowing the resolution of the encoders, it is possible to find an Rotations per minute (RPM) value. Coupling this will the belt-roller circumference and the length of the conveyor belt, a carrier velocity can be extrapolated in metres per second. Using this velocity data, it can be concatenated with the rest of the data packets being sent by that specific PLC. F = Encoder pulse frequency, R = Encoder Resolution = 8, S = Revolutions per second, T = conveyor travel per encoder wheel resolution = 94.2mm, V = Carrier Velocity in m/s

$$S = F/R \tag{3.1}$$

$$V = S * T * 0.001 \tag{3.2}$$

On the virtual system side, this carrier velocity can be directly translated into a co-ordinate displacement per frame function, meaning the carrier in the system will track at the same speed as the physical system. To account for possible perturbations of the outside world and to continuously validate the exact position of the carrier, the induction and capacitive sensors are used in conjunction with the Radio Frequency Identification (RFID) readers, situated at the centre of each conveyor belt.

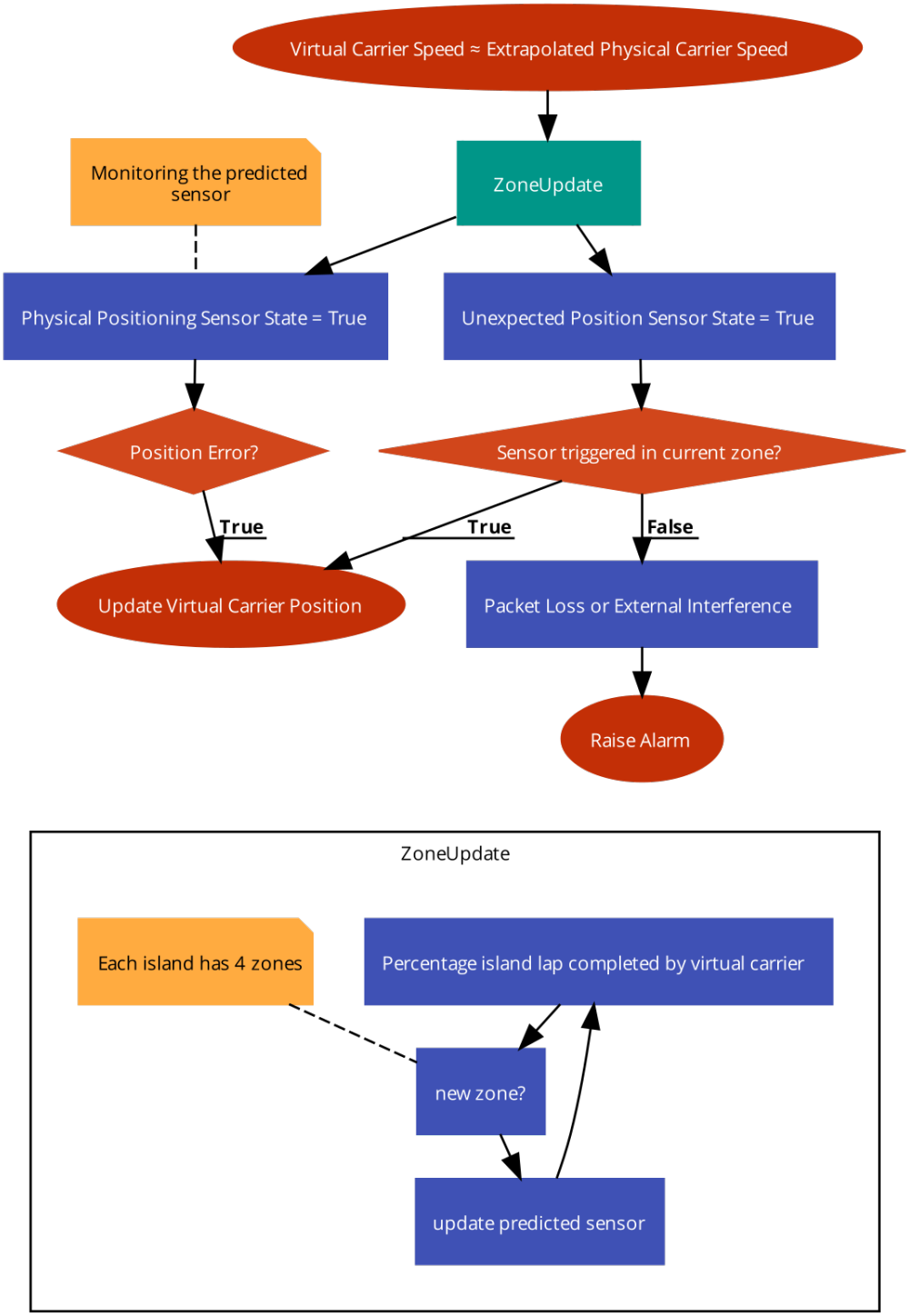


Figure 3.6: Flow diagram showing how carrier position is approximated.

If a capacitive sensor on the physical system is triggered before the carrier in the virtual system has reached a virtual sensor, the carrier position and speed is offset by a rolling average scaler. This acts as quick fix position auto correct. It works equally well if

the carrier is ahead. However, this auto correcting system is limited since unpredicted collisions on the physical system are possible. Therefore, if a physical system carrier is halted, but the conveyor is still running, the virtual carrier will hang at the next nearest capacitive sensor and trigger a warning after a certain amount of time. To start the tracking of a physical carrier, the virtual system needs a starting point for the seemingly meaningless position data that is being received. When a physical carrier finally pauses at one of the many RFID readers, that data is sent by the PLC and is received by the master script. The master script checks if a carrier with that specific ID has already been spawned into the virtual system, if it hasn't, a carrier will spawn and start moving on the physical pneumatic piston, releasing the physical carrier. Shown in Figure A.4. This carrier will then be tracked. If no order has been placed, the carrier will oscillate aimlessly around the island. With the physical system occupied by 'carriers with orders' and 'carriers with no orders', bottle necks and queues start to form. Carriers would collide with the one in front until it is released from the pneumatic stopper. Having a location index function means that carriers are constantly in their own zone and have that number associated with them in the Carrier Class. Not having this meta data addition can cause system unpredictability, where carriers can swap places or jump to different areas. This is shown in Figure 3.6 as a block diagram or Figure A.5 as a code snippet.

3.4.2 The Multithread server

As mentioned above, the multithread server script is an array of socket servers. Socket connections require a server and a client. The servers accept connections from devices. A device continuously look for, and tries to connect to, its allocated server. Since there are eight PLCs in the physical system (two Codesys and six Siemens), data needs to be filtered into the software in a fast, efficient way so the latency between the virtual system and the physical system is minimal. The algorithm used is much like you would see in a modern online multiplayer game.

