University of South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2022

Industrial IT and Automation

Digital twin for monitoring, optimization and training in battery production

Ronnie André Horne Moe

Faculty of Technology, Natural sciences and Maritime Sciences

Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis, 2022

Title: Digital twin for monitoring, optimization and training in battery production

Number of pages: 66 + appendices

Keywords: Digital twin, Industry 4.0, Battery production, Virtual Reality

Student: Ronnie André Horne Moe

Supervisor: Ole Magnus Brastein

External partner: Norwegian battery manufacturer represented by Pascal Viala

Summary:

A Norwegian battery manufacturer with goals of becoming the most technological workplace globally, is working on developing 1D and 3D digital twin to model and optimize the battery production process.

The objective of the thesis is to model parts of the battery production process, visualize the model using a virtual reality solution, and present a networking solution to facilitate fast and easily configured communications.

The thesis presents a solution using discrete event simulations with SimPy in Python, a virtual reality environment developed in Unity and a communications platform using ØMQ.

The selected technologies are found to be capable of handling the tasks, but some limiting factors of the solution are identified. Remaining work to make the solution a realistic representation and fully functional 3D digital twin is discussed.

Preface

This thesis was written in the spring of 2022 for the course FMH606 Master's Thesis as a conclusion to the master program in Industrial IT and Automation at the University of South-Eastern Norway (USN). For the last four years I've been a part time online student while working full-time at Boliden.

The project of creating a solution combining 1D and 3D simulation of battery plant production originates from our external partner which is a large battery cell manufacturer in Norway. The initial project description was formulated by Pascal Viala, which has also been functioning as the external supervisor on the project.

My four years as a student at USN has been both challenging and rewarding. The workload of the studies next to a full-time job has been hard at times, but the reward of the work that has been put in has far outweighed the struggle. I would like to thank the teachers and the university for developing a great program with very interesting and useful topics and courses, and the amazing job that has been put into making it possible to be a remote student.

I would also like to thank my employer, Boliden for giving me the opportunity and for giving me much needed support and time to perform my studies.

I owe big thanks to my family who has supported me while spending the afternoons and late nights working on tasks, projects and preparing for exams.

Finally, I would like to thank my supervisors Ole Magnus Brastein from USN and Pascal Viala, for the excellent support and cooperation in working on this interesting project.

Tyssedal, 17.05.2022

Ronnie André Horne Moe

Contents

Nomenclature

1D	One dimensional
2D	Two dimensional
3D	Three dimensional
API	Application Programming Interface
AR	Augmented Reality
BIM	Building Information Modelling
C#	C Sharp, programming language
CAD	Computer-Aided Design
DES	Discrete Event Simulation
DT	Digital Twin
EUD	End-User Development
FPS	Frames Per Second
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HMD	Head-Mounted Display
HWID	Human Work Interaction Design
IDE	Integrated Development Environment
IMRaD	Introduction, Methods, Results and Discussion
IoT	Internet of Things
IIoT	Industrial Internet of Things
MQTT	Message Queuing Telemetry Transport
MR	Mixed Reality
OPC	Open Platform Communications
OPC UA	OPC Unified Architecture
PCA	Principle Component Analysis
PLM	Product Lifecycle Management
PUB	Publish
RAM	Random Access Memory
SQL	Structured Query Language
SSD	Solid State Drive
SUB	Subscribe
UML	Unified Modeling Language
VR	Virtual Reality
XR	eXtended Reality
ØMQ	Zero Message Queuing (also ZeroMQ, 0MQ, zmq)

Figures

Codeblocks

Tables

1 Introduction

In this chapter the background of the thesis is presented as well as the objective and scope of the project. Relevant other research and theory connected to the thesis topic is presented. At the end of the introduction the structure of the thesis text is explained briefly.

1.1 Background

A battery manufacturer in Norway, which intends to produce environmentally friendly batteries at a large scale to power the green shift, wishes to create a state-of-the-art technological advanced workplace. To reach their goal they wish to develop 1D and 3D digital twin to optimize all factors of the production process. The thesis project intends to be a study and evaluation of possible technologies that can be part of a battery plant digital twin.

1.2 Objective and scope

The objective of the thesis is to model different parts of the battery production process and visualize them in a 3D environment. The solution goal is to identify production bottlenecks and inefficiencies, optimize factory layout, and alternatively be used to train personnel. The solution should strive to be usable in a future full scale digital twin implementation, with reusable and reconfigurable solutions.

To reach this goal the thesis will give an overview of research on the different topics and on combining data-driven models and 3D representations in digital twins, virtual reality (VR) and other eXtended Reality (XR) solutions. The solution should target best practice methods and highlight benefits of building a digital twin solution. Research methods and models that can be used to represent material flow of a battery plant production line.

The next step will be to implement a dynamic material flow model including a select set of processes in the battery plant, in python or other suitable programming language. The solution should be ready for replacing part of the simulation with actual process information when the factory is realized.

A VR environment should be developed in the Unity game engine or alternative software. The VR solution should use actual Building Information Modeling (BIM) files and other 3D models.

The two solutions should be integrated using easy to configure network-based communication, to enable updating of the VR visualization and output from the VR simulation to the material flow model.

Finally, the combined solution should be evaluated based on performance, versability and usability and future possibilities or needed improvements

1.3 Other research and theory

1.3.1 Digital Twin

The concept of twinning is not new, maybe one of the best examples of great use of twins is the mirrored system created by NASA during the Apollo 13 mission [1]. The twin system enabled the engineers on the ground to replicate the problems that the astronauts encountered during their space travel when an oxygen tank exploded two days after launch. With the use of the twin on ground, a solution was found and tested and the crew in space managed to overcome the challenges they had and returned safely to earth.

In later years the term Digital Twin (DT) has grown popular both in academia and in the industry, but there has been a lot of different definitions and characterizations of the DT presented over the years some of which is missing the original intent or the benefits that the digital twin can provide [2]. The term was first presented by Michael Grieves in a lecture on Product life-cycle management (PLM) in 2003, where he presented the DT as a three part system; The real space with the physical object, the virtual space with a virtual object, and the link between the two spaces that enables data flow between them and the possibility of synchronization of the objects, as seen in Figure 1-1. While the DT concepts first gained traction in the aerospace and defense industry it has now gained a lot of attention in other industries and is an integral part of Industry 4.0 and Smart Manufacturing [1].

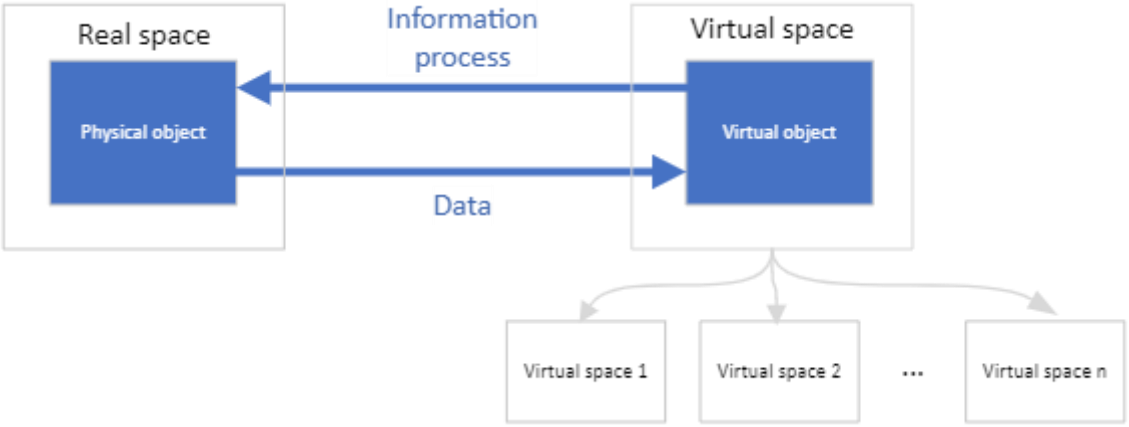

Figure 1-1: The Digital Twin as proposed by Grieves, as presented in [1]

In the survey done in [1] the authors went through 75 papers on the topic of DTs out of the 75, 31 include a definition of the DT concept, whereas 29 can be considered unique definitions, this spread in definitions signals that the term is still quite new and still evolving, but the definitions can be grouped based on some key factors that connect them, and papers with applications linked to manufacturing seem to favor definitions surrounding terms like “Virtual”, “Mirror” and “replica”. The other groups are centered on the terms “Integrated

system", "Clone, counterpart", "Ties, Links", "Description, construct, information" and "Simulation, test, prediction".

The paper "Characterising the digital twin: A systemic literature review" [2] lists 12 themes that characterize the DT namely physical- and virtual entity, physical- and virtual environment, fidelity, state, parameters, physical-to-virtual connection, virtual-to-physical connection, twinning and twinning rate, and physical- and virtual processes.

The physical entity is the real-world artefact that is the component, system or product that is the original twin, while the virtual entity is the physical entity's counterpart, the twin that exist in the virtual domain.

The physical environment is the real-world or real-space in which the entity "lives", the aspects or parameters of the physical environment is measurable and is fed into the virtual environment to ensure that the virtual counterpart mirrors the physical. The virtual environment is in literature sometimes referred to as the "database", "data-warehouse", "cloud-platform", "server" and "API", but linking the concept to tightly to the underlying technology might be considered an unwise approach in a fast-changing technological space.

The parameters of the DTs are the information that is passed between the physical and virtual spaces in the system and can be grouped into ten themes "Form", "Functionality", "Health", "Location", "Process", "Time", "State", "Performance", "Environment" and "Misc. Qualitative" as listed by Jones Et al. in [2].

The fidelity refers to the number of parameters that is shared between the physical and virtual spaces and their accuracy and level of abstraction.

The state refers to the current values of the parameters both measured values for the physical entity and environment and the virtual counterparts.

All the literature studied in Jones Et al.'s paper describes physical-to-virtual connections as a part of the DTs. The connection consists of a "Metrology phase" and a "Realization phase", whereas in the "Metrology phase" the physical state is measured and in the "Realization phase" the difference between the virtual and physical state is determined and the virtual is adjusted.

The virtual-to-physical connection is not always part of the description of DTs in literature even though it is included in Grieves' original works [2]. In the virtual-to-physical connection the running conditions of the physical twin can be altered by "solutions" found by the virtual twin to counter for example high temperatures or other unnormal conditions of the physical twin. The full potential of the DT relies on connections in both directions. Kritzinger et al. [3] proposes three levels of integration between the physical and virtual domains in digital twins as seen in Figure 1-2, whereas the "Digital Model" does not have an automatic flow of data between the two domains, the "Digital Shadow" in which data continuously flows from the physical to the virtual domain updating the state of the virtual to match the state of the physical counterpart, and finally the "Digital Twin" where data flows in both directions and the physical state is mirrored in the virtual, but also the other way, where a change of state in the virtual domain can be mirrored to the physical.

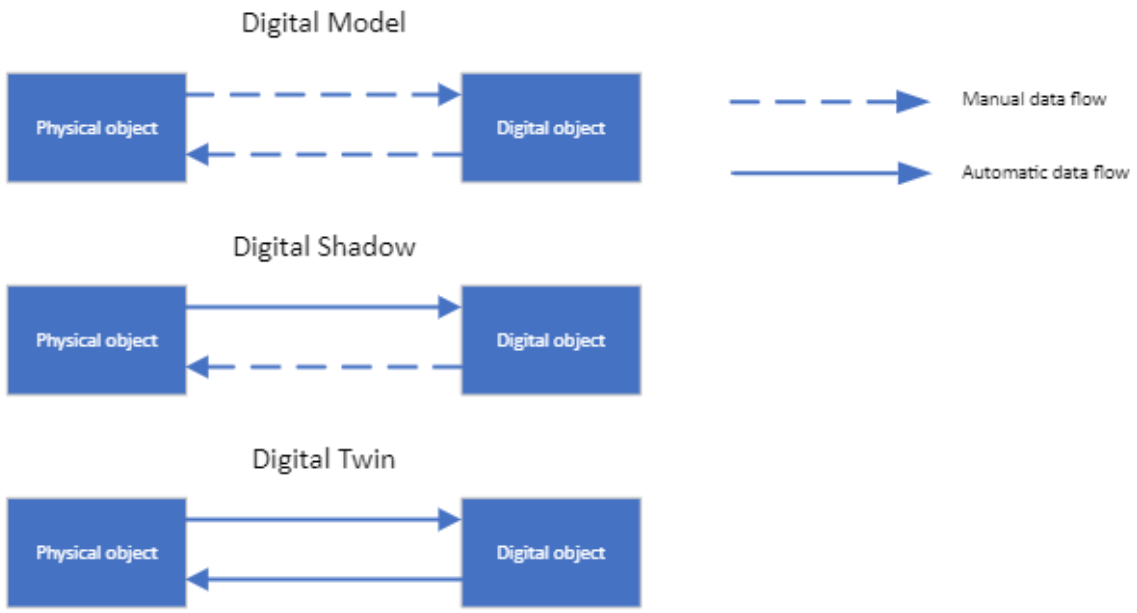

Figure 1-2: Kritzinger et al.'s three intergration levels [3]

The two lesser levels of integration can't be seen as full Digital Twin implementations, but for example in a concept phase of a new project these can be stages of integration before the physical object even exists or the digital objects state predictions can be trusted as accurate.

Next theme in Jones et al.'s paper is "twinning" and "twinning rate" which refers to the interval of which the virtual twin is updated or synchronized. A change in either the physical or virtual twin that results in a change in optimal parameters is automatically passed between the twins. An example presented by Jones et al is for example the automatic rescheduling of an assembly line production to counter a batch of faulty components.

Lastly is the theme of physical and virtual processes where the physical processes refer to the activities performed in the physical environment, which again results in changes in the physical measurable states that is communicated to the virtual twin. The virtual processes are the activities performed in the virtual environment with examples as simulation, optimization, diagnostics and prediction. These can result in changes in the virtual states and optimization recommendations that can be realized in the physical twin.

1.3.1.1 Benefits and Use-cases

Jones et al [2] lists several benefits highlighted in literature like reduction of costs, risk and design time, complexity and reconfiguration time as well as improving after-sales service, efficiency, maintenance decision making, security, safety and reliability, manufacturing management and improvement of processes and tools, enhancement of flexibility and competitiveness of manufacturing systems and the fostering of innovation that Grieves mentions in his papers. Broo et al [4] lists better asset and resource management, faster project delivery, organizational transformation, untapped resources utilization, bringing value

from data, lower cost through better information flows, optimizing operations and decreasing risk and increasing safety.

Even though the list of benefits is large and shows great potential for DTs, the literature has very few examples that validate these perceived benefits according to Jones et al.'s studies. This can be seen as a barrier to overcome, because of the costs and challenges posed by implementing DT strategies with higher demands on infrastructure, sensors, and changes to work-flows. Justifying the investment in DTs is difficult when the benefit of the DT is not clear, and the areas that can give higher returns is not easy to identify [2].

Use-cases for DTs are many, Jones et al. sites "Simulation, modelling and optimization", "Data driven design", "Data management", "Geometry Assurance", "Reconfiguration", "Health monitoring", "Learning" and more. Bécue et al. sites "Decision support for reconfiguration", "Predictive maintenance", "Supply chain optimization" and "Anomaly detection" [5].

1.3.1.2 Challenges

Modoni et al. [6] goes through several challenges when implementing DTs; Connection with the real factory, where high-speed transmissions from multiple sources might pose a challenge, demanding the use of new data communication technologies like OPC Unified Architecture (OPC:UA), Industrial Internet of Things (IIoT) sensors etc.; Granularity of the synchronization process, where you have to weigh the cost of processing high fidelity parameters and find the right compromise between the level of detail and processing efficiency; Management of the real-time and historical data, the DT system needs to be able to effectively harvest real-time data from the process which is dependent on the fidelity and complexity of the DT, in addition accumulation of historical data is needed for offline analysis which affects the volume of the data storage over time. Enabling technologies in this sphere is cloud-platforms/cluster systems with for example NoSQL databases. Important factors to consider is scalability, communication capabilities, possibilities to separate storage from data processing and management; Support for advanced simulation and forecasting tools is a key to enable simulation of factory performance and testing of different configurations. Modoni et al. mentions software tools for discrete or continuous simulators, multi-scale modeling technology, virtual reality, augmented reality and high performance computing as important technologies that enables the utilization of the DTs in this area; Data and Intelligence distribution is another challenge, when high fidelity of the DT can overrun the network with data, a solution to this can be distribution of intelligence also known as "Edge computing" where the processing of the data is moved closer to the process which enables reduction of the data moving over the network to central storage; Enhancing the interoperability of the production resources, for the DT concept to be effective the software systems involved in tackling the different aspects of the virtual domain needs to speak the same language, follow standards so that integration of the systems is possible. The creation of a data model and a well thought-through architecture of the data flow infrastructure in the system can help overcoming this challenge.

Barricelli et al. [1] lists Ethical issues that arise from data collections especially in healthcare and medicine applications, where privacy and anonymity is important factors to consider. The

challenge can also be applicable when including customer / supplier data into the development and simulations of the DT; Security and privacy is also a challenge when introducing IIoT and cloud computing into industrial applications. Extra attention needs to be put into securing the data and confidentiality, again these are very important in healthcare and medicine applications; Cost of development, the development and introduction of DTs can prove to be very costly due to reconfiguration, changed hardware and software needs. There is a lot of research and experiments being done in the area of DTs and as more results in the area are shared hopefully the validity of the benefits will be proven; Equally distributed wealth is another challenge, where Barricelli et al. states that DTs can widen the gap between the rich and the poor, because of the lack of underlying infrastructure and know-how to implement such systems in underdeveloped countries; Human Work Interaction Design (HWID) and End-User Development can be a challenge, where computer scientist neglect the design and development of the user interface of the DTs. The DTs needs to be accessible and usable by anyone in an effective way; Technical Limitations is the final challenge stated in Barricelli et al.'s paper, where the challenge of integrating the human factor in a DT system is emphasized as well as the limitation of fast and reliable data connections and collection of large amounts of data and effective interfaces for complex data visualization.