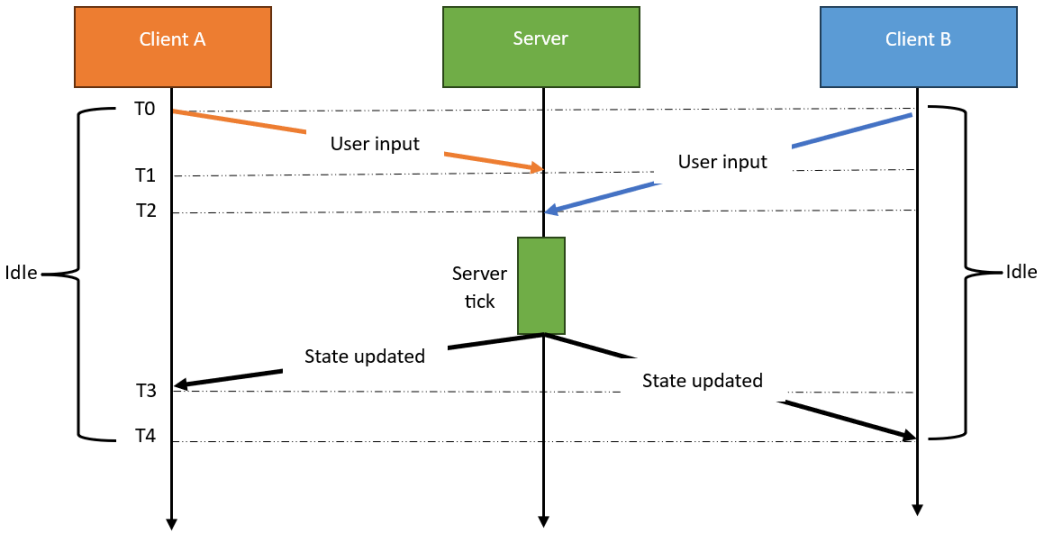


Figure 3.7: Server/Client communication example [4]

Data must be funnelled into the system in parallel, since a huge amount of data is always required. Upon starting the system in live tracking mode, eight threaded servers are started, each on a unique port number that has been predetermined and is linked in the PLC code. All PLCs connect to this and start sending data. At every clock cycle of the PLC, a data packet is created and sent to the designated server. When the server receives the data packet, it compares its contents to the previous data packet. If the information is identical to the packet before, there is no need for computer to parse the data and rewrite every variable concerned, as there is clearly no change to the virtual system. A method like this is shown in Figure 3.7, that is commonly used in online multiplayer games.

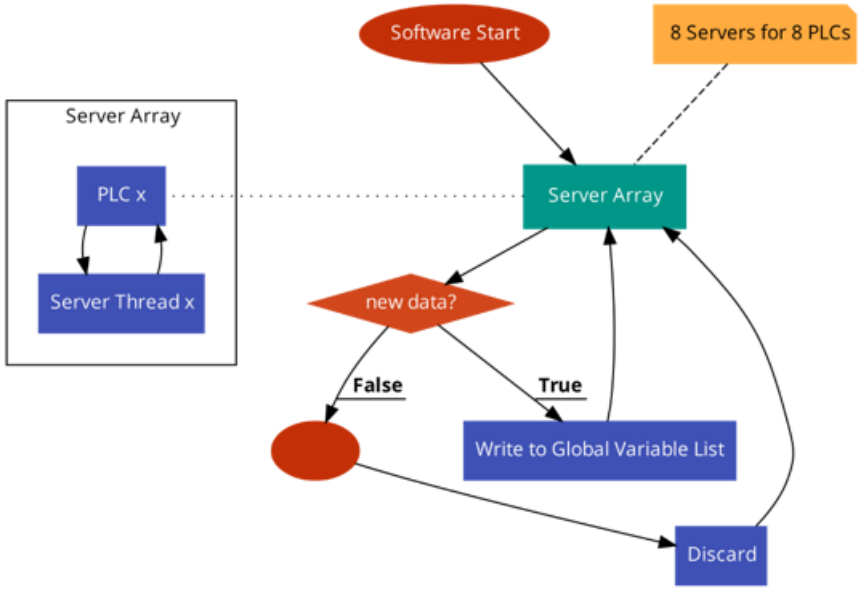


Figure 3.8: Flow diagram showing how data is passing from the PLCs to the DT.

As shown in Figure 3.8, once a new data packet has been received, it will be picked up by the master script that is polling the multithread server script for new data. Here, the data is passed into a data distribution function. This is a simple algorithm that checks the station ID in the data packet and routes it to the specific parsing function. The data is split up by its specific index and written to the corresponding global variables.

3.5 Predictions and error cases

3.5.1 Air

Air volume sensors coupled with two Siemens S7-1200 PLCs have been added to the Festo CP-Lab to enable it to monitor air pressure vs air volume, specific to each station. As the number of cycles of the mechanical components and fittings approach the predicted end-of-life zone, the mechanical switching time will increase, leakage can occur [33]. Using the air pressure and air volume relationship vs time, the mechanical switching time can be plotted. If the component is beginning to fail, it can be predicted using time-series regression. Detecting leaks is less complex as the air pressure vs air volume usage ratio will change. Air pressure is regulated to 6 bars and will remain there, however, air volume will increase if there is a leak, as more air is needed to operate the

system.

3.5.2 Energy

The CP-Lab ships with an energy monitor that returns Power, Current and Voltage. Using time-series regression, trends can be spotted, and decisions can be made about the health of the system. For instance, imagine the bearings on the conveyor belt are beginning to fail due to a build of up grime from the belt. In this scenario, the added friction will cause the motors to work harder, building temperature and power usage, both of which can be used as indicators of potentially imminent failure.

3.5.3 Collisions and Foreign Objects

If a foreign object were to impede a carrier's movement, even for only a few seconds, speed would be reduced, or the carrier would stop completely. As 'timeout' functions have been built into each location, based on acceptable time frames, if the carrier doesn't make the next sensor, an error flag will be shown and logged, and the virtual carriers will be halted. If the obstacle was then removed, the virtual carriers would start moving again once a capacitive sensor is triggered. The virtual carrier ID is be compared against the physical ID sent by the next RFID scanner. If these are different, the digital twin needs to be reset. If not, the system will run as normal.

3.6 Data Logging and Output

When the data arrives, the data + timestamp is sent to a buffer inside the logging function. The buffer is periodically cleaned, and the CSV data file is updated. This ensures the computer's CPU is minimally affected, at the cost of RAM. If an order is placed in the Festo MES system, the digital twin will initialise a new CSV file and start a system thread so it can work in parallel to the main software. On every update cycle, all sensor data is written into a buffer along with a timestamp. The buffer is then written to the CSV. Once an order is complete, the CSV is closed, and the file name is changed to the order number that was processed. A Python script was created to talk with the cloud. This script polls the folder in which the CSV files are stored, searching for new additions. Once a new file is found, the CSV is parsed into sensor arrays in tuple form with the data + timestamp. The meaningful data is selected from these arrays that has been predetermined by the user and uploaded to the cloud. Cloud data structure:

- carrier position against time
- air volume/pressure
- power consumption
- each station's processing time
- temperature

Using the cloud data structure, the received data allows predictions to be made on system performance and efficiency.

3.7 Robotino

The Robotino is an automatic guided vehicle (AGV) that transfers carriers from one island on the CPS to the other. The robot communicates over a socket server with Festo Fleet Manager; this talks to the MES software used by the rest of the CPS. The robot sends all sensor data, including the Light Detection and Ranging (LiDaR) and odometry data. Unfortunately, to access the odometry data via an API, the robot must run a specific firmware version that isn't compatible with the rest of the CPS, therefore applying the live-tracking algorithm cannot be carried out. In the live-tracking software, Simulation Mode is force-enabled to allow the virtual Robotino scripts to run.

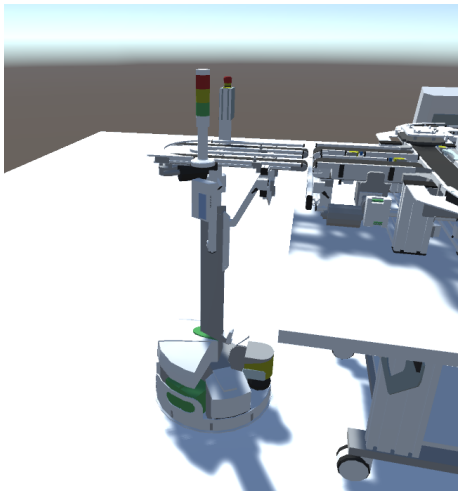


Figure 3.9: Virtual Robotino

Chapter 4
Results and Discussion

The final system can produce full and detailed data about every order that comes off the production line, whether it be historical or instantaneous. Meaningful data can be selected and uploaded to any cloud computing service, if required. This is due to software handling in a separate system, running in parallel.

4.1 Tracking

Full live tracking works with a singular carrier on the production line. When carrying out multi-carrier tests, it was discovered that there are scenarios where the carrier tracking will merge two virtual carriers inside each other, an example for this is shown in Figure 4.1.

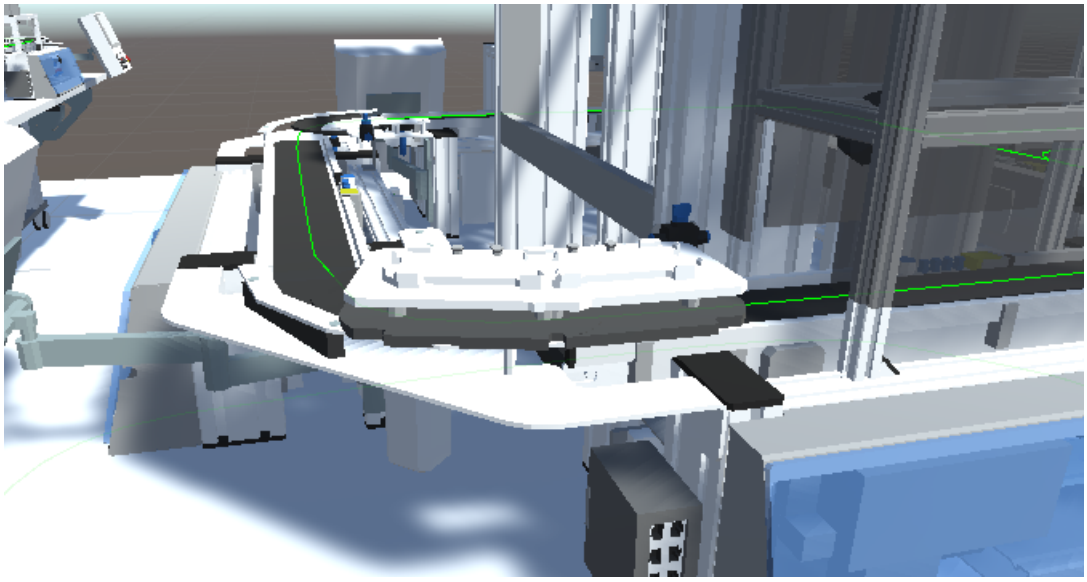


Figure 4.1: Carriers Clipping into each other

The causes for these problems are shared between holes in the logic of the software and the condition of the equipment being used. A remedy for this is to add more advanced rigid body physics. The addition of which would mean a further revision to the carrier approximation algorithm to account for slight obstacles in the carrier's path instead of terminating and flagging an alarm when tracking is lost.

4.1.1 Physical vs Virtual system performance

Clearly, a virtual system can never truly have the same quirks and characteristics as a physical system but the idea is to get a close as necessary to have data that is reliable

but also does not cost too much in performance. Figure 4.4 below shows the average journey time of a carrier with no order placed upon it, therefore sensor data is only that of the conveyor system.The trigger points are three sensors mounted upon each conveyor, making 22 sensors total - two stations do not require three sensors.

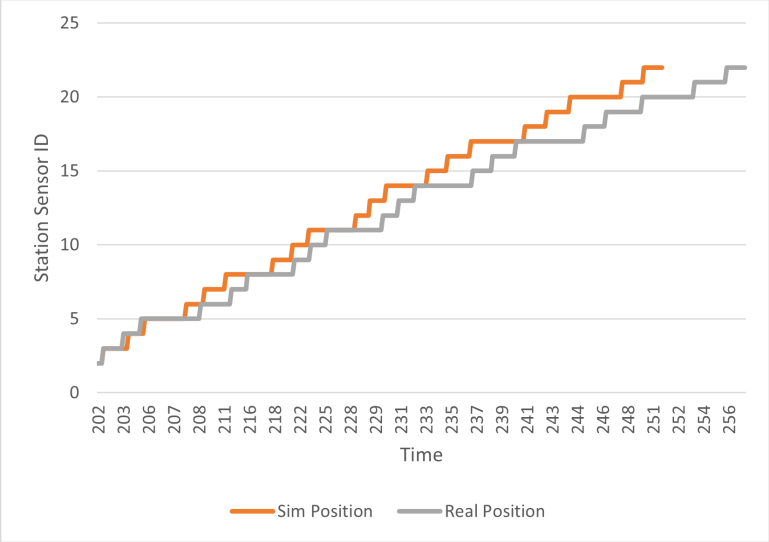


Figure 4.2: A graph to show one lap of the CPS in simulation vs real-life (robotino is excluded)