1.3.2 Discrete Event Simulation

Vargas proposes the definition "A Discrete Event System is a system where state changes (events) happen at discrete instances in time, and events take zero time to happen." [7] Additionally, Vargas states that between two events nothing, or nothing of interest happens, or no changes to the state occurs in the state between the events. Barrett et al. defines Discrete event simulation (DES) as "a method used to model real world systems that can be decomposed into a set of logically separate processes that autonomously progress through time" [8].

In the book "Use Cases of Discrete Event Simulation: Appliance and Research" [9] Bangsow has collected 16 different articles covering different use cases of Discrete Event Simulation. The cases come from a variation of industries with different simulation purposes. The articles cover topics like variance reduction, material flow simulation, virtual commissioning, layout planning, and optimization. Bangsow achieved great results in his own study when simulating an auto body shop using DES, with the simulation only deviating on average 2% from the real implementation. Voorhorst et al. has an article in the book where they investigate optimizing a highly flexible shoe production plant where they conclude that the simulations enabled them to get a full picture of the production dynamics.

1.3.3 eXtended Reality

Virtual Reality (VR), Augmented Reality (AR) and Mixed Reality (MR) has in recent years been referred to by the common term eXtended Reality (XR) [10]. VR refers to a technology where with the use of VR headsets the real world is replaced with a virtual world to the user, while AR refers to technology where the real world can be overlayed with additional information or new virtual elements. While AR is closer to the real world, and VR is entirely virtual, MR is in

the middle, and in some cases the virtual objects and real objects can affect one another in MR solutions. The AR and MR terms are in literature often mixed and not well defined, the AR term is still the most used, but the use of MR is on the rise [10]. These technology does not only refer to head-mounted displays (HMDs), but also refers to mobile implementations and other more traditional display technologies. Even though the first examples of head mounted VR technology was introduced in the 1960s, the popularity and maturity of these technologies has exploded in recent years. New technology is introduced at a high rate and computer hardware that can handle this is practically mainstream [11].

1.3.4 Combining Virtual Reality and simulation/digital twins

Oyekan et al. [12] claims there is an increasing need for layout optimization of factory plants to avoid sub-optimal designs that can negatively impact the flow of material and work processes, the authors also mentions the safety of employees as that needs to be considered, and it should be optimized before construction, to get it right from the start. Oyekan et al. performed an extensive literature review on the topic of combining DES and VR ranging from flight simulators in military training simulations, factory production lines, construction industry applications and more, and the benefits of combining them is higher rate of identifying modeling errors than what is possible with only DES. Another mentioned benefit is a better description of the situation is made possible with integrated solution, as well as better understanding of the subject area for both modeler and user than what is possible with 2D representations alone.

Oyekan et al. further discusses the lack of an established preference when it comes to communication between simulation and 3D or VR applications in the article. Harvard et al. [13] proposes a reusable communication architecture and the use of ZeroMQ (0MQ, zmq, ØMQ) socket-based machine-to-machine communication with an example of a digital twin solution for a workstation simulation with motion of a cobotic robot-arm. In Kuts et al.'s article on synchronization of digital twin and physical factory [14] a solution of a digital twin with VR capabilities is presented, which is synchronized using the Message Queuing Telemetry Transport (MQTT) publisher/subscriber protocol. The MQTT protocol is a standardized protocol which is lightweight, fast and often used in Internet of Things (IoT) solutions.

Xin presents a digital twin/VR solution for a Lithium Battery Pilot Production line [15] with some findings on design frameworks, communication data flows and benefits like feasibility confirmation of equipment and confirmation of safe working areas for staff in the workshop area through the use of DT and VR, but the conference protocol does not provide good descriptions of how to implement such solutions.

1.4 Thesis structure

The thesis is built up following the IMRaD-structure, with this chapter, the introduction to the topic, project objective and scope and relevant research. Following is the methods and materials chapter where the equipment and software used to set up and develop the solution

is presented as well as other relevant data sources. In the results chapters the developed solution is presented. The thesis is concluded by the discussion and conclusion chapters where the results are discussed, findings and possible future improvements are presented.

2 Materials and Methods

In this chapter the equipment, software and APIs used to develop the solution is presented, as well as other sources of data and input finally a short segment on the development process and research work.

2.1 Equipment

To facilitate the development and testing of the solution a powerful computer with AMD Ryzen 7 3700X 8-Core Processor at 4.05GHz, 16GB of RAM, Nvidia RTX 3070 GPU, 1.5TB SSD with Windows 11 Pro was used in addition to less powerful computers.

The virtual reality solution was developed and tested with the use of HTC Vive Pro HMD with HTC Vive Base stations and Vive Pro 2 Controllers with a room scale setup.

2.2 Software and technology choices

2.2.1 XR development engines

There are several choices when it comes to XR development today, like CryEngine, ApertusVR and Amazon's Sumerian, but two tools dominate in this field, Unity and Unreal Engine. [16] Both Unity and Unreal Engine are considered very capable and comparable in all aspects of XR development. Both tools started out as game development engines and has added XR features in the later years.

Both engines have free entry level versions or licenses for use by small private developers or students, and paid versions for enterprise.

According to Gajsek [16] Unity stood for 60% of AR/VR content created and 50% of mobile games as of February 2022. The engine can be used to develop for a total of 28 different platforms, while Unreal Engine supports 15 platforms.

Both engines have dedicated asset stores where it's possible to download free and paid solutions, 3D models/assets and scripts to make project development easier and faster. The Unreal Engine marketplace has around 10.000 assets available [16] while the Unity asset store is the biggest one with above 42.000 3D assets alone [17], and several other assets.

When it comes to documentation both engines offer great documentation on development using the engines in general, but also specific documentation directed at XR development. Regarding the availability of training courses aimed at development using the engines there are more courses available for Unity than Unreal [16]. A simple google search for "Unity xr development course" gives 934 000 results while "Unreal engine xr development course" gives 601 000 results.

With millions of users worldwide Unity and Unreal Engine both have large developer communities, with forums where thousands of threads discuss general engine problems and solutions and thousands of threads on specific XR development topics. [16]

The development process with the two different engines is a bit different, both have development interfaces comprising of scene editors, where it is possible to layout levels with 3D objects, organize the projects assets and assign features to “game objects”. Features and scripts in Unity are done using the C# programming language while in Unreal Engine the main programming language is C++, but Unreal Engine also offer the use of “Blueprints” which is a visual scripting interface where the user does not need to “code” in the traditional sense. [16]

Both engines are considered able to produce high quality content. Companies like Audi, INVISTA [16], Daimler, Honda and Lockheed [18] has been using Unity for extended reality projects. C4X Discovery, Air Canada [16], Precision OS, Toyota [19] and more uses Unreal Engine for extended reality. The use cases range from visualizing complex data, training, improvement of logistics, improving service, evaluation of ergonomics, architectural design, vehicle design and more.

In conclusion there is no wrong choice between Unity and Unreal Engine when it comes to XR development. In sum there is a slight advantage of choosing Unity over Unreal Engine when the developer is new to the field, as is the case in this thesis project. In addition, the project is done by a developer with some experience in using Unity and the C# programming language.

2.2.2 Simulation software

A choice was made to use Python as the programming language for the simulation part of the project, due to its scientific computing abilities, use in artificial intelligence and machine learning projects and its popularity in the field [20].

When it comes to the actual implementation of the simulator in Python, the use of conventual programming could be an option or the use of libraries to support the development.

When it comes to discrete event simulations as discussed in 1.3.2, the options in Python are limited, the SimPy framework is the most used option which is used in many simulation implementations, where some are well described like the simulation of queueing by Jain [21] and Mesquita et al.’s Skateboard factory case [22].

There are optional discrete event simulation frameworks for Python like Moddy [23], which has some features that are not present in SimPy, like the ability to output sequence diagrams and structure graphs from simulations, but lack other features e.g., real-time simulations.

Outside Python there are several options to model discrete event simulations, commercial products as SIMUL8, Rockwell Automation Arena, FlexSim, MathWorks SimEvents (for MATLAB/Simulink), Siemens Plant Simulation, WITNESS and open-Source alternatives as SIM.JS, Simula, PowerDEVS and more [24].

2.2.3 Communication platforms

In the literature review done for this thesis communication technology like MQTT, ØMQ and OPC UA are mentioned as possible technologies that can be used in digital twin implementations.

Øvern [25] lists pros of OPC UA as open and freely available, cross-platform with robust security, service-oriented architecture with a focus on data collection and control. OPC UA is a promising technology in machine-to-machine communications field and is mentioned as a possible foundation in the factory of the future and could play a big part in the move towards a future where every machine and sensor are online.

The available implementations of OPC UA are well described, in Unity there are limited options where the easiest would be to use paid assets to implement it.

MQTT is a messaging protocol based on the publish/subscribe model. The protocol is designed with low network and device resource usage in mind. The protocol is reliable and in some degree message delivery is assured. The protocol is considered ideal for machine-to-machine communication. The protocol relies on the use of brokers to handle the delivery of messages and several public brokers are available for free use [26]. The MQTT protocol can be easily implemented in both Python and Unity without paid assets, like Viganò's Unity implementations available on GitHub [27] and others.

ØMQ (also known as ZeroMQ, 0MQ or zmq) is an open-source universal messaging library available in many programming languages and runs on most operating systems, it supports several messaging patterns like publish/subscribe and push/pull, it is lightweight, fast and simple to implement, and it supports different transportation protocols like TCP, UDP, inproc and more [28].

For this thesis project the choice of communication platform was ØMQ because of the ease of implementation in both Python and Unity. Initial trials of implementing OPC UA in Unity without the use of paid assets was unsuccessful and abandoned at an early stage. MQTT would have been a valid option as well, but the additional communication models possible with ØMQ is considered a good possibility to have in the project.

2.3 Development software

The development has been performed using a set of tools suitable for the different areas of the solution. For the visualization of the solution the game engine framework Unity was selected, Unity game code is done using C# where Visual Studio Community 2019 were used. The simulation and digital twin prototype were programmed in python using the Visual Studio Code IDE. Preparation of 3D models was done using FreeCAD and Blender.

2.3.1 Unity

On Unity's webpage they state that "Unity is the world's leading real-time 3D development platform" [29]. Unity is primarily a game engine and framework for making games on

platforms ranging from mobile devices to Microsoft Windows PC, Apple Mac, gaming consoles and Microsoft HoloLens. Unity has also gained a lot of interest from industry and research and has been used in visualizing projects both in standard screen solutions and XR implementations. Unity has met this with the development of new products like Unity Industrial Collection and Unity Reflect which is directly targeting industrial customers [30] [31].

2.3.2 Microsoft Visual Studio Community 2019

To develop programs and features for the Unity solution Visual Studio is used. The software provides excellent integration with Unity, enabling debugging to identify bugs and evaluation of the code created. In addition, the IDE offers intelliSense to be more efficient in code writing, and more [32].

2.3.3 C#

C# is an object-oriented programming language developed and maintained by Microsoft and the programming language used to program game code for Unity. The C# code uses the Unity API, and most of the code created inherits the MonoBehavior class which enables the code to be added to game objects in Unity. The objects can then be updated using the Unity game loop.

2.3.4 Microsoft Visual Studio Code

The Microsoft Visual Studio Code IDE is a free alternative which is great for programming in a lot of different programming languages, but does not offer the same integration with Unity as Visual Studio Community/Pro/Enterprise. The IDE is lightweight and can be set up with extensions suiting the needs of the user. In this project the IDE has been used to program and debug the python code.

2.3.5 Python

Python is general-purpose programming language first worked on in the late 1980s and released in 1991. The designer of the language is Guido van Rossum [20]. Python is currently developed by “Python Software Foundation” and the latest stable release is 3.10.4 (24th of March 2022) Python is the most, or at least one of the most popular programming languages today. Python is very popular in scientific computing, and in projects with artificial intelligence and machine learning projects. In this project Python version 3.8.7 has been used.

2.3.6 Blender

Blender is a free 3D modelling software which can import and export several 3D-model formats. Blender is developed by the Blender Foundation. In this project version 2.93.0 was used and the software has been used to convert some 3D-model formats and to model some simple models for the VR application.

2.3.7 FreeCAD

FreeCAD is an open-source 3D-modeler used in product design, mechanical engineering and architecture [33]. In this project the software version used is 0.19.3 and it has been mainly used to convert some already made CAD 3D models to formats that can be handled in Blender.

2.4 APIs and tools

Several application programming interfaces (APIs) and tools has been used in this project, most connected to the Python software implementations, but also some additional for the Unity implementation.

2.4.1 Python APIs and tools

The most important python APIs used in the project is SimPy and pyzmq, but several others have been used.

2.4.1.1 SimPy

SimPy is a discrete event simulation framework for python. Using this it is possible to model processes and active components. SimPy enables the user to model the use of shared resources in a model. The SimPy simulations can be run “as fast as possible” or in real-time using different environments [34].

2.4.1.2 Matplotlib

Matplotlib is a plotting library for Python, that can provide the user with different types of plots that can be embedded in Graphical User Interfaces (GUIs) or printed to image file formats [35].

2.4.1.3 pandas

Is a fast and powerful data analysis and manipulation tool for use with Python. It is excellent tool when working with data sets, observational or statistical. The data can be labeled or not [36].

2.4.1.4 NumPy

NumPy is the python tool for scientific computing. Matplotlib/pandas rely on NumPy, as does machine learning libraries like scikit-learn, SciPy and TensorFlow and more [37].

2.4.1.5 tkinter

“The tkinter package (“Tk interface”) is the standard Python interface to the Tcl/Tk GUI toolkit” [38]. The interface is used to create a GUI for python programs, in this project it has been used to create a simple GUI to present simulation status, plots and some control features.

2.4.1.6 PyZMQ

PyZMQ is the Python implementation of the ØMQ API. [39] The ØMQ (or zmq, 0MQ, ZeroMQ) API is, as mentioned in 1.3.4 a fast and lightweight machine-to-machine communication API, which support different communication patterns, like request-reply, push-pull and publish-subscribe, the API uses sockets for network programming, the input-output model is asynchronous [28]. The zero in the name originally referred to “zero broker” and “zero latency” in contrast to MQTT protocol which is broker dependent.

2.4.2 Unity APIs and packages

The Unity VR implementation relies on the SteamVR Unity Plugin and the NetMQ API

2.4.2.1 SteamVR Unity Plugin

The SteamVR Plugin is developed and maintained by Valve and is a multi-VR-platform API that enables developers to easily program VR-applications for different headsets like HTC Vive, Meta Quest, or Windows Mixed Reality headsets [40]. The main purpose of the API is to load 3D models of VR controllers, handle input and estimating the look of the hand holding the controller, but in addition the plugin provides an extensive example of interaction systems that enables developers to quickly prototype VR-applications with interactions with the virtual world of the solution. In the thesis project the plugin has been used to set up the VR-camera, hand interaction with virtual world menus and teleportation of the user around the virtual environment.

2.4.2.2 NetMQ

NetMQ is the C# counterpart to PyZMQ, which enables ØMQ communication in C# programs and Unity. The library is a 100% native port of the ØMQ library [41]. To get the library working smoothly in Unity the guide and example codes of Nicola S. [42] can be of great help.

2.5 Other sources of data

The external partner in the project has provided important input on the process of battery manufacturing [43], an initial small scale process model implemented in SimPy and provided some CAD 3D-models that has been used in VR-solution.

2.6 Development process

The project was developed by creating smaller sample projects before expanding the solutions and implementing more and more features into the project. New features or elements of the project was tested with smaller programs developed for the purpose.

To test the ability of the simulator-solution different setups and number of running processes will be tested. The final VR solution will be analyzed using the profiler tool of Unity and additional tests.

3 Results

In this chapter the developed solution is present as well as results from performed testing. Firstly, the Battey Production process studied in this project is presented with focus on different stages of the production process, and the flow of raw materials and products in the process. Then the developed simulation, monitoring and virtual reality solutions and results are presented.

3.1 Battery Production process

In this segment an overview of the battery production process is given with focus on machines/processes and materials used and products coming from the different processes. The processes in 3.1.1 to 3.1.5 is included in the developed simulation, while 3.1.6 and 3.1.7 was not.

A process area numbering system has been used in the figures where numbers 001-099 refer to raw materials, 100-199 are processes/machines and products from the Anode/Cathode powder processing, 200-299 from Anode/Cathode slurry processing and brick formation, 300-399 from Foil cut and lamination stage, 400-499 Electrode casting and unit assembly, 500-599 Pouch assembly, 600-699 Formation and aging and 700-799 range is from Inspection and packing.

The process description is not a complete overview, some of the stages are simplified. The basis of the process description are presentations from the external partner in the project [43].

3.1.1 Anode/Cathode powder processing

In the powder processing steps the anode and cathode powder blends are prepared. The anode blend is milled, while the cathode blend is also dried in an additional step as seen in Figure 3-1.