The simulated system is faster per lap but tends to even out when orders are running because the Robotino is the main bottleneck of the system. Depending on the scale of production, this speed discrepancy could scale up and produce unusable data if the system was used for forecasting. This would only be possible once the Robotino was upgraded to having an accessible API. Further revision should be made to the system to add the likely instabilities that the physical system possesses. At least, that is one conclusion that can be drawn from this graph - the simulated system has consistent stop and start times where the physical system has naturally more variance in stop times. This result is localised just to full simulation vs headless real time. The live tracking algorithm works will with position variance being minimal as it is constantly being micro-adjusted.

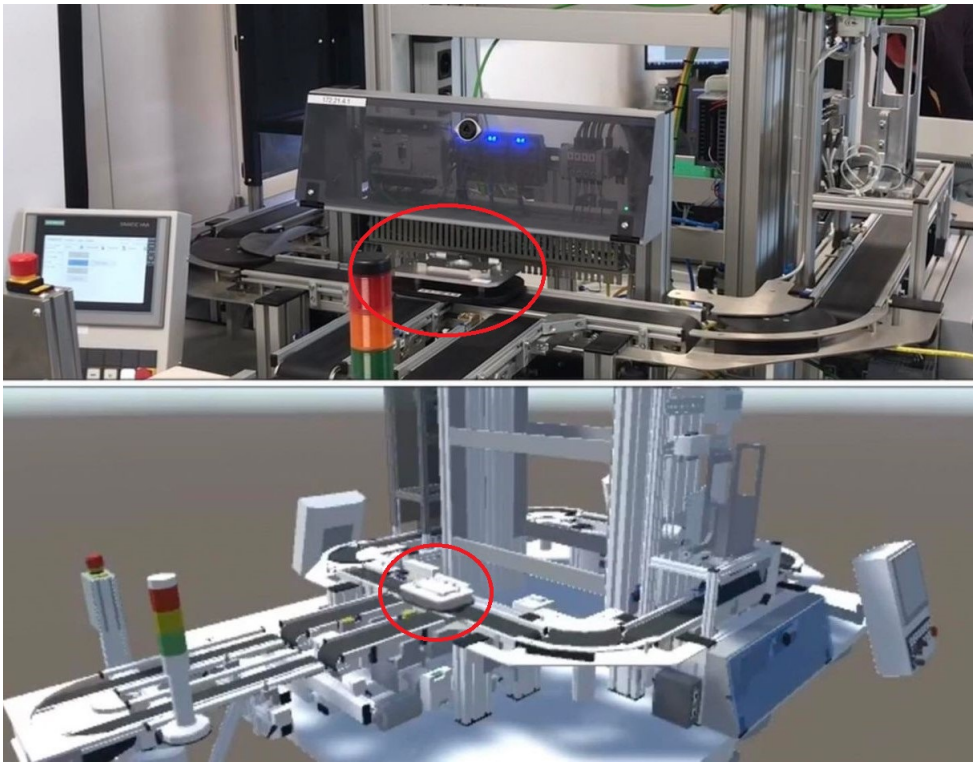


Figure 4.3: Showing tracking accuracy in live tracking mode between the physical and the simulated system.

4.2 Flaws in CPS functionality

Consistent and continuous testing of the heating station proved to be a challenge when trying to understand how to spot trends in component degradation. Festo's algorithm is as follows:

Algorithm 2 Festo's heating station algorithm

1: H: hysteresis = 6, A: ambient temperature, T: Target = 50
2: **if** $A < (T - H/2)$ **then**
3: Switch on heater
4: **end if**
5: **if** $A >= (T - H/2)$ **then**
6: Wait for 5 seconds at target temperature
7: **end if**

The overall time a FMP spends inside the heating station fluctuates majorly depending on density of orders in that area of the production line.

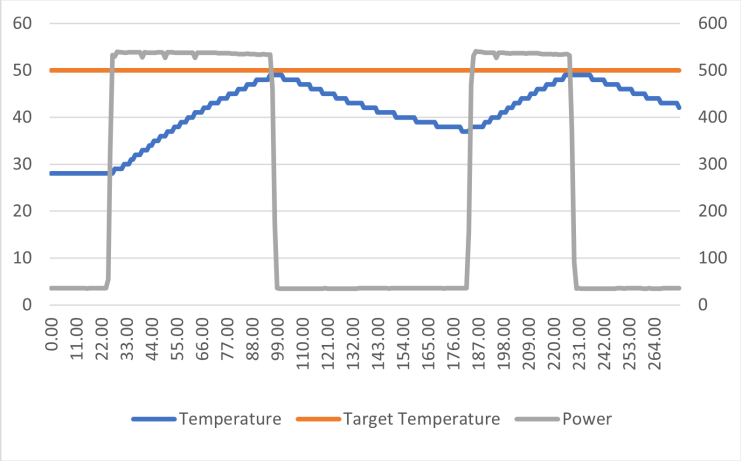


Figure 4.4: A graph to show the inconsistency in default heating times

As can be seen from the graph, the FMP is in the heating station for considerably less time on the second order compared to the first. A number of ways could fix this issue, some are better than others depending on throughput. The heating station (Figure A.8) could be left powered on all the time, but this will waste energy, or they could increase the heating time at target temperature, but this will slow down overall takt time.

4.3 Communication

Digital Twin to MES communication was tested with a UI created in Node-Red, using OPC-UA communication directly with the CPS. This can operate at a remote HMI but can also be interacted with directly from within the Digital Twin software. A packet sent from the digital twin is sent to Node-Red, which in turn is sent to the MES software. Communication between the MES and the digital twin was tested with the robust parallel socket server. Initial efforts on this were to recycle code from a blockchain project using an array of socket servers, written in Python, sending the necessary data into Unity after processing. This was overly heavy on the computation power available at the time as everything was serialised. One packet had to be decoded at one time. Updating the PLC software, the controllers would only send relevant packets of data, as TCP was now being used instead of UDP. Rewriting the server array in Unity and allocating each socket connection an individual system thread means data processing is more than eight times faster than the initial solution.

4.3.1 Latency in live tracking

There is always going to be some sort of latency in a project like this, nothing is instantaneous. Speeds vary due to the network setup, quality of hardware and processing power of the machine running the DT software. The computer in this project is a mid-range, off the shelf laptop. Setting a timer from when a sensor is actuated on the CPS to when it has been processed on the DT comes to an average around 200ms.

4.3.1.1 PLC sampling and balancing

Eight PLCs are on the network. Two of which are running Codesys from Festo. The multithreaded server script that is running on the DT expects data to be arriving at a constant speed from all eight controllers. Since the Codesys PLCs are running at a different cycle time, the speed at which they could provide sensor status updates was far faster then the Siemens PLCs. The network would get saturated with codesys data, creating an overflow in Unity, causing data to be missed from the other PLCs. Due to the limited performance available from the laptop, turning the publishing time down on the codesys PLCs proving a viable fix and data flowed in at around 100 samples per second. Shown below is the average cycle time coming from the Siemens PLC.

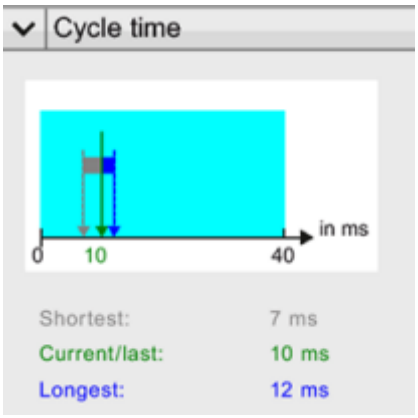


Figure 4.5: ET200sp Cycle time

4.4 Failure forecasting and predictive maintenance

4.4.1 Leak point detection

Six Festo flow sensors were installed on the CPS. If a leak occurs, the data from these allow the digital twin to pinpoint which station has an air issue. By monitoring average

air consumption per cycle (litre/s) and current air pressure at the regulator, a leak point can be detected. If air consumption rises, and regulator pressure remains constant, the cause must be a leak. This is an improvement over the original system that could detect if there was a leak in the system somewhere, but not know where, since the flow meter was at the compressor.

4.4.2 Obstruction detection

Obstruction detection works on the basis that the live-tracking system is reliable. The idea was that the if the carrier didn't reach the next expected sensor, or the physical RFID reader read a different carrier ID from the one expected, then there was an issue. However, the live-tracking system is not robust as there are still some glitches. During testing, it was also found that the Festo RFID readers were also unreliable as they would occasionally write data to the wrong carrier. For example, carrier ID 5 would be read as carrier ID 3.

4.4.3 Predicting component failure

Predicting component failure on a system with no historical data prior to the digital twin upgrade means predicting failure from a retro fit is almost impossible. Covid19 required work on this project to be carried out remotely. The CPS was left turned on and remote access was set up to monitor the noise levels in the lab and live images. After a month, the manual station conveyor belt failed. The manual station pneumatic stopper stopped retracting and releasing the carrier round the island, effectively turning the conveyor into a bench sander. Microparticles of plastic worked their way into the conveyor bearings, increasing the load on the motor. The conveyor eventually stopped when the motor driver's current limit was reached. A photo of the aftermath is shown in Figure 4.6. As well as monitoring actuator firing times, assessing the condition of actuators and motors through current draw is a key indicator of friction.

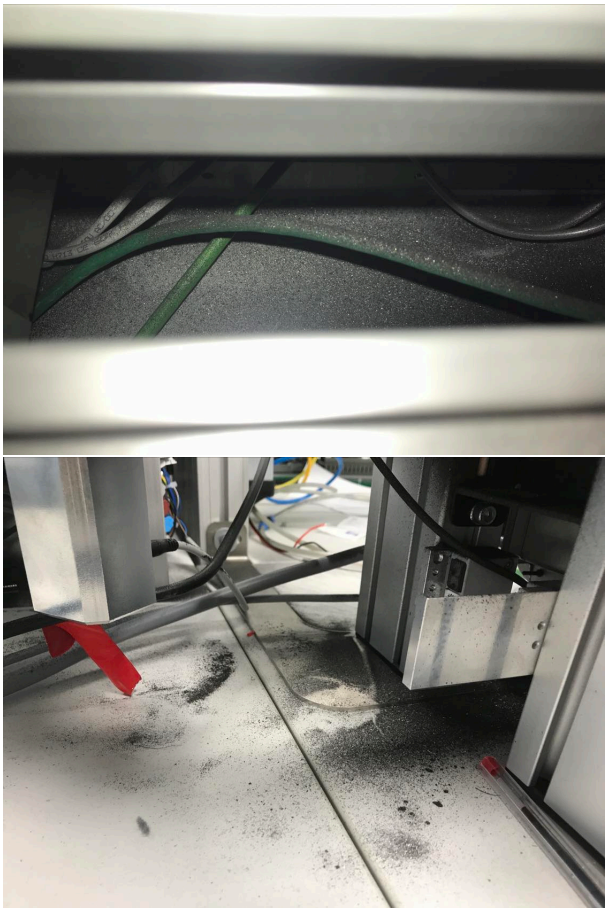


Figure 4.6: Aftermath from conveyor failure

Chapter 5
Conclusion and Future Work

5.1 Conclusion

This research aimed to investigate and develop a digital twin framework, working in real-time to replicate and monitor an industry 4 smart factory system. It has shown that it is possible for digital twin technology to be applied to a smart factory system, demonstrating that live tracking is possible and in a modular, framework based approach. It has been demonstrated that it is possible to incorporate all the elements of the previously stated definition of the DT used in this paper.
The first developmental mile stone in becoming successful and a key development was being able to efficiently receive and process high speed data from eight different controllers fast enough as to not hinder the main functionality of the system. The second being the main live tracking algorithm. Approximating multiple carrier positions without the aid of sight. Third, creating a virtual system that behaves identically to the physical system and being able to run independently. It should be taken that the digital twin focus is reducing potential unscheduled downtime of a system, rather than increasing product throughput. The digital twin successfully carried out two-way communication to mirror the physical system and to run independently. It has sent time-series data for analysis in the cloud, and it has initial groundwork laid out for various predictive maintenance features.

5.2 Way-Forward

This research should be used as a building blocks framework to build upon for this to be viable and useful in an actual facility. Further work is required to include effective predictive maintenance strategies and better alarms/notifiers if immediate work on the system is required. Packaging this software in a usable format for deployability and easy-of-use should also be a consideration if taking this project further.
It has been shown that digital twin technology is the next stage in smart manufacturing and industry 4 and creating a framework for this next stage is vital for those that want to stay competitive.