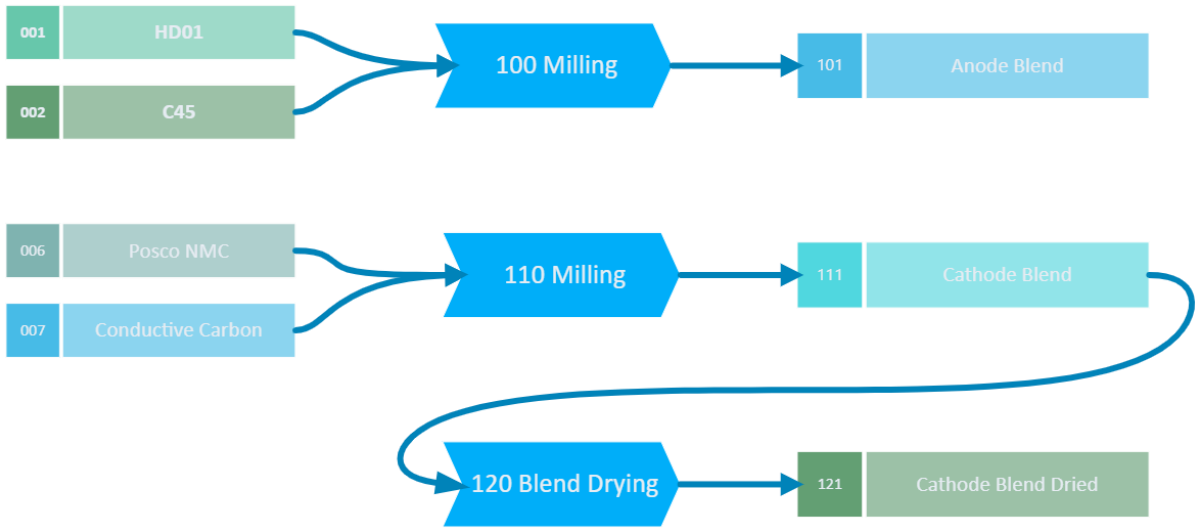

Figure 3-1: Powder processing materials, processes and products

3.1.2 Anode/Cathode slurry processing and brick formation

In Figure 3-2 the Slurry processing and brick formation stage is presented, where anode blend and dried cathode blend is mixed with electrolyte to create slurry that is compressed and loaded into cartridges.

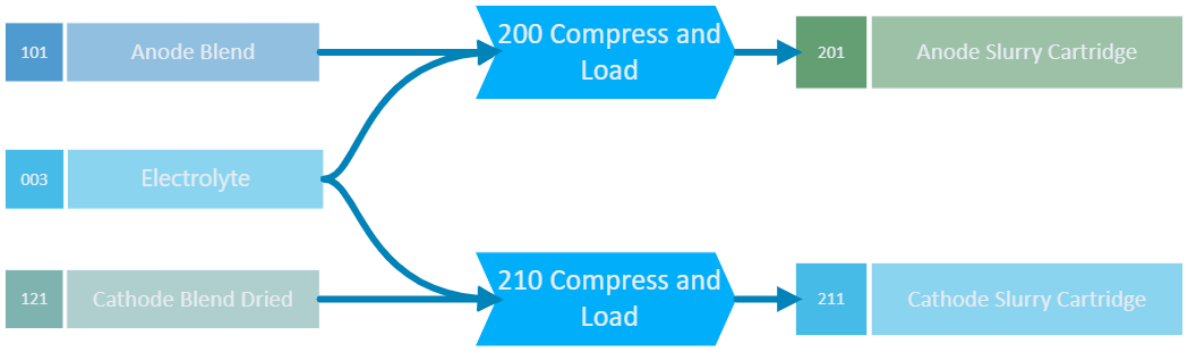

Figure 3-2: Slurry processing and brick formation, materials and products

3.1.3 Foil cut and lamination

In the Foil cut and lamination stage anode and cathode laminated foils are prepared as well as the separator as seen in Figure 3-3.

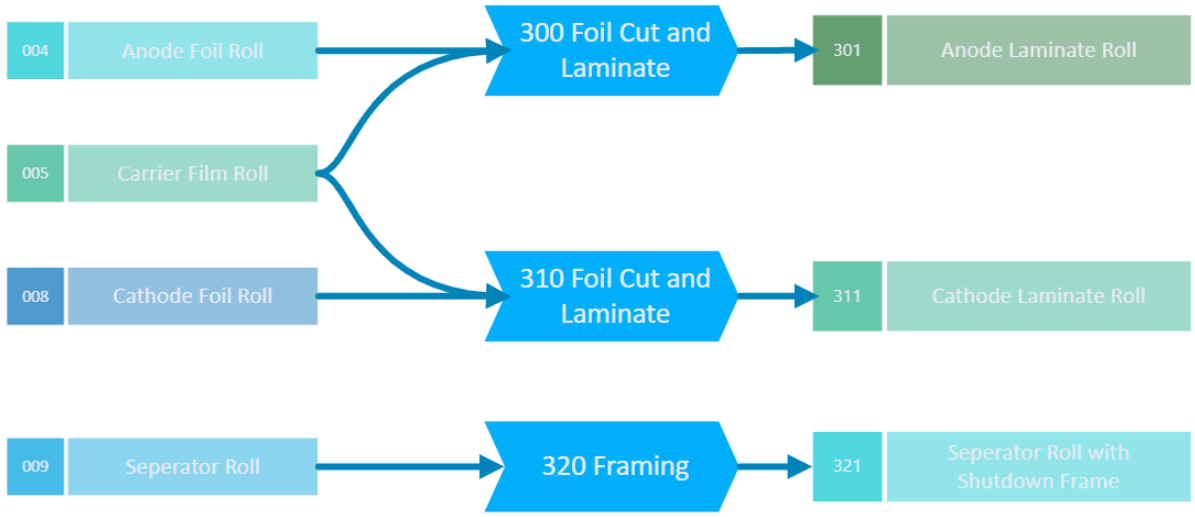

Figure 3-3: Foil cut and lamination, materials, process steps and products

3.1.4 Electrode casting and unit cell assembly

In Figure 3-4 the Electrode casting and unit cell assembly is described. These processes is handled by one machine, but in the implementation of the simulation it is treated as separate processes with single buffers between them for the electrode products. The final product from this stage is the Unit cell. In the assembly stage of the process a quality check is performed, if the product is faulty the product is sent to scrap storage.

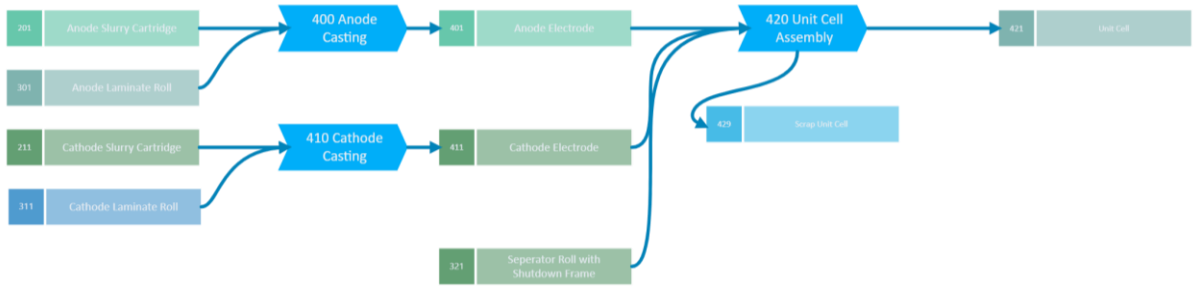

Figure 3-4: Electrode casting and unit cell assembly, materials, process steps and products

3.1.5 Pouch assembly

The Pouch assembly stage is described in Figure 3-5. In this stage the Unit cells are stacked, compressed and tabs are welded, and the completed cell is vacuum sealed in formed pouches.

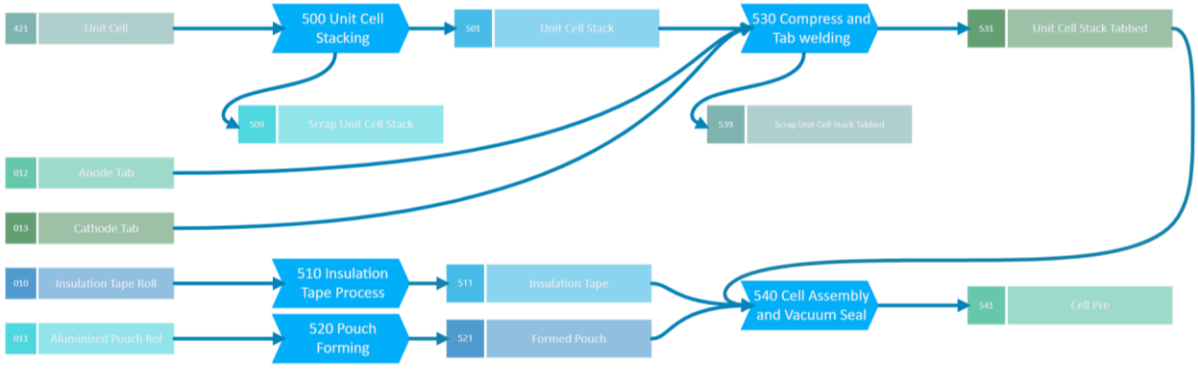

Figure 3-5: Pouch assembly, materials, process steps and products

3.1.6 Formation and aging

In the formation and aging stage, the cells go through formation and discharge, degassing and aging and testing. The process is simplified as shown in Figure 3-6.

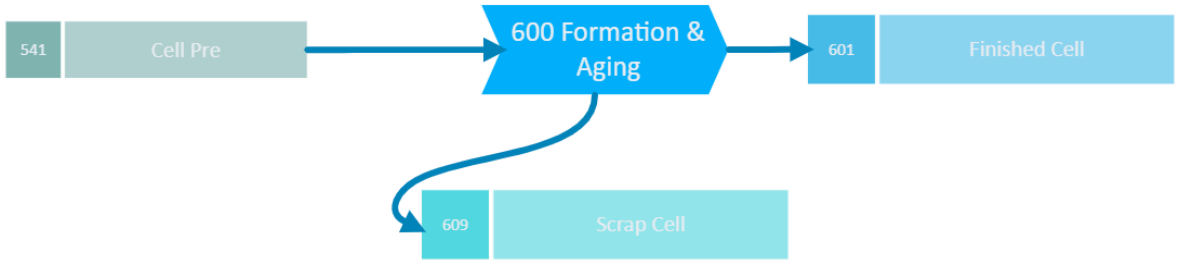

Figure 3-6: Formation and aging, materials process steps and products

3.1.7 Inspection and packing

Finally, the finished and approved cells are packed in boxes and loaded onto pallets on which they are stored until they leave the warehouse to the customer as can be seen in Figure 3-7.

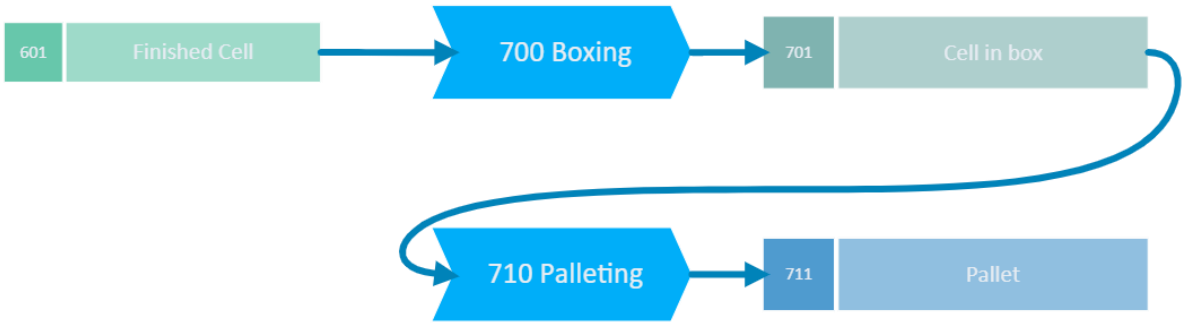

Figure 3-7: Inspection and packing, materials, process steps and products

3.2 Software topology and communication

The software components or applications developed in the solution were set up with the simulation also functioning as the central server of the communications as described in Figure 3-8. The communication is handled using ØMQ socket communication with the Publish/Subscribe model.

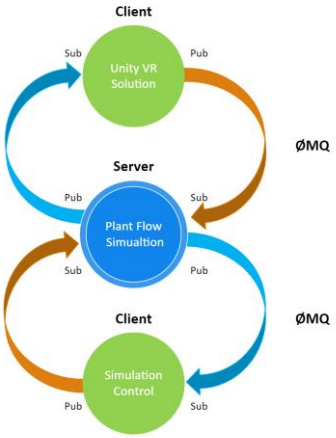

Figure 3-8: Software topology and communications

An optional topology considered for the project is presented in Figure 3-9. Where a broker or router functions as the server centrally in the topology.

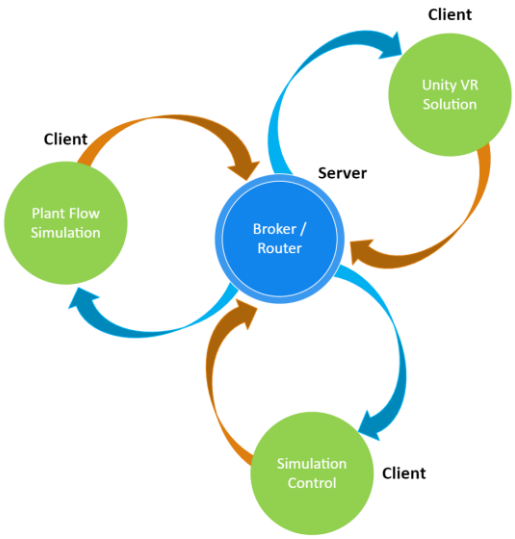

Figure 3-9: Optional topology

To simplify the developed solution the topology in Figure 3-8 were favored. In a full-scale implementation, the optional topology would probably be preferred to keep the functions more separated. If the ØMQ communications were to be replaced with MQTT communications, the topology would be required.

3.3 Simulation program

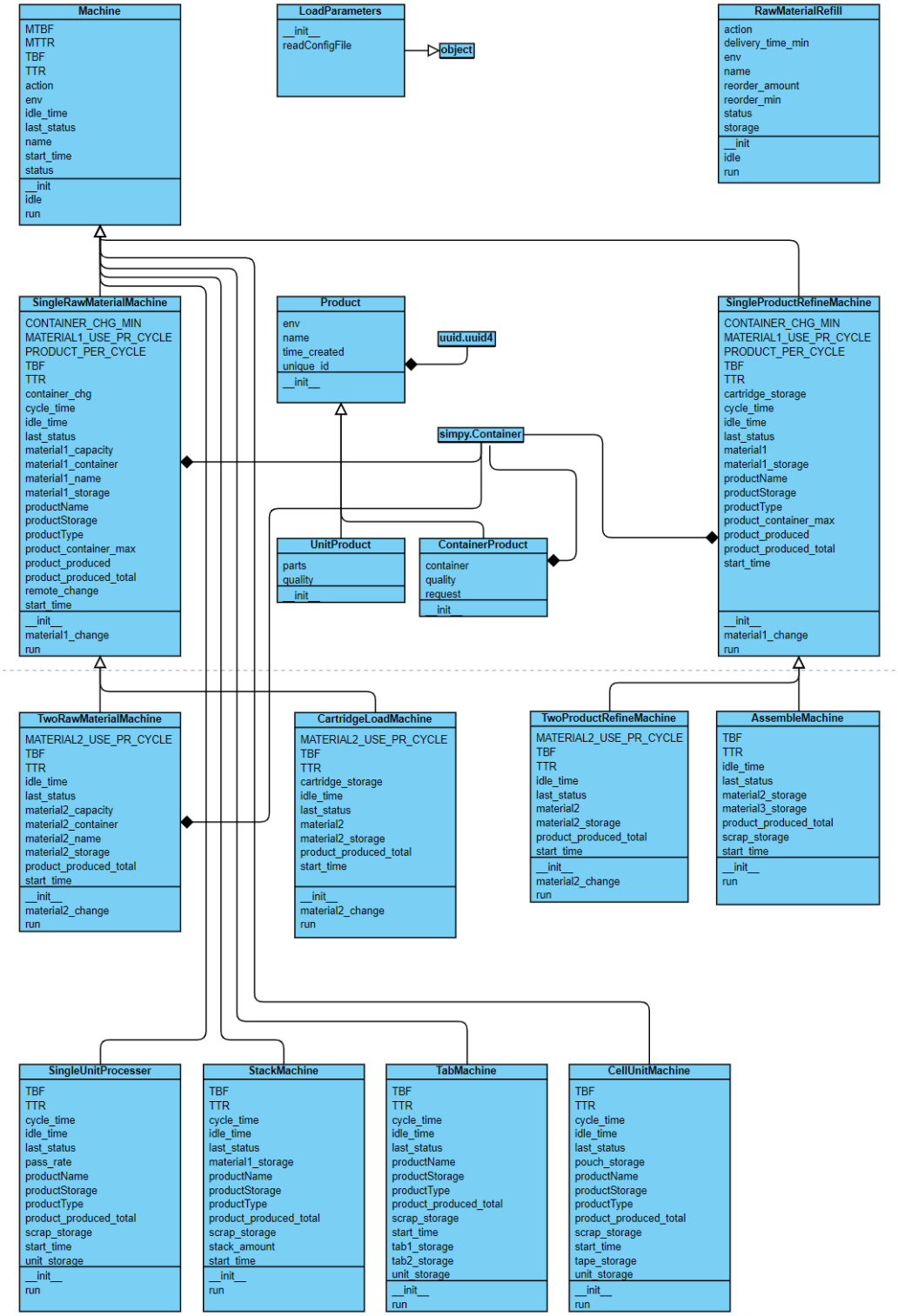

Figure 3-10: UML Class diagram for Python script PlantFlowSim.py

In Figure 3-10 an UML class diagram of the developed simulator solution is presented, showing the developed classes and subclasses. Each class in the diagram is presented with its attributes and operations as well as inheritance.

In the UML diagram the structure of the classes is shown, where the target was to reuse code by means of inheritance. As can be seen the Machine class is inherited by several machine type classes, and some machine classes has more levels of inheritance. The same has been done for the product classes, where common features are made available in the top-level class.

The program uses the SimPy API for modeling of the discrete event simulation of the production process, with a real-time environment to output events in real-time instead of a as fast as possible manner. For communications the ØMQ API Python implementation is used. In Code 3-1 the program imports are shown and in Code 3-2 the real-time environment is initialized.

```
import simpy
import simpy.rt
import threading
import zmq
import json
import uuid
from numpy import random

from zmq.eventloop import ioloop
```

Code 3-1: Python program imports of the Simulation program

```
# initialize simulation environment
env = simpy.rt.RealtimeEnvironment(factor=1, strict=False)
```

Code 3-2: The initialization of the real-time environment of SimPy

The main function of the program seen in Code 3-3, starts by loading simulation parameters from file, runs the function to set up storages, machines and reorder processes, then starts the communication subscription function on a separate thread before the runsimulation() function seen in Code 3-4.

```
def main():
    global Parameters
    Parameters = LoadParameters('SimParameters.cfg')
    SetupStorages()
    SetupMachines()
    SetupReorderProcesses()
    reciever = threading.Thread(target=sub_handler)
    reciever.start()
    runsimulation()
    reciever.join()

if __name__ == "__main__":
    main()
```

Code 3-3: The main initialization of the program

```
def runsimulation():
    global exitflag, start
    endproc = env.process(end_process(env))
    while True:
        if start:
```

```
            env.run(until=endproc)
            exitflag = False
            start = False
```

Code 3-4: RunSimulation function

As can be seen in Code 3-4 the simulation is not started immediately because the start variable is initialized as False. When the simulation is started it will run until the end process is completed, which is done when the exitflag variable is set to True.

In Code 3-5 the LoadParameters class is seen, which is used to load all simulation parameters from file.

```
class LoadParameters(object):
    def __init__(self, config_file):
        self.readConfigFile(config_file)

    def readConfigFile(self, file):
        try:
            with open(file, 'r') as cfg:
                lines = list(cfg)
            print("Read", len(lines), "lines from file", file)
            for line in lines:
                if not line == "\n" and not line.startswith("#"):
                    key, value = line.split('=')
                    setattr(self, key.strip(), eval(value.strip()))
        except:
            print('Configuration file read error')
            raise
```

Code 3-5: Class used for loading of parameters from text file

3.3.1 ØMQ setup and use

The ØMQ communication is very easily set up. In Code 3-6 the publishing and subscribing sockets are initialized and bound to ports. The subscription socket is set to subscribe to topics “C”, this means that the socket will receive every topic starting with “C” e.g., “COMMAND” and “CHANGE” which is used in this project.

```
context = zmq.Context()
socketPub = context.socket(zmq.PUB)  # <- the PUBLISH socket
socketSub = context.socket(zmq.SUB)  # <- the SUBSCRIBE socket
socketSub.setsockopt_string(zmq.SUBSCRIBE, "C") # <- topic subscription

socketSub.bind("tcp://*:5556")
socketPub.bind('tcp://*:5555')
```

Code 3-6: Setting up the zmq context and sockets for pub/sub communications

The code in Code 3-7 encodes topic and json object for outgoing messages and decodes topic and json object from incomming messages. The functions are attributed to Mike Ellis [44]

```
def mogrify(topic, msg):
    """ json encode the message and prepend the topic """
    return topic + ' ' + json.dumps(msg)

def demogrify(topicmsg):
    """ Inverse of mogrify() """
    json0 = topicmsg.find('{')
    topic = topicmsg[0:json0].strip()
    msg = json.loads(topicmsg[json0:])
```

```
    return topic, msg
```

Code 3-7: Functions to encode topic and json of outgoing and decode topic and json from incoming

In Code 3-8 the sub_handler() function which is run on a separate thread is seen, in this the socket is checked for new messages which is then processed by the getcommand function in Code 3-9.

```
def sub_handler():
    while True:
        topic, msg = demogrify(socketSub.recv_string())
        getcommand(topic, msg)
```

Code 3-8: Loop handling incomming messages

```
def getcommand(topic, msg):
    global exitflag, start
    print("Received: {topic} {message}".format(topic=topic, message=msg))
    if topic == "COMMAND":
        if msg['command'] == "Exit":
            print("Received exit command, client will stop receiving messages")
            exitflag = True
        elif msg['command'] == "Start":
            print("Received Start command, Simulation starts")
            start = True
    elif topic == "CHANGE":
        name = msg['name']
        if any(elem.name == name for elem in machines):
            machine = next(machine for machine in machines if machine.name == name)
            try:
                machine.container_chg = True
            except:
                print('Container change failed')
    else:
        print("Unknown topic recieved")
```

Code 3-9: Function handling incoming commands and change orders

In Code 3-10 the functions for sending status and storage updates to the publishing socket is seen. The data sent is json formatted.

```
def update_status(self, new_status):
    self.status = new_status
    if self.last_status != self.status:
        print(self.name, 'STATUS:', self.status)
        data = {
            'name': self.name,
            'status': self.status,
            'time': self.env.now

        }
        socketPub.send_string(mogrify("STATUS", data))

def update_storage(from_to, storage_name, level, direction):
    if direction == "from":
        print(from_to, " <- ", storage_name, ": ", level)
    else:
        print(from_to, " -> ", storage_name, ": ", level)
    data = {
        'from_to': from_to,
        'storage': storage_name,
        'level': level,
        'direction': direction,
        'time': env.now
    }
    socketPub.send_string(mogrify("STORAGE", data))
```

Code 3-10: Functions to send updates on machine status and updates on buffer storages

3.3.2 Product and machine classes

In the SimPy implementation the simulated machines can output product objects which is defined as shown in Code 3-11. The main product types are the UnitProduct and the ContainerProduct, where the container product is a product containing variable amount of material.

```
class Product:
    def __init__(self, env, name):
        self.env = env
        self.unique_id = uuid.uuid4()
        self.time_created = env.now
        self.name = name

class ContainerProduct(Product):
    def __init__(self, env, name, init_level, request=None, quality="PASS"):
        super().__init__(env, name)
        self.container = simpy.Container(
            env, init_level, init=init_level)
        self.request = request
        self.quality = quality

class UnitProduct(Product):
    def __init__(self, env, name, parts=[], calculate_quality=False, quality="PASS", passRate=1.0):
        super().__init__(env, name)
        self.parts = []
        self.quality = quality
        for part in parts:
            self.parts.append((part.name, part.unique_id, part.quality))
            if(part.quality == "FAIL"):
                self.quality == "FAIL"
        if self.quality == "PASS" and calculate_quality and not random.binomial(n=1, p=passRate,
size=1):
            self.quality = "FAIL"
```

Code 3-11: Product class and subclasses created by the different machines in the simulation

In Code 3-12 the parent machine class is shown, all the machines inherits this class directly or indirectly.

```
class Machine:
    def __init__(self, env, name, MTBF, MTTR):
        self.env = env
        self.name = name
        self.action = env.process(self.run())
        self.MTBF = MTBF
        self.MTTR = MTTR
        self.TBF = random.normal(MTBF, MTBF/4)
        self.TTR = random.normal(MTTR, MTTR/2)
        self.start_time = env.now
        self.idle_time = 0
        self.status = "IDLE"
        self.last_status = "IDLE"

    def run(self):
        pass

    def idle(self):
        update_status(self, "IDLE")
        self.idle_time += Parameters.SIM_STEP
        yield env.timeout(Parameters.SIM_STEP)
```

Code 3-12: The Machine class

```
class SingleRawMaterialMachine(Machine):
    def __init__(self, env, productType, productName, productStorage, PRODUCT_PER_CYCLE, cycle_time,
product_container_max, material1_name, MATERIAL1_USE_PR_CYCLE, material1_capacity, material1_storage,
CONTAINER_CHG_MIN, MTBF, MTTR, name="ROLLER", remote_change=False):
        super().__init__(env, name, MTBF, MTTR)
        self.productType = productType
        self.productName = productName
        self.productStorage = productStorage
        self.material1_name = material1_name
        self.material1_capacity = material1_capacity
        self.material1_container = simpy.Container(
            env, capacity=material1_capacity+MATERIAL1_USE_PR_CYCLE, init=0)
        self.material1_storage = material1_storage
        self.MATERIAL1_USE_PR_CYCLE = MATERIAL1_USE_PR_CYCLE
        self.PRODUCT_PER_CYCLE = PRODUCT_PER_CYCLE
        self.CONTAINER_CHG_MIN = CONTAINER_CHG_MIN
        self.remote_change = remote_change
        self.container_chg = False
        self.product_produced = simpy.Container(env, init=0)
        self.product_produced_total = 0
        self.product_container_max = product_container_max
        self.cycle_time = cycle_time

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif self.material1_container.level >= self.MATERIAL1_USE_PR_CYCLE and
(self.productStorage.capacity - len(self.productStorage.items) > 0):
                update_status(self, "RUN")
                self.material1_container.get(self.MATERIAL1_USE_PR_CYCLE)
                yield env.timeout(self.cycle_time)
                self.product_produced.put(self.PRODUCT_PER_CYCLE)
                self.product_produced_total = self.product_produced_total + self.PRODUCT_PER_CYCLE
                if self.product_produced.level >= self.product_container_max:
                    self.product_produced.get(self.product_container_max)
                    if(self.productType == ContainerProduct):
                        product = self.productType(
                            self.env, self.productName, self.product_container_max)
                    else:
                        product = self.productType(
                            self.env, self.productName)
                    yield self.productStorage.put(product)
                    update_storage(self.name, product.name, len(
                        self.productStorage.items), "to")
            elif self.material1_container.level < self.MATERIAL1_USE_PR_CYCLE and
self.material1_storage.level > 0:
                yield self.env.process(self.material1_change())
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

    def material1_change(self):
        update_status(self, "{material_name}_CONTAINER_CHANGE".format(
            material_name=self.material1_name))
        yield self.material1_storage.get(1)
        update_storage(self.name, self.material1_name,
                       self.material1_storage.level, "from")
```

```
        if self.remote_change:
            while not self.container_chg:
                yield env.timeout(1)
            self.container_chg = False
            yield self.material1_container.put(self.material1_capacity)
        else:
            yield env.timeout(self.CONTAINER_CHG_MIN)
            yield self.material1_container.put(self.material1_capacity)
```

Code 3-13: The SingleRawMaterialMachine class

In Code 3-13 a machine type is shown which can have the statuses “RUN”, “IDLE”, “DOWN” or “_CONTAINER_CHANGE” depending on the availability of raw material, how long it has been running and if there is room for more products in storage/buffer.

In Code 3-14 the RawMaterialRefill class is shown which takes care of refilling storages with raw material if the amount in storage is under a limit, considering the delivery time from the suppliers.

```
class RawMaterialRefill:
    def __init__(self, env, storage, reorder_amount, reorder_min, delivery_time_min, name="STUFF"):
        self.env = env
        self.action = env.process(self.run())
        self.storage = storage
        self.name = name
        self.reorder_amount = reorder_amount
        self.reorder_min = reorder_min
        self.delivery_time_min = delivery_time_min
        self.status = "IDLE"

    def run(self):
        while True:
            # Downtime
            if isinstance(self.storage, simpy.Container) and self.storage.level <= self.reorder_min:
                self.status = "REFILL ORDERED"
                print(self.name, 'ORDERED')
                yield env.timeout(self.delivery_time_min)
                self.status = "ORDER ARRIVED"
                print(self.name, 'ARRIVED')
                yield self.storage.put(self.reorder_amount)
                update_storage("Supply", self.name, self.storage.level, "to")
            else:
                yield self.env.process(self.idle())

    def idle(self):
        self.status = "IDLE"
        yield env.timeout(10*Parameters.SIM_STEP)
```

Code 3-14: RawMaterialRefill class

In Code 3-15 and Code 3-16 a storage container of raw material is initialized, and a machine set up and initialized.

```
    cathode_foil_roll_storage = simpy.Container(
        env, capacity=Parameters.CAFOILROLL_CAP, init=Parameters.CAFOILROLL_INIT)
    storages.append(cathode_foil_roll_storage)
```

Code 3-15: Initialization of a raw material storage, SimPy Container

```
    unit_cell_assembler = AssembleMachine(env, UnitProduct, "UNCE", unit_cell_storage,
        scrap_cell_storage, 1, Parameters.ASSEMBLE_CYCLE, 1, 1, seperator_roll_shutdown_frame_storage,
        anode_storage, cathode_storage, Parameters.CONTAINER_CHG, Parameters.MTBF_ASSEMBLE,
        Parameters.MTTR_ASSEMBLE, "UNCEAS")
    machines.append(unit_cell_assembler)
```

Code 3-16: Initialization of a machine class

3.3.3 Testing of the simulator

The simulation solution has been tested with a varying number of initialized machines to put it to the test. With the current timing parameters set up the maximum number of machines the solution can handle is 19 before struggling to keep up with the real-time event processing (Table 3-1) This were checked by setting the strict parameter set to True when initializing the real-time environment of SimPy. This way if the simulation can't keep up with the events processing it throws an exception and stops the program.

Considering that the timing parameters used in this simulation is highly sped up in comparison to the actual processing times of the machines the solution should be able to handle a lot higher number of machines and processes than what is needed.

Table 3-1: Test results of varying number of machines

Test #	*Number of machines*	*Number of Reorder processes*	*Result*
1	16	13	Pass
2	17	13	Pass
3	18	13	Pass
4	21	13	Not passed
5	19	13	Pass
6	20	13	Not passed

3.4 Simulation Control Application

To control the simulation solution and view the data messages coming from the simulation a 2D application was developed. The UML class diagram of the software is shown in Figure 3-11. The software is simplistic, with a few frames for controlling, checking status, view logs and graphs.

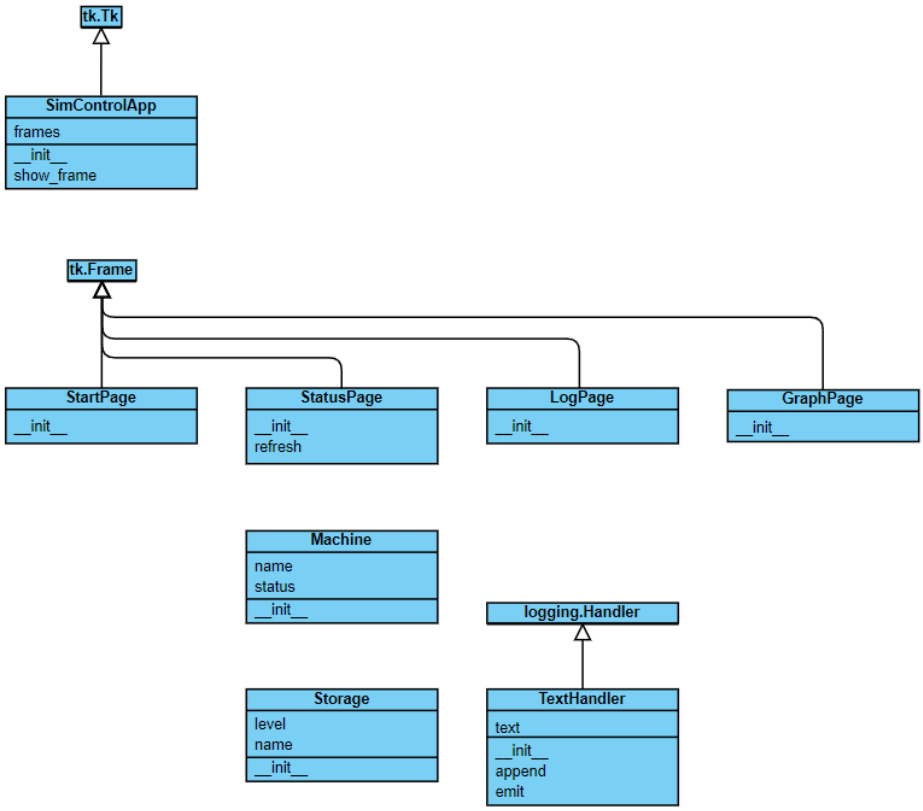

Figure 3-11: UML Class diagram for Python script SimControlAndGraph.py

The software uses tkinter for the GUI handling, matplotlib for ploting of values, and PyZMQ for communication. In Code 3-17 the imports of the Python program are shown.

```
from tkinter import ttk
import tkinter as tk
import tkinter.scrolledtext as ScrolledText
from matplotlib import style
import matplotlib.animation as animation
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
import logging
import zmq
import json
import threading
import pandas as pd
import matplotlib
```

Code 3-17: The Python program imports of the Simulation Control Application

The application is initialized as a TkApp with four frames as seen in Code 3-18.

3.4.1 GUI setup and updating

```
class SimControlApp(tk.Tk):
    def __init__(self, *args, **kwargs):
```

```
        tk.Tk.__init__(self, *args, **kwargs)

        tk.Tk.wm_title(self, "Battery Plant Simulator")

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        for F in (StartPage, StatusPage, GraphPage, LogPage):

            frame = F(container, self)

            self.frames[F] = frame

            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(StartPage)

    def show_frame(self, cont):

        frame = self.frames[cont]
        frame.tkraise()
```

Code 3-18: SimControlApp class

The four frames or views of the application can be seen in Figure 3-12, Figure 3-13, Figure 3-14 and Figure 3-15. The first view is the Controls where the ØMQ listener or subscription socket can be started, the simulator can be started and stopped and the socket subscriptions can be changed.