References

[1] M. Y. Stefan Mihai, "Digital twins: A survey on enabling technologies, challenges, trends and future prospects.," *Communications Surveys Tutorials*, vol. 24, 2022.

[2] M. D. Julia Robles, Cristian Martín, "Opentwins: An open-source framework for the development of next-gen compositional digital twins," *29th International Conference on Flexible Automation and Intelligent Manufacturing(FAIM2019), June 24-28, 2019, Limerick, Ireland*, 2023.

[3] S. C. R. A. S. V. K. Shah, T. V. Prabhakar and V. K. T, "Construction of a digital twin framework using free and open-source software programs," *IEEE Internet Computing*, vol. 26, 2022.

[4] Q. W. Lim, "How do multiplayer games sync their state part 1." `https://medium.com/@qingweilim/how-do-multiplayer-games-sync-their-state-part-1-ab72d6a54043`, 2017.

[5] D. S. E. H. Glaessgen, "The digital twin paradigm for future nasa and u.s. air force vehicles," *53rd Structures, Structural Dynamics, and Materials Conferenc*, 2012.

[6] G. Immerman, "The real cost of down time in manufacturing." `machinemetrics.com`, 2018.

[7] Infineon, "Predictive maintenance: Smart servicing." `https://www.infineon.com/cms/en/discoveries/predictive-maintenance/`, 2021.

[8] R. Schiavullo, "'reflekt has built up an enterprise augmented and mixed reality ecosystem for maintenance, operations." `https://twittertechnews.com/virtualreality/reflekthas-built-up-an-enterprise-augmented-and-mixed-reality-ecosystem-fo` 2019.

[9] L. Sharpe, "'railway bridge gets its 'digital twin'.." https://eandt.theiet.org/content/articles/2018/12/railway-bridgegets-life-extending-digital-twin/, 2019.

[10] P. v. Manen, "Digital twins in f1 and the built environment." https://www.ice.org.uk/news-insight/news-and-blogs/ice-blogs/the-civil-engineer-blog/digital-twins-in-f1-and-the-built-environment, 2019.

[11] E. N. Lorenz, "The essence of chaos.," *Taylor Francis*, 1995.

[12] pbc today, "How a formula 1 mindset can revolutionise digital twins in construction." https://www.pbctoday.co.uk/news/digital-construction/f1-construction-digital-twins/98529/, 2021.

[13] T. Menteth, "The tunnelling revolution." https://www.geplus.co.uk/news/podcast-the-tunnelling-revolution-14-01-2022/, 2022.

[14] D. S.-M. Gawel, ". a validated single-cell-based strategy to identify diagnostic and therapeutic targets in complex diseases.," *Genome Med.*, 2019.

[15] R. S. Mohd Javaid, Abid Haleem, "Digital twin applications toward industry 4.0: A review," *Cognitive Robotics*, 2023.

[16] S.-J. S. Sanjay Jaina, Guodong Shaob, "Manufacturing data analytics using a virtual factory representation," *International Journal of Production Research*, 2017.

[17] C. M. E.P.Hinchy, N.P. O'Dowd, "Using open-source microcontrollers to enable digital twincommunication for smart manufacturing," *29th International Conference on Flexible Automation and Intelligent Manufacturing(FAIM2019), June 24-28, 2019, Limerick, Ireland*, 2019.

[18] MathWorks, "Thingspeak documentation." https://uk.mathworks.com/help/thingspeak/, 2019.

[19] Siemens, "'connectivity - mindsphere documentation." https://documentation.mindsphere.io/resources/html/getting-connected/en-US/119185827339.html, 2019.

[20] E. Griffith, "What is cloud computing?." https://uk.pcmag.com/networking-communications-software/16824/what-is-cloud-computing, 2022.

[21] L. C. A. Galletta, "A cloud-based system for improving retention marketing loyalty programs in industry 4.0: A study on big data storage implications.," *IEEE Access*, vol. 6, 2018.

[22] N. B. Markopoulos, "A review on the readiness level and cyber-security challenges in industry 4.0.," *2017 South Eastern European Design Automation, Computer Engineering, Computer Networks and Social Media Conference.*, 2017.

[23] Britannica, "Stuxnet." https://www.britannica.com/technology/Stuxnet, 2023.

[24] T. Burke, "Automation world 2009, 'top six benefits of opc ua for end-users." https://www.automationworld.com/home/article/13299514/top-six-benefits-of-opc-ua-for-endusers, 2009.

[25] G. S. M. Silveira Rocha, "Performance comparison between opc ua and mqtt," *2018 Workshop on Metrology for Industry 4.0 and IoT. Brescia: pp. 175-179, doi: 10.1109/METROI4.2018.8428342*, 2018.

[26] SSnap7, "Siemens communications overview.." http://snap7.sourceforge.net/, 2019.

[27] N. Jennings, "Python sockets." https://realpython.com/python-sockets/, 2023.

[28] D. D. Clark, "Window and acknowledgement strategy in tcp.," *MIT Laboratory for Computer Science Computer Systems and Communications Group.*, 1982.

[29] M. Cook, "Tcp vs udp: What's the difference?." https://www.lifesize.com/blog/tcp-vs-udp/, 2017.

[30] A. EKonrad, "Simulation of mobile robots with unity and ros: A case-study and a comparison with gazebo," *urn:nbn:se:hv:diva-14019*, 2019.

[31] Siemens, "Siemens nx." plm.automation.siemens.com/global/en/products/nx/, 2019.

[32] J. Ovenden, "Beginner's guide to digital twin technology." https://channels.theinnovationenterprise.com/articles/a-beginner-s-guide-to-digital-twin-technology/, 2019.

[33] Festo, "Product service lift as festo." https://www.festo.com/net/SupportPortal/Files/293239/Product_Service_Life_at_Festo_135502_EN_09_2013.pdf, 2019.

Appendices

Appendix A

Code Snippets

```
while (keepReading)
{
    bytes = new byte[24];
    int bytesRec = handler.Receive(bytes);
    //Debug.Log("Received from Server");

    if (bytesRec <= 0)
    {
        keepReading = false;
        handler.Disconnect(true);
        break;
    }

    data = Encoding.ASCII.GetString(bytes, 0, bytesRec);

    if (data.Equals(old) == false)
    {
        Debug.Log(data);
        old = data;
        magFData = data;

        //GameObject.Find("Main Camera").GetComponent<runInSimMode>().
    }
```

Figure A.1: Filtering so the same data isn't processed twice

```
if (simMode == false)
{
    mainOrderStucturePhysical();

    dataDist(GameObject.Find("Main Camera").GetComponent<MultiThreadServer170520>().manData); //dataDist is data distribution.
    dataDist(GameObject.Find("Main Camera").GetComponent<MultiThreadServer170520>().magFData);
    dataDist(GameObject.Find("Main Camera").GetComponent<MultiThreadServer170520>().camData);
    dataDist(GameObject.Find("Main Camera").GetComponent<MultiThreadServer170520>().codesys1Data);
    dataDist(GameObject.Find("Main Camera").GetComponent<MultiThreadServer170520>().codesys2Data);
    dataDist(GameObject.Find("Main Camera").GetComponent<MultiThreadServer170520>().magBData);
    dataDist(GameObject.Find("Main Camera").GetComponent<MultiThreadServer170520>().pressData);
    dataDist(GameObject.Find("Main Camera").GetComponent<MultiThreadServer170520>().heatData);
}
```

Figure A.2: Showing how the dataDist function with new data at the input

```
public void dataDist(string data)
{

    string[] splitData = data.Split(',');
    //Debug.Log(splitData[0] + " splitdata");
    try
    {

        if (splitData[0].Equals("1"))
        {
            magFrontStartInduction = splitData[1].Equals("1");
            magFrontStopInduction = splitData[2].Equals("1");
            magFrontCarrierRelease = splitData[3].Equals("1");
            magFrontEndInduction = splitData[4].Equals("1");
            magFrontRFID = Int32.Parse(splitData[5]);
            magFrontConveyorSpeed = Int32.Parse(splitData[6]);

            magFrontLiftUpS = splitData[7].Equals("1");
            magFrontLiftDownS = splitData[8].Equals("1");
            magFrontLiftUpQ = splitData[9].Equals("1");
            magFrontLiftDownQ = splitData[10].Equals("1");
            magFrontCylinderClamp = splitData[11].Equals("1");

        }
```

Figure A.3: Snippet of the main dataDist function

```
if (manualCarrierRelease == true && manualSpawnRunOnce == false && carrierArray.Contains(manualRFID) == false)
{
    ID = manualRFID;
    GameObject clone = Instantiate(carrierPrefab) as GameObject;
    carriers[ID] = clone;
    carriers[ID].GetComponent<follower>().carrierID = ID;
    carriers[ID].GetComponent<follower>().spawnLocation = 2;
    carriers[ID].GetComponent<follower>().order = false;
    carriers[ID].GetComponent<follower>().goStop = true;
    carriers[ID].GetComponent<follower>().speed = speed;
    carriers[ID].GetComponent<follower>().pauseTime = 5000;
    carriers[ID].GetComponent<follower>().manualConvCase = 10;
    carriers[ID].GetComponent<follower>().manualSpawnInit = true;
    carriers[ID].name = carriers[ID].name + ID.ToString();
    carrierArray[ID] = ID;
    manualSpawnRunOnce = true;
}

if (manualCarrierRelease == false && manualSpawnRunOnce == true)
{
    manualSpawnRunOnce = false;
}
```

Figure A.4: Instantiation of a carrier

```
void location()
{
    if (percentLapFirstIsland <= 67 && percentLapFirstIsland > 38f && pathMode == 1) { currentLocation = 1; }
    if (percentLapFirstIsland <= 92.5f && percentLapFirstIsland > 69 && pathMode == 1) { currentLocation = 2; }
    if (percentLapFirstIsland <= 17f || percentLapFirstIsland > 94f) { if (pathMode == 1) { currentLocation = 3; } }
    if (percentLapFirstIsland <= 36.5f && percentLapFirstIsland > 19 && pathMode == 1) { currentLocation = 4; }
    if (toRobotPercentLap <= 13.25f && percentLapFirstIsland > 35 && pathMode == 2) { currentLocation = 5; }
    if (toRobotPercentLap <= 60.75f && toRobotPercentLap > 13.25f && pathMode == 2) { currentLocation = 6; }
    if (toRobotPercentLap <= 84.25f && toRobotPercentLap > 61f && pathMode == 2) { currentLocation = 7; }
    if (pathMode == 3) { currentLocation = 8; }
    if (percentSecondIslandRobotinoLap >= 100 && pathMode == 3) { currentLocation = 9; }
    if (percentLapSecondIsland <= 67 && percentLapSecondIsland > 38f && pathMode == 4) { currentLocation = 10; }
    if (percentLapSecondIsland <= 92.5f && percentLapSecondIsland > 69 && pathMode == 4) { currentLocation = 11; }
    if (percentLapSecondIsland <= 17f || percentLapSecondIsland > 94f) { if (pathMode == 4) { currentLocation = 12; } }
    if (percentLapSecondIsland <= 36.5f && percentLapSecondIsland > 19 && pathMode == 4) { currentLocation = 13; }
    if (percentSecondIslandRobotinoLap >= 59 && GameObject.Find("Main Camera").GetComponent<runInSimMode>().robotinoCarrierStop == true) { currentLocation = 14; }
}
```

Figure A.5: Indexing carrier location

```
void OnTriggerEnter(Collider other)
{
    if (GameObject.Find("Main Camera").GetComponent<runInSimMode>().simMode == true)
    {
        caseSwtich = gameObject.name;
    }


    switch (caseSwtich)
    {
        case "magFrontStartInduction":
            GameObject.Find("Main Camera").GetComponent<runInSimMode>().magFrontStartInduction = true;
            break;
        case "magFrontStopInduction":
            GameObject.Find("Main Camera").GetComponent<runInSimMode>().magFrontStopInduction = true;
            break;
        case "magFrontEndInduction":
            GameObject.Find("Main Camera").GetComponent<runInSimMode>().magFrontEndInduction = true;
            break;

        case "magFrontTop":
            GameObject.Find("Main Camera").GetComponent<runInSimMode>().magFrontTop = true;
            break;
        case "magFrontBottom":
            GameObject.Find("Main Camera").GetComponent<runInSimMode>().magFrontBottom = true;
            break;
```