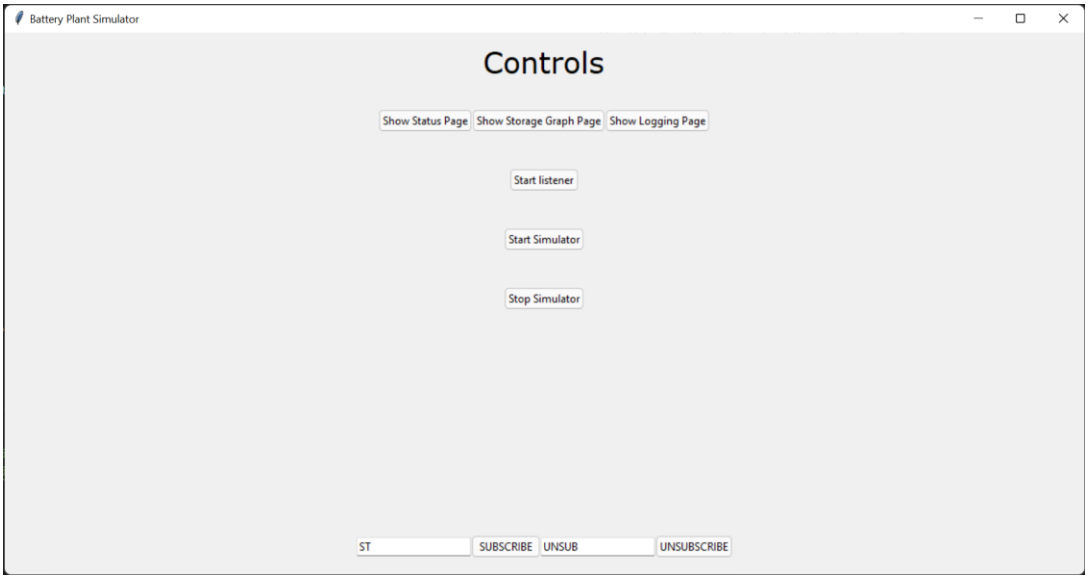

Figure 3-12: GUI of Simulator Control Application (Controls page)

In the status view in Figure 3-13, the current status of the machines can be seen, as well as current storage levels in the simulation.

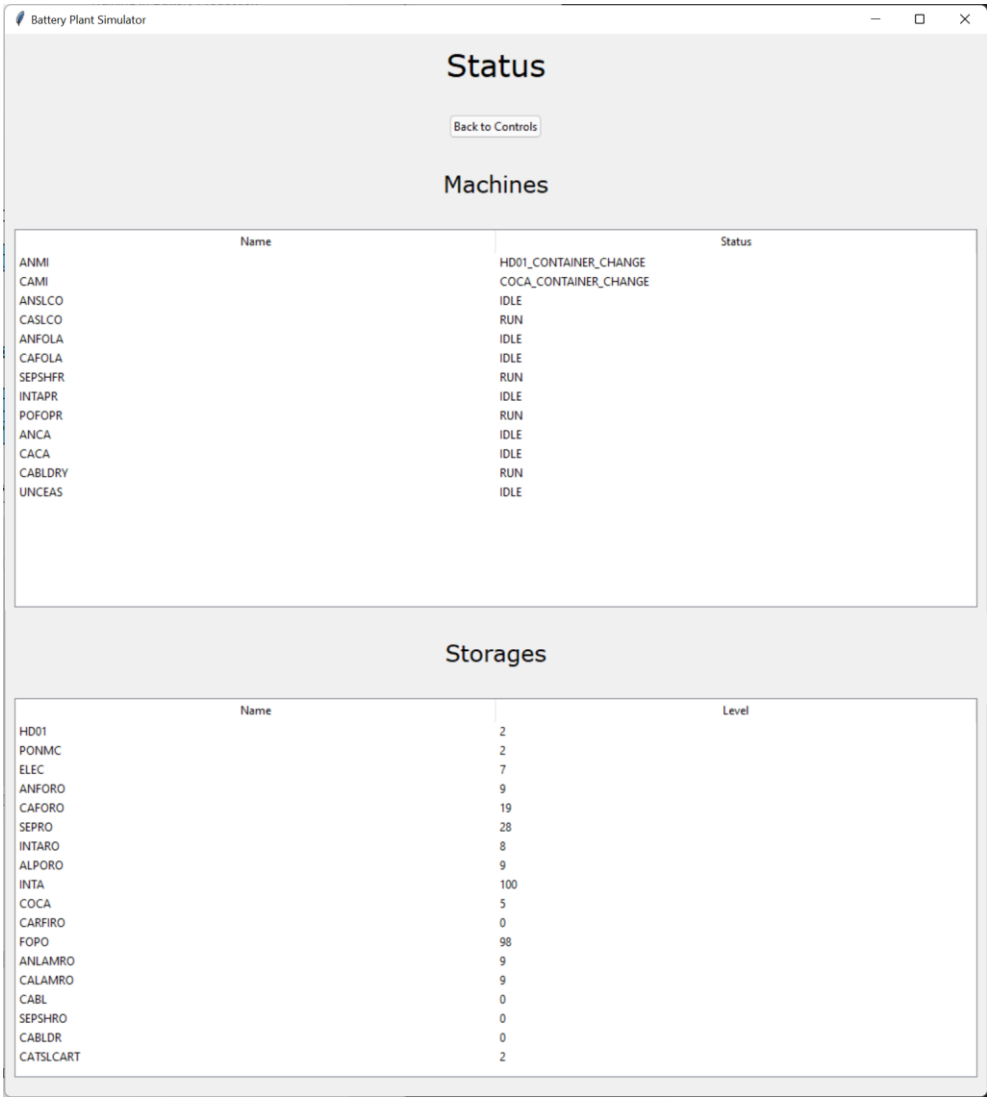

Figure 3-13: GUI of Simulator Control Application (Status page)

The status page is updated every second to display new data. The update of the view is done using the refresh function seen in Code 3-19.

```
class StatusPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        global statustree, storagetree
        label = tk.Label(self, text="Status", font=LARGE_FONT)
        label.pack(pady=10, padx=10)

        button1 = ttk.Button(self, text="Back to Controls",
                             command=lambda: controller.show_frame(StartPage))
        button1.pack(pady=20)

        label1 = tk.Label(self, text="Machines", font=MEDIUM_FONT)
        label1.pack(pady=10, padx=10)
        statustree = ttk.Treeview(self, columns=('name', 'status'), show='headings')
        statustree.heading('name', text='Name')
```

```
        statustree.heading('status', text='Status')
        statustree.pack(pady=20, padx=10, fill=tk.BOTH, expand=True)

        label2 = tk.Label(self, text="Storages", font=MEDIUM_FONT)
        label2.pack(pady=10, padx=10)

        storagetree = ttk.Treeview(self, columns=('name', 'level'), show='headings')
        storagetree.heading('name', text='Name')
        storagetree.heading('level', text='Level')
        storagetree.pack(pady=20, padx=10, fill=tk.BOTH, expand=True)

        self.after(0, self.refresh())
    def refresh(self):
        for item in statustree.get_children():
            statustree.delete(item)
        for machine in machines:
            statustree.insert('', 'end', values=(machine.name, machine.status))

        for item in storagetree.get_children():
            storagetree.delete(item)
        for storage in storages:
            storagetree.insert('', 'end', values=(storage.name, storage.level))
        self.after(1000, self.refresh)
```

Code 3-19: The StatusPage class

In the Storage graph page in Figure 3-14, the storage levels of the simulation can be viewed over time.

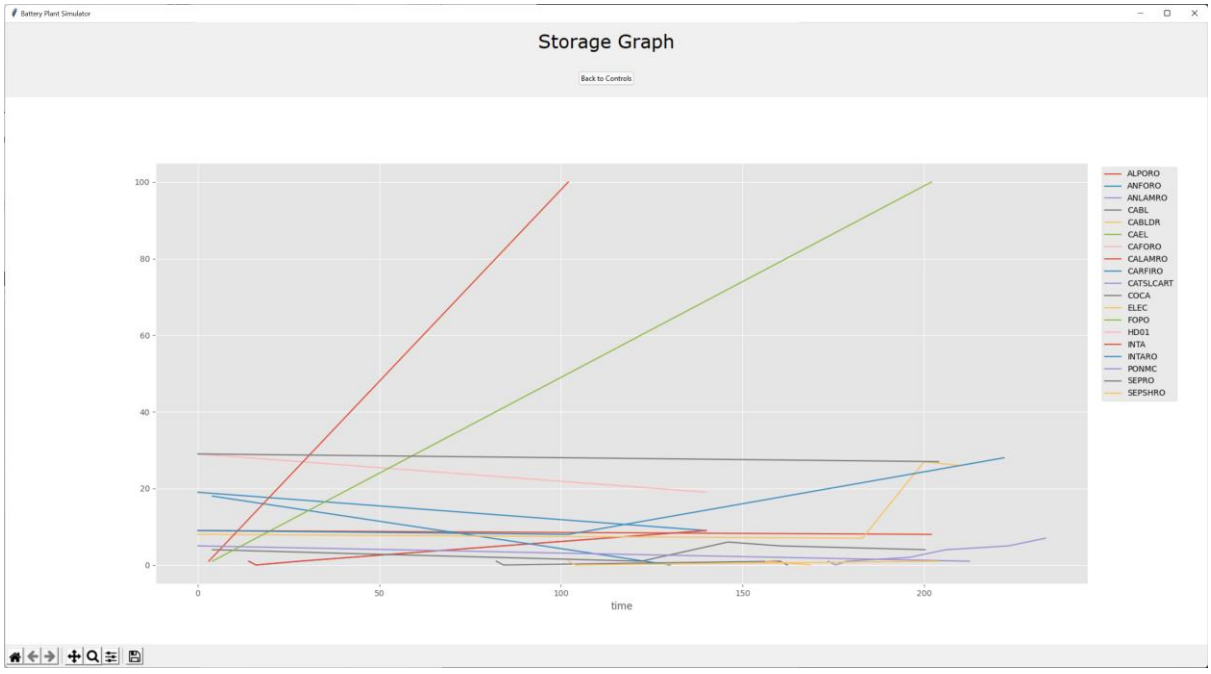

Figure 3-14: GUI of Simulator Controls Application (Storage graph page)

The Logging page seen in Figure 3-15 uses a TextHandler and logging handler enhancement developed by Moshe Kaplan [45] to display logs of incoming messages in the GUI as well as logging them to file.

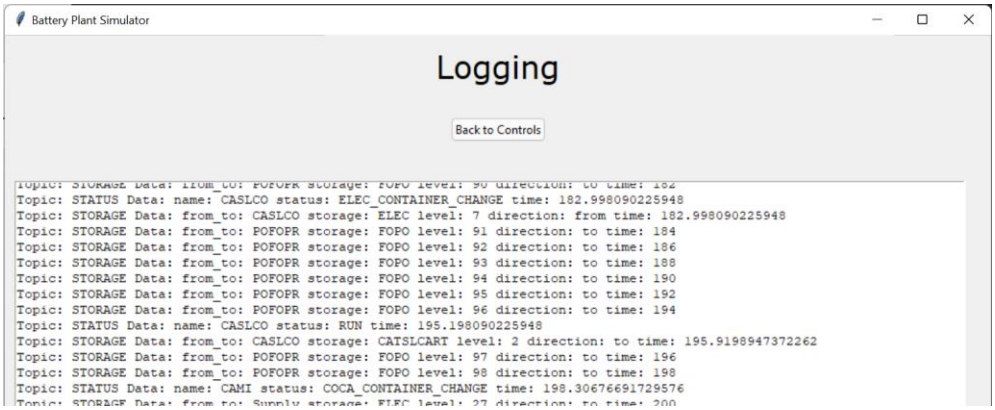

Figure 3-15: GUI of Simulator Controls Application (Logging page)

In the logging page, all incoming messages from the subscription socket can be viewed.

3.5 Unity VR implementation

The VR-solution is done using the Unity development environment as seen in Figure 3-16. The solution has been created with a series of game objects seen in the hierarchy in Figure 3-17. These game objects have different features assigned to them, e.g., Transform which define the location, rotation and scale of the game object in the virtual world and other features programmed in C# scripts.

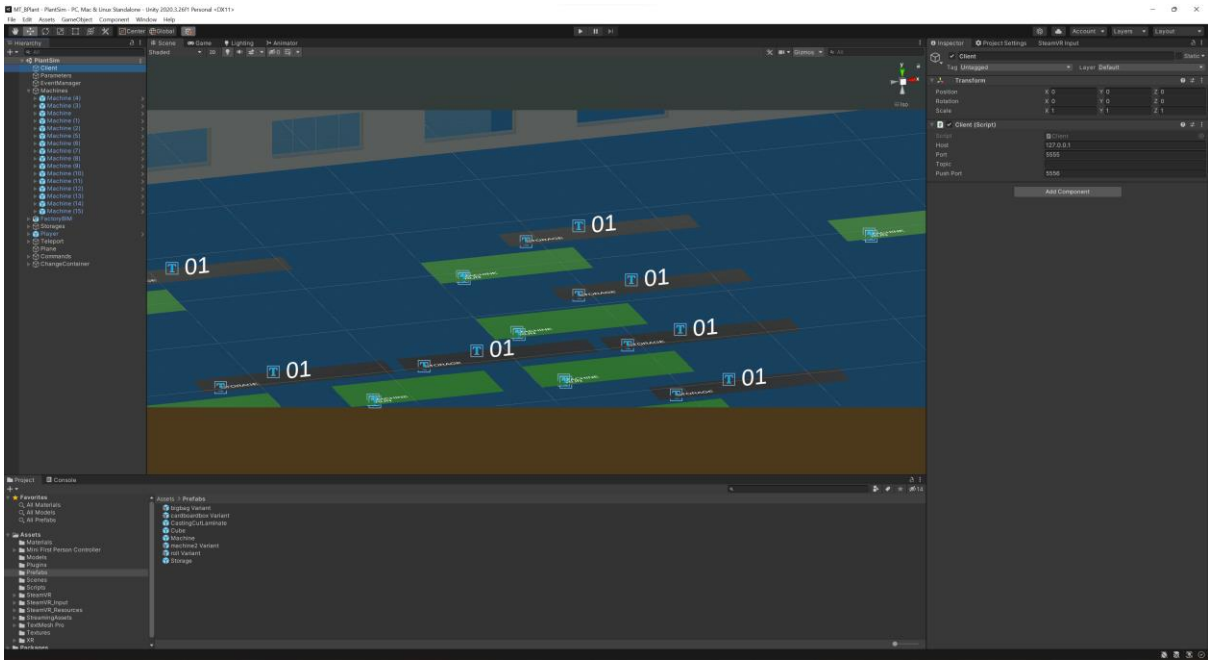

Figure 3-16: Unity Development environment

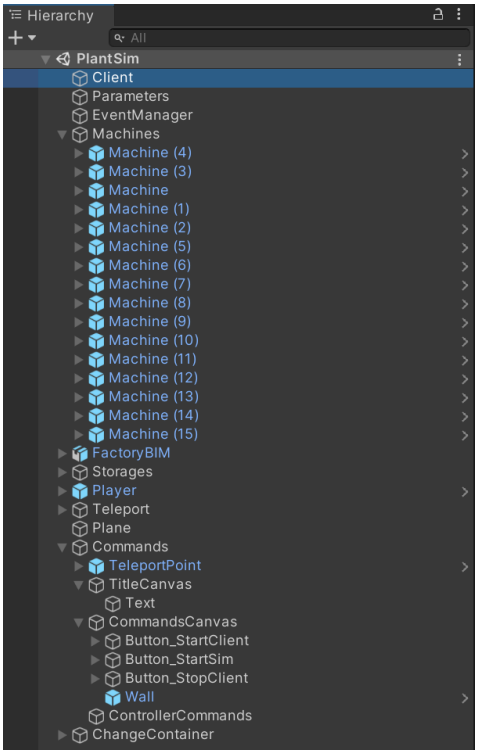

Figure 3-17: The VR solution hierarchy

In Code 3-20 the ReadParameters class is shown, this class handles loading of the simulation parameters from the same file as used in the Python program in 3.3. The class reads the file line by line, and adds the results from it to a Dictionary, which can be referenced in other scripts.

```
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;

public class ReadParameters : MonoBehaviour
{
    public static ReadParameters Instance;
    public Dictionary<string, string> setupParameters = new Dictionary<string, string>();
    string fileName = "SimParameters.cfg";

    // Start is called before the first frame update
    void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            StreamReader sr = new StreamReader(Application.dataPath + "/" + fileName);
            string line;

            while (!sr.EndOfStream)
            {
                line = sr.ReadLine();
                if (line.Contains("="))
                {
                    string[] arr = line.Split('=');
                    string key = arr[0].Trim();
                    string value = arr[1].Trim();
```

```
                    setupParameters.Add(key, value);
                }
            }
        }
        else
            Destroy(this);

    }
}
```

Code 3-20: Loading of paramters in Unity

In Code 3-21 the Update() function of the MachineHandler class is shown where the text object connected to the machine game object is updated with new text if a change in status is seen, as well as the color of the floor objects under the machine's virtual representation is changed with regards to the current status. The last part of the code checks if the player object is further away from the machine than a set draw distance, and if so, disables the model. This check is to lighten the load of the virtual environment rendering, in this way only the machines nearest to the viewer needs to be handled, thus saving processing power.

```
void Update()
    {
        if(statusTextMesh.text != machineStatus){
            statusTextMesh.text = machineStatus;
            switch (machineStatus)
            {
                case "RUN":
                    floorIndicator.GetComponent<MeshRenderer>().material.color = Color.green;
                    break;
                case "DOWN":
                    floorIndicator.GetComponent<MeshRenderer>().material.color = Color.red;
                    break;
                case "IDLE":
                    floorIndicator.GetComponent<MeshRenderer>().material.color = Color.blue;
                    break;
                default:
                    floorIndicator.GetComponent<MeshRenderer>().material.color = Color.yellow;
                    break;
            }
        }
        if (lastPos != player.position) {
            if (Vector3.Magnitude(player.position - transform.position) > drawDistance)
            {
                machine.SetActive(false);
            }
            else
            {
                machine.SetActive(true);
            }
            lastPos = player.position;
        }
    }
```

Code 3-21: MachineHandler class Update function

The parameters of the Machine handler are set from the development environment of Unity as seen in Figure 3-18. Here the machine name is assigned, the Id of the machine (which is used to link synchronizing messages with the object), draw distance and several other parameters that are linked to script.