Figure A.6: Snippet of the sensor trigger script

```
public bool manualStartInduction = false;
public bool manualStopInduction = false;
public bool manualEndInduction = false;
public int manualRFID = 0;
public int manualConveyorSpeed = 0;
public bool manualCarrierRelease = false;
public bool manualHMIScreenDisplay = false;
public bool camInspectStartInduction = false;
public bool camInspectStopInduction = false;
public bool camInspectEndInduction = false;
public bool camInspectCarrierRelease = false;
public int camInspectRFID = 0;
public int camInspectConveyorSpeed = 0;
public bool codesys1StartInduction = false;
public bool codesys1StopInduction = false;
public bool codesys1EndInduction = false;
public bool codesys1ToRobotino = false;
public bool codesys1FromRobotino = false;
public bool codesys1CarrierRelease = false;
public int codesys1RFID = 0;
public bool robotinoIslandSensor = false;
public bool robotinoCarrierStop = false;
public bool firstIslandRobotino = false;
public bool secondIslandRobotino = false;
public bool magFrontStartInduction = false;
public bool magFrontStopInduction = false;
public bool magFrontEndInduction = false;
public bool magFrontCarrierRelease = false;
public bool magFrontLiftUpS = false;
public bool magFrontLiftDownS = false;
public bool magFrontLiftUpQ = false;
public bool magFrontLiftDownQ = false;
public bool magFrontCylinderClamp = false;
public bool magFrontTop = false;
public bool magFrontBottom = false;
public int magFrontRFID = 0;
public int magFrontConveyorSpeed = 0;
public bool magBackStartInduction2 = false;
public bool magBackStopInduction2 = false;
public bool magBackEndInduction2 = false;
public bool magBackCarrierRelease2 = false;
public bool magBackTop = false;
public bool magBackBottom = false;
public int magBackRFID2 = 0;
public int magBackConveyorSpeed2 = 0;
public bool pressStartInduction2 = false;
public bool pressStopInduction2 = false;
public bool pressEndInduction2 = false;
public bool pressCarrierRelease2 = false;
public float pressTargetPressureN = 70;
public float pressCurrentPressureN = 0;
public float pressCurrentTime = 2;
public float pressTargetTime;
public bool pressWorkpieceCheck = false;
public int pressRFID2 = 0;
public int pressConveyorSpeed2 = 0;
public bool heatingStartInduction2 = false;
public bool heatingStopInduction2 = false;
public bool heatingEndInduction2 = false;
public bool heatingCarrierRelease2 = false;
public float heatingTargetTime;
public float heatingCurrentTime;
public float heatingTargetTemp;
public float heatingCurrentTemp = 0;
public int heatingRFID2 = 0;
public int heatingConveyorSpeed2 = 0;
public bool codesys2StartInduction2 = false;
public bool codesys2StopInduction2 = false;
public bool codesys2ToRobotino2 = false;
public bool codesys2FromRobotino2 = false;
public bool codesys2CarrierRelease2 = false;
public bool codesys2EndInduction2 = false;
public int codesys2RFID2 = 0;

public bool magFrontConvRunning = true;
public bool manualConvRunning = true;
public bool camInspectConvRunning = true;
public bool codesys1ConvRunning = true;
```

Figure A.7: Global Variable List

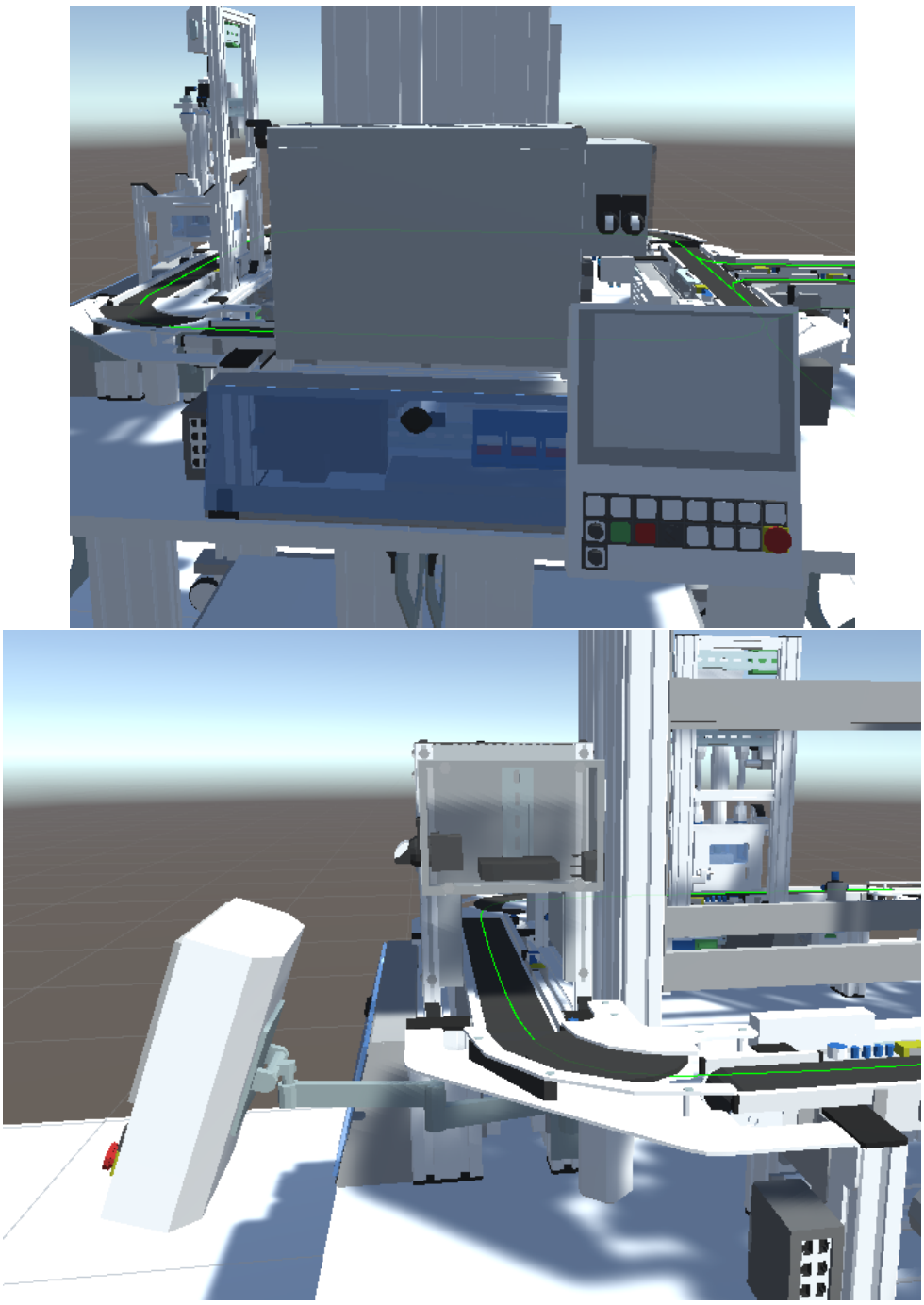


Figure A.8: Heating Station