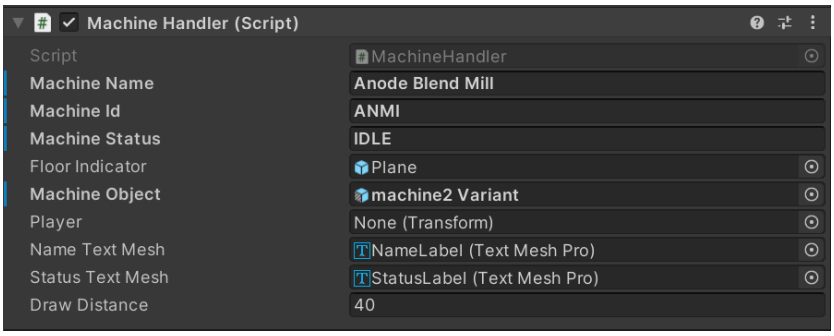

Figure 3-18: Parameter setup of a Machine handler

The storage handler script is parametrized in a similar way as the machine handler script, but with different parameters as seen in Figure 3-19.

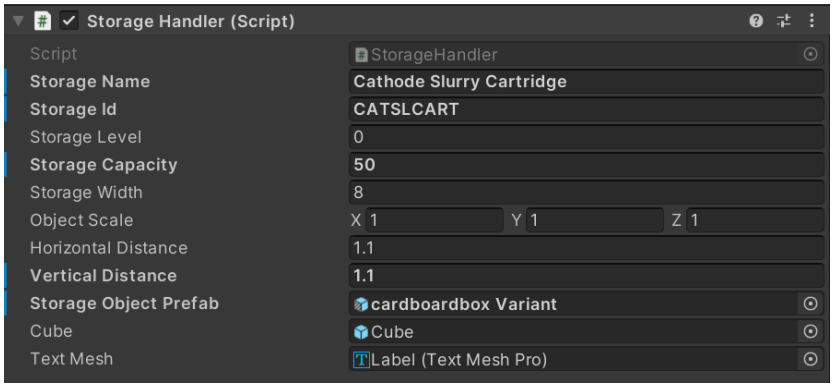

Figure 3-19: Parameter setup of a Storage handler

The incoming messages are handled depending on the message's topic by the HandleMessage function of the Client class as seen in Code 3-22. If the topic contains "STORAGE" the message data is used to update the level of the corresponding storage object's handler script. And if the topic contains "STATUS" the message data is used to update the corresponding machine.

```
private void HandleMessage(string message)
        {
            Debug.Log(message);
            if (message.Contains("STORAGE"))
            {
                string jsonString = message.Substring(message.IndexOf("STORAGE") + "STORAGE ".Length);
                StorageUpdate storageUpdate = JsonUtility.FromJson<StorageUpdate>(jsonString);
                GameObject storageObject = GameObject.Find(storageUpdate.storage);
                if (storageObject != null)
                {
                    storageObject.GetComponent<StorageHandler>().SetLevel(storageUpdate.level);
                }
            }
            else if (message.Contains("STATUS"))
            {
                string jsonString = message.Substring(message.IndexOf("STATUS") + "STATUS ".Length);
                StatusUpdate statusUpdate = JsonUtility.FromJson<StatusUpdate>(jsonString);
                GameObject statusObject = GameObject.Find(statusUpdate.name);
                if (statusObject != null)
                {
                    statusObject.GetComponent<MachineHandler>().SetStatus(statusUpdate.status);
                }
            }
        }
```

Code 3-22: Client class HandleMessage function

The ØMQ implementation in Unity was largely based on the works of Nicola S. [42] for the subscribing part of the communications, but in addition a Sender class was developed to handle outgoing messages from the Unity project, where messages were published using the ØMQ publisher class, seen in Code 3-23.

```
    public class Sender
    {
        private readonly string _host;
        private readonly string _port;
        private PublisherSocket _sender;

        public Sender(string host, string port)
        {
            _host = host;
            _port = port;
        }

        public void Start()
        {
            _sender = new PublisherSocket();
            _sender.Connect($"tcp://{_host}:{_port}");
        }

        public void Stop()
        {
            _sender.Close();
        }

        // Start is called before the first frame update
        public void SendMessage(string topic, string message)
        {
            string msg = topic.ToString() + " " + message.ToString();
            _sender.SendFrame(msg);
        }

    }
```

Code 3-23: Sender class for publishing from Unity to the simulator

Setting up the VR part of the project using the SteamVR API was solved using the ready-made components provided in the Interaction samples of the SteamVR API package. For navigating the environment, the teleport functions of the samples were used as seen in Figure 3-20.

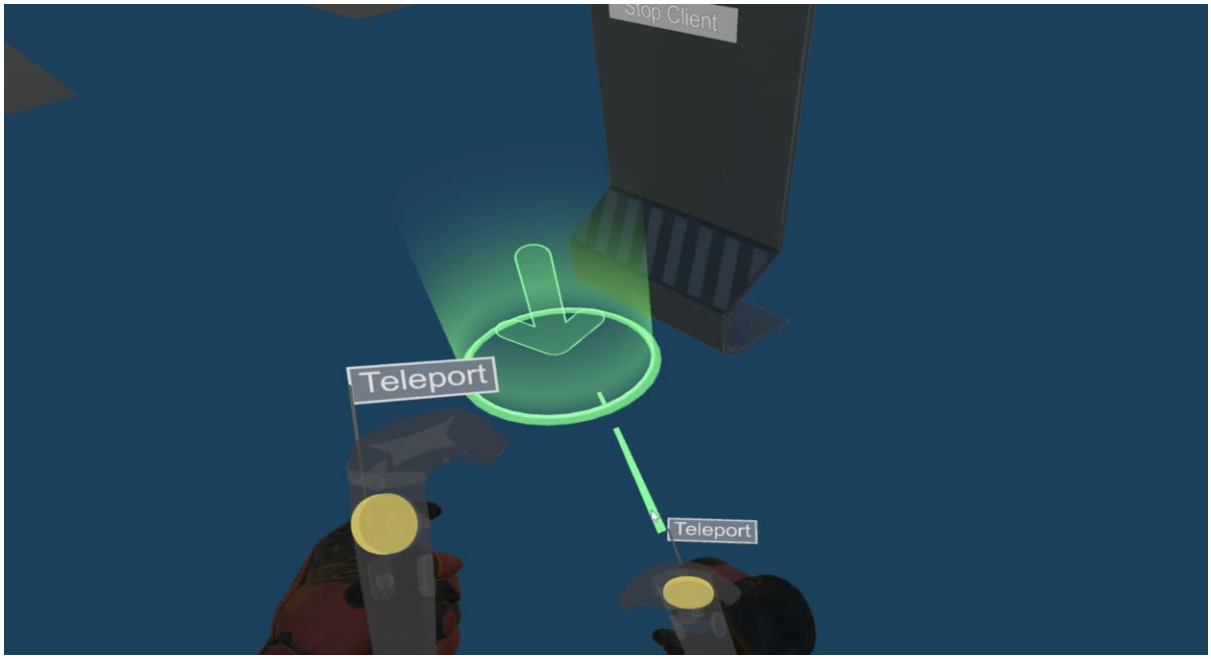

Figure 3-20: Using teleport to move around the VR environment

Menus placed in the environment were set up to be manipulated with the hand controllers as seen in Figure 3-21.

Figure 3-21: Using hand interaction with menus in the environment

In Figure 3-22 and Figure 3-23 two different machine statuses can be seen rendered on the floor of the environment. Machines with “IDLE” status are rendered blue, and “RUN” rendered green. The machine in Figure 3-23 also has a menu connected to it where the user can send “Change” commands back to the simulation, simulating the change of material in the VR-environment. The simulation will in turn respond by changing status from waiting-for-new-container-status to running again.

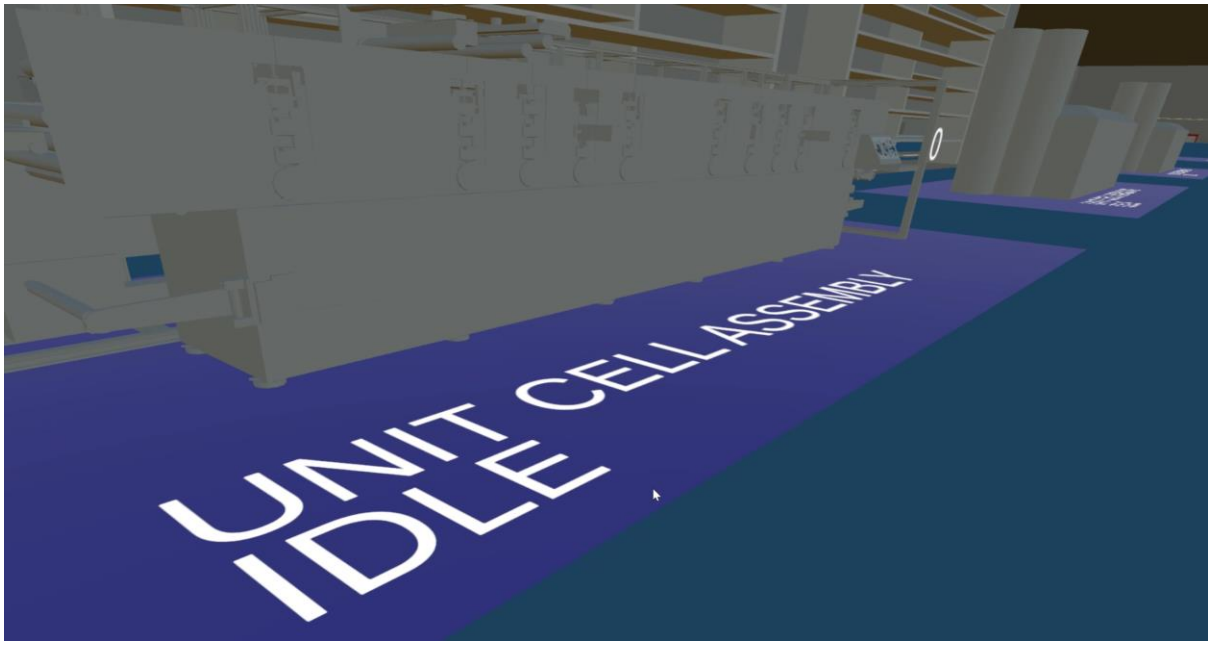

Figure 3-22: Displaying the status of simulated processes in the environment

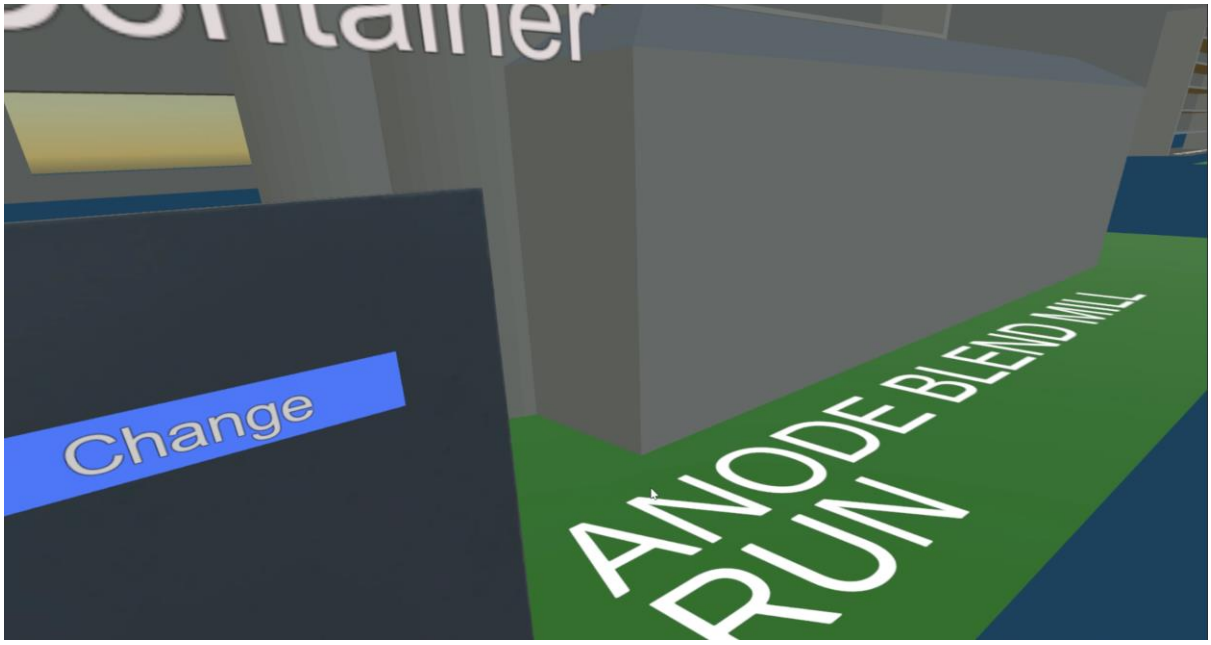

Figure 3-23: Interacting with the machine simulation from the VR environment

Storages are visualized with the corresponding number of “material” or products shown in the shelfs, and the level/number of items is shown in 3D text objects as seen in Figure 3-24.

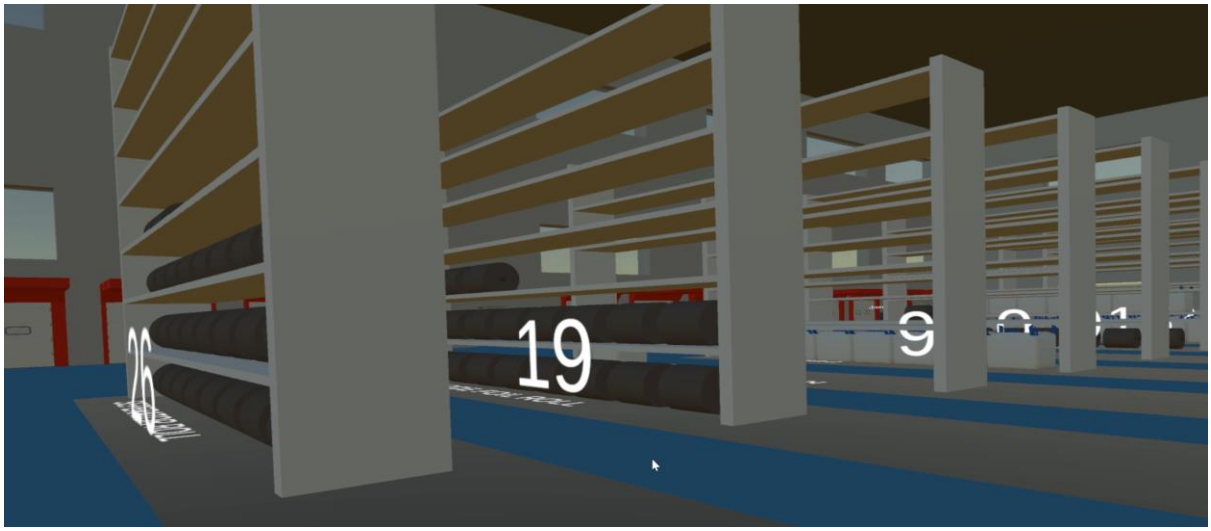

Figure 3-24: Displaying storage levels in the VR environment

Analyzing the performance of the solution is done by checking the stats while running the solution in the Unity Editor, in addition to checking the profiler seen in Figure 3-25. The running solution averages at a frame rate around 90-100 frames per second (FPS) while in short periods dips below 60 FPS.

Figure 3-25: Unity Profiler output

3.6 VR-solution and communications testing

To test the performance of the VR-solution a couple of scripts were made to record the performance in Unity (Appendix J), as well as a script to stress test the communication setup from Python (Appendix K).

Three runs were recorded and the data from the runs were analyzed in this section. The first run recorded is a normal running of the simulation while the two other runs were runs done during stress testing of the communication/handling solution. The stress test starts with sending of 1 update per second while the number of messages is doubled after every ten sent message in run 2 and after every 5 second passed in run 3, until the test is stopped. The output from the test is stored in CSV format which gives us data as in Table 3-2.

Table 3-2: Example data from performance test

Time	fps	mpsr	mpst	noo	not	nov
8,04	89,51	8,89	0,99	1680	1629646	2981184
9,05	90,54	0	0	1647	1005352	2228660
10,06	78,41	5,93	0	1645	865860	1001292
11,06	87,27	0	0	1672	870548	1003500
12,06	90,15	2,99	0	1649	486542	540132
13,07	90,28	0	0	1680	188986	209388
14,08	90,25	0	0	1652	47742	56204
15,08	90,49	1	1	1654	634854	912746
16,09	90,43	0	0	1655	501638	560690

The values collected was the time in seconds from start of the program (Time), frames per second displayed (fps), messages received per second (mpsr), messages transmitted per second (mpst), number of objects in the scene (noo), number of triangles rendered (not) and number of vertices rendered (nov). Additionally, a control was performed to see if any messages were lost on the way to the VR solution in the second and third run.

The data was processed using a python script (Appendix L) to plot the data as well calculating correlation between the variables, as well as a principal component analysis was performed on the data to see if there were any patterns to observe.

In the first run which lasted 257 seconds the output the normal operating conditions of the solution was evaluated. The output data from the run is seen in Figure 3-26 and Figure 3-27. As seen the FPS count is quite stable at 90 FPS (which is the max refresh rate of the HMD), with some dips down to 70-75 FPS at times.

The number of messages received by the solution starts low with between 0-7 messages a second, while after 170 seconds there are between 4 and 14 messages received a second. Messages going from the VR solution to the Python simulation are quite low, to see if more messages affected the fps performance some additional messages were sent around 30 seconds and 200 seconds into the run. The number of objects in the run rises during the simulation as products are created.

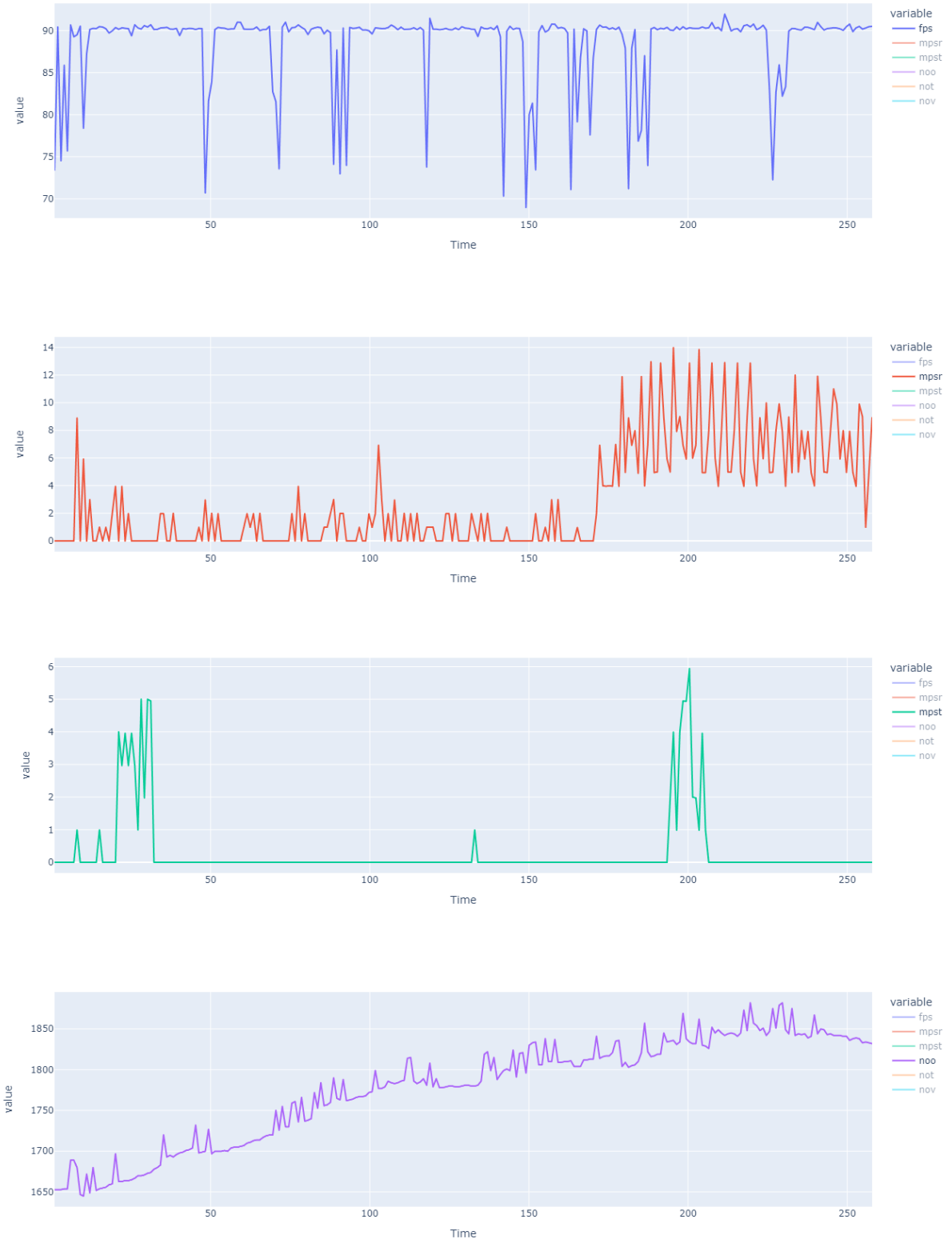

Figure 3-26: Data plots for run 1 (fps, mpsr, mpst, noo)

Figure 3-27: Data plots for run 1 (not, nov)

As can be seen from Figure 3-27 the number of triangles and vertices are tightly connected and varies throughout the run, this is due to Unity's occlusion feature, which makes the rendering engine of Unity only process the triangles of objects which is in the view of the game camera. Making the number of triangles/vertices be dependent on the complexity of the objects viewed at different times during the run.

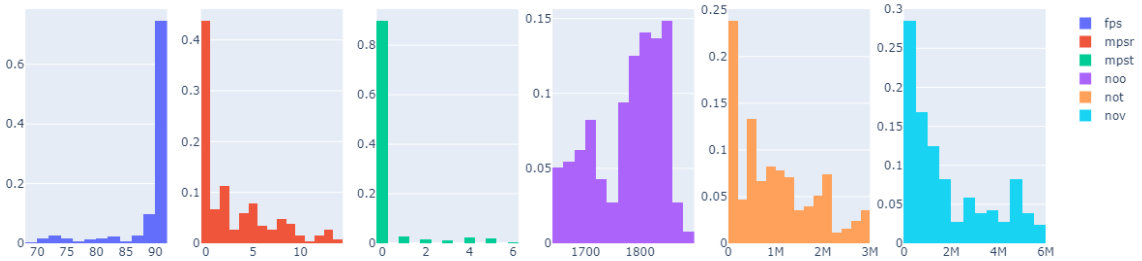

Figure 3-28: Distribution plots for run 1

The distribution of values in the data can be seen in Figure 3-28, where the fps has a high number of recordings around 90, mpsr is mainly at 0, but range from 0-14, mpst is about 90% of the time at 0 and range from 0-6 messages a second. The number of game objects in the

solution range from 1600-1900, number of triangles range from almost 0-3 million and vertices from 0-6 million.

In Figure 3-29 a PCA is performed on the data from run 1, to see if there are any patterns to be seen in the data. The color of the points in the plot is ranging from yellow which is a point in time with high FPS, to blue where the FPS is low. The data points are spread over the entire plot with no apparent clustering, looking at the loadings (drawn as lines from the center), the fps/mpst and the not/nov loadings point in opposite directions. While the mpst pointing in the opposite of not/nov is probably coincidental, it's more probable that a drop in fps could be caused by a high number of triangles/vertices rendered. That the Time/noo/mpsr loadings are pointing in the same direction is natural as the number of objects in the scene rises over time, as does the number of messages received during the simulation.

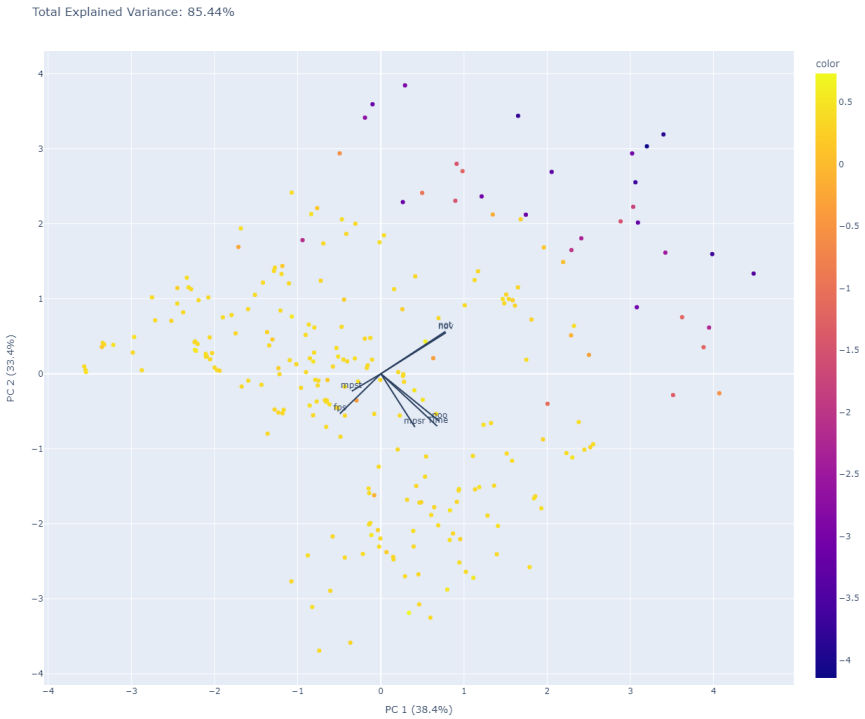

Figure 3-29: PCA for run 1

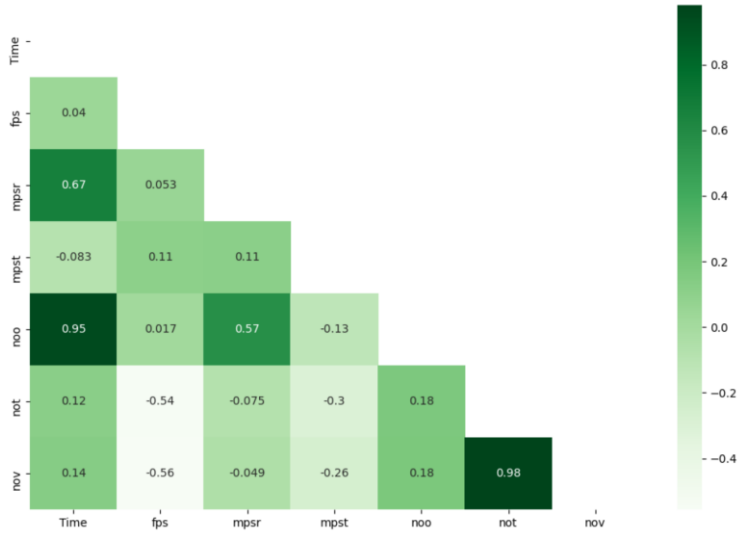

Figure 3-30: Correlation matrix for run 1

Another way of looking at the data is by checking the correlation between the different recorded data. In Figure 3-30 a correlation matrix is presented, with the correlation number between the variables on the vertical and horizontal axis shown in the matrix. The highest correlation is between number of triangles and vertices, which is to be expected. Number of objects in the scene and time is the next highest number, which is explained by the gradual rise of number of objects in the simulation. Fps vs not/nov has a negative correlation of -0.54/-0.56 which is interesting. Fps vs Time is 0.04 which is a good indication that the solution is not much affected by how long the solution has been running, at least not for runs of this length.

In the second run the normal simulation was replaced with a communication stress test, where the VR-solution received an increasing number of messages during a run of about 170 seconds. In Figure 3-31 some of the data from run 2 is presented, fps, messages received per second and the number of triangles.

What can be observed is that even though the Python script is sending messages at an increasing rate, the number of messages handled by the VR application stops increasing after 36 seconds when it's at a rate of 63 messages per second.

The FPS seem to be unaffected by the number of messages received and stays at a steady 90 FPS until the amount of rendered triangles is high over a period from 130 seconds to 160 seconds. The drop in fps to zero in the end is because of the closing of the application.

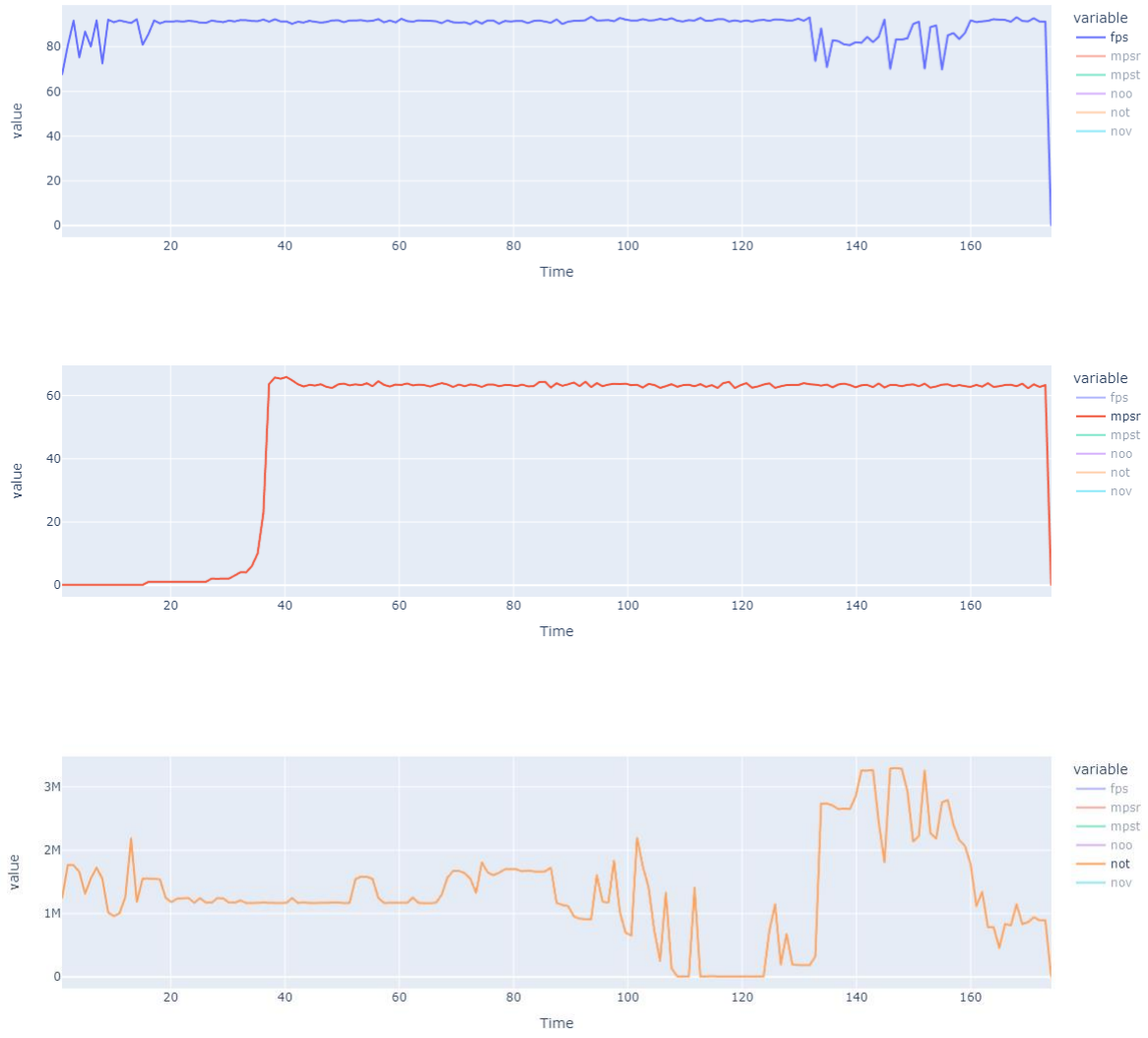

Figure 3-31: Data plots for run 2

Looking at the distribution of the values in the second run shown in Figure 3-32, it is observed that the FPS is more stable in this case, which is probably due to less variations in rendered triangles/vertices.

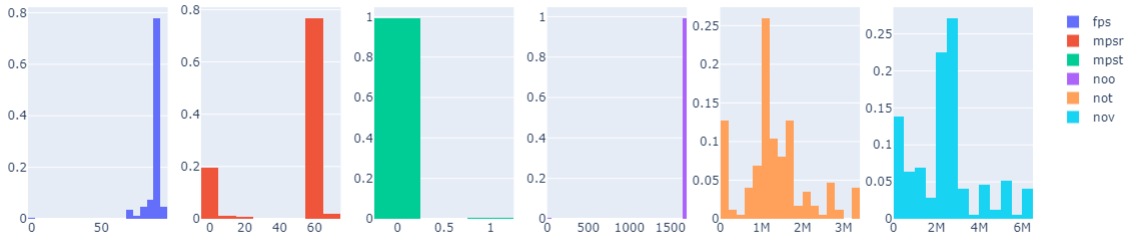

Figure 3-32: Distribution plots for run 2

Looking at the PCA in Figure 3-33, an outlier is observed in the top right of the plot, which is probably the second where a 0 FPS was recorded at the end of the run. Otherwise as before there are no clear clusters to be seen from the PCA.

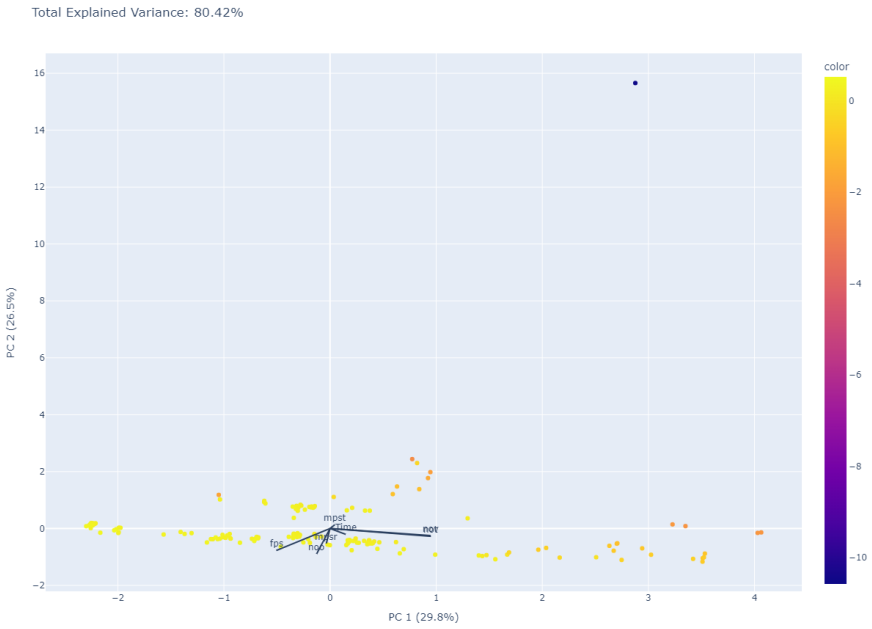

Figure 3-33: PCA for run 2

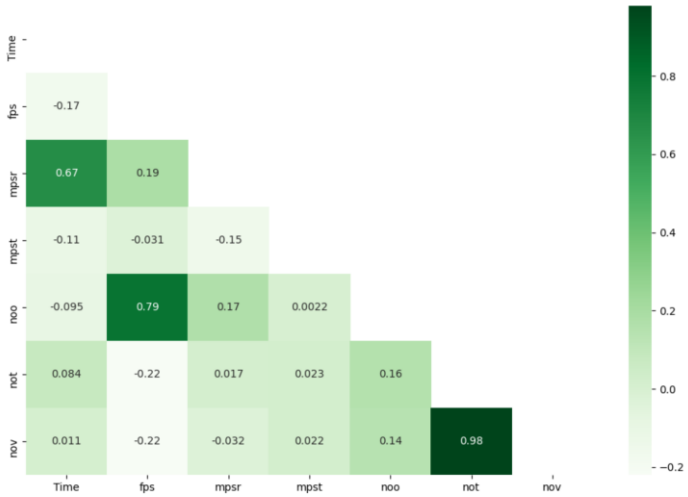

Figure 3-34: Correlation matrix for run 2

Not much new information can be extracted from the correlation matrix in Figure 3-34, still the number of triangles is the value that has the biggest negative impact on the fps.

In the third run the increase of messages from the Python script was changed to increase every 5 seconds generating the data plotted in Figure 3-35 and Figure 3-36.

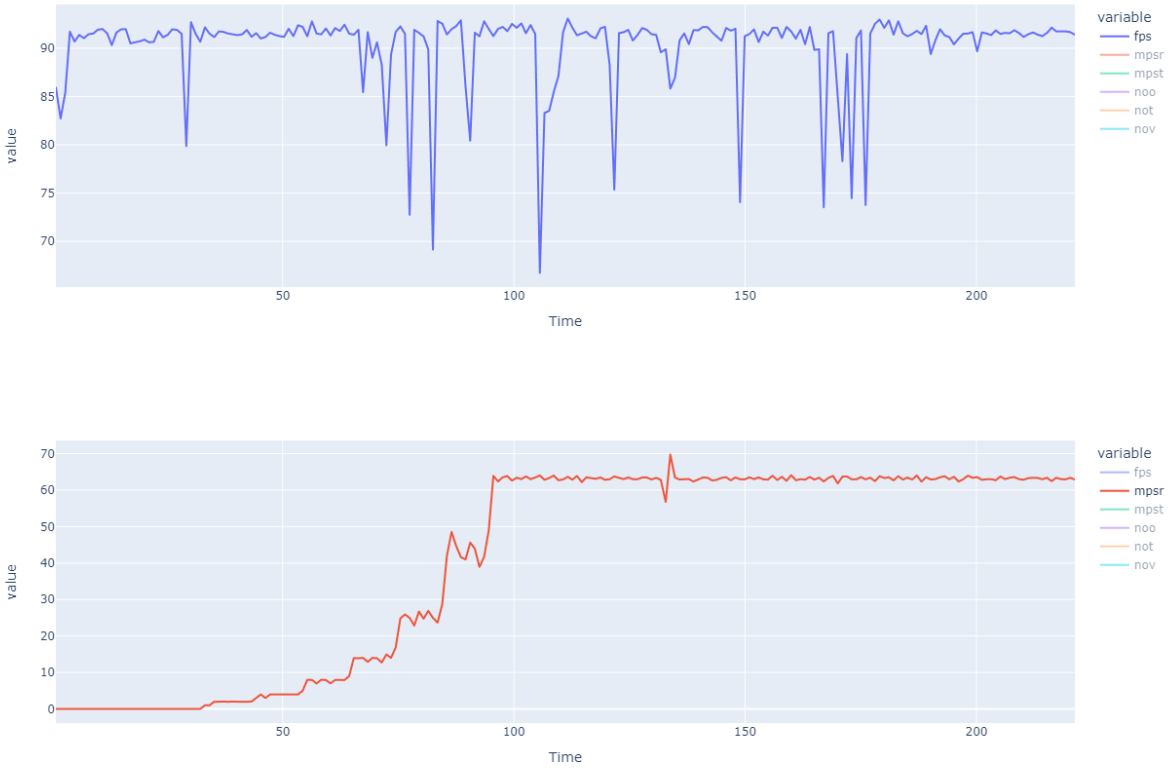

Figure 3-35: Data plots for run 3 (fps, mpsr)

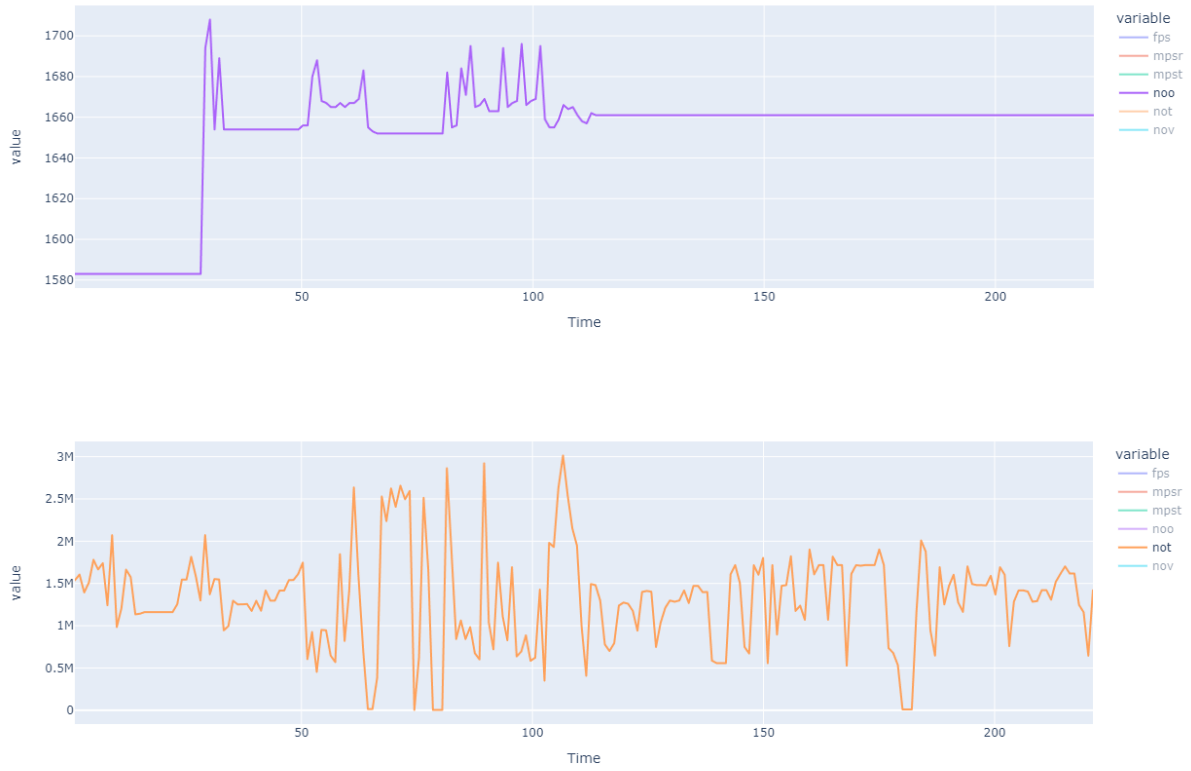

Figure 3-36: Data plots for run 3 (noo, not)

The distribution of the values shown in Figure 3-37 is very similar to that of the previous run.

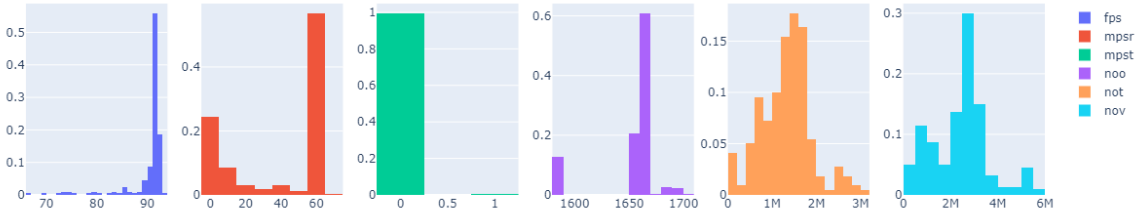

Figure 3-37: Distibution plots for run 3

Looking at the PCA in Figure 3-38 and the correlation matrix in Figure 3-39, as before there is a negative correlation between the frames per second metric and the number of triangles rendered, but no correlation to mention between messages handled per second and the frames per second.

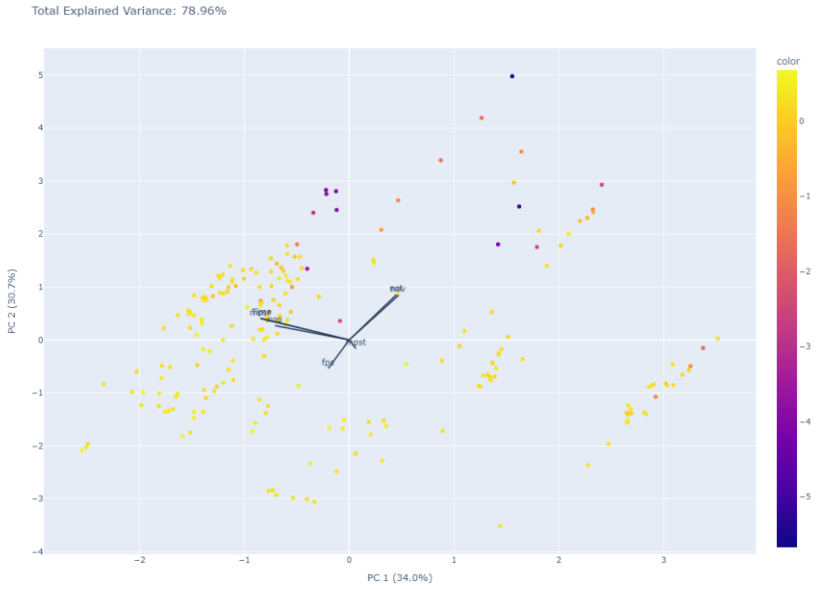

Figure 3-38: PCA for run 3

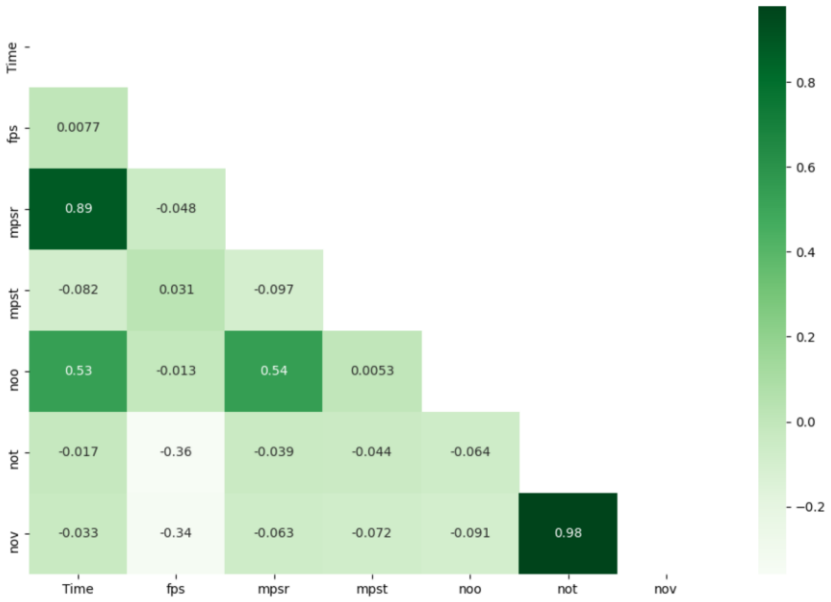

Figure 3-39: Correlation matrix for run 3

4 Discussion and conclusion

In this thesis the goal was to review research on the topic of combining digital twins/data driven models with VR solutions, developing a dynamic material flow model of a battery plant production process and a connected VR environment applying best practices, easy to configure network-based communications and evaluate the performance, versability and usability of the solution.

The developed system consists of all the parts envisioned, but the material flow model developed is not tuned or compared to a real system, due to lack of needed information and data on an actual production plant. The material flow model is implemented in a way so that this tuning of parameters is easily done by changing configuration of each process as well as the number of machines and sizes of buffers and storages.

IrisVR states that framerates below 90 frames per second is likely to induce disorientation, nausea and other negative effects [46], while other sources state that if the framerate does not go below 60 its fine [47]. The implemented VR solution in this project targets 90 FPS, which is also the refresh rate of the HTC Vive Pro headset used for the project. According to test performed in chapter 3.6, the solution manages in a large degree to stick to the target, with only some cases where the framerate goes below, but is still higher than 60 FPS.

The communication between the VR and simulation software is able to handle a large number of messages, but with a maximum of about 63 messages processed per second. At higher message rates than 63 per second the messages are put into a queue and handled in the order they are sent from the simulator. According to tests performed, which is described in 3.6 no messages are dropped, as the number of messages sent is the same as the number handled in the VR application.

With the ØMQ publish/subscribe implementation, the solution handles the messages with no experienced delay when the simulator is run in normal operation. The pub/sub model of messaging is working well for this purpose, and topics can be used in smart ways to limit the number of messages each unit in the system needs to receive and process.

4.1 Future improvements

To make this solution into a fully functioning digital twin there needs to be established a connection to the physical environment it is representing. Possible communication interfaces towards the physical environment and entity could be OPC UA or MQTT which is probably the most popular technologies mentioned in literature. With this, new possibilities would arise as the use of artificial intelligence to tune the parameters of virtual entities in the twin. This would also make it possible to display actual values from the process in the VR environment instead of simulated.

To improve the realism of the VR experience, more realistic representations of the actual factory layout, and models of all the processes should be included. As well as introducing

different actors on the plant floor moving around doing tasks like transporting material and products around the environment.

The material flow model could be improved by creating dynamic models of the transport of materials and products in the factory, this could include positional data of each machine an adjust the transport model accordingly.

The Unity solution is developed in such a way that the objects and scripts could be reused in implementations for other XR technologies. The reuse capabilities of the Unity solution could be evaluated and improved if such projects are done. The developed solution is ready to be used with other VR headsets and other controllers, but has only been tested using the equipment listed in 2.1.

The publish and subscribe communication model could have been improved in the project with a better topic hierarchy in place, a topic hierarchy with a structure like process area code/process (or product) code/message type e.g., “100/110/status” would make it possible to filter subscription topics in a much better way. This could also be used in ways to tackle the limited message processing capabilities at 63 messages per second of the VR solution. E.g., by subscribing only to messages of machines nearest in the VR-environment.

4.2 Conclusion

The objective of the thesis has been to model part of the battery production process and visualize them in a 3D environment. The purpose of the solution is to enable the identification of production bottlenecks and inefficiencies, optimizing factory layout, and be used for training. The goal has been a solution that is usable in a future full scale digital twin implementation, with reusable and reconfigurable solutions.

The thesis has given an overview of research on digital twin technologies, discrete event simulations, extended reality solutions and the combination of these. The thesis has identified some best practice methods and identified benefits of building digital twin solutions.

The thesis has identified discrete event simulations as a great way to represent material flow of a battery plant production line. Large parts of a battery production line have been simulated. The solution has been developed using the SimPy framework in Python, with efforts made to make the solution flexible and customizable.

A VR environment has been developed using the Unity game engine and the SteamVR API. The environment is built with a combination of actual BIM files from the external partner and models created for the project in Blender.

The two solutions have been integrated using the ØMQ messaging library which provide fast, lightweight messaging with easy-to-use APIs. The solution is built using a publisher/subscriber model which provides message topic filtering, a solution that can give benefits in the future with an increase of messages moving through the system.

Several scripts have been developed to animate the status and levels of storages and machines in the virtual environment.

The solution performance has been evaluated and key limitations has been identified.

The thesis project has resulted in a solution that fulfills the objectives it set out to, but there are still a lot of work that needs to be completed if the solution is to be considered as a fully functioning DT/VR integrated solution as stated in 4.1.

Testing show that the selected technologies can handle the task at hand and should be suitable for a state-of-the-art solution, but care needs to be taken if there is an increase in complexity of the simulation, considering the limitations that have been identified.

5 References

[1] B. R. Barricelli, E. Casiraghi and D. Fogli, "A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications," *IEEE Access, Volume 7, 2019,* 19 November 2019.

[2] D. Jones, C. Snider, A. Nessehi, J. Yon and B. Hicks, "Characterising the Digital Twin: A systemic litterature review," *CIRP Journal of Manufacturing Science and Technology,* vol. 29, pp. 36-52, 2020.

[3] W. Kritzinger, M. Karner, G. Traar, J. Henjes and W. Sihn, "Digital Twin in manufacturing: A categorical literature review and classification," *IFAC PapersOnLine,* no. 51-11, pp. 1016-1022, 2018.

[4] D. G. Broo and J. Schooling, "Digital twins in infrastructure: definitions, current practicesm challanges and strategies," *International Journal of Construction Management,* 2021.

[5] A. Bécue, E. Maia, L. Feeken, P. Borchers and I. Praca, "A New Concept of Digital Twin Supporting Optimization and Resilience of Factories of the Future," *Applied Sciences,* no. 10, 28 June 2020.

[6] G. Modoni, E. G. Caldarola, M. Sacco and W. Terkaj, "Synchronizing physical and digital factory: benefits and technical challanges," in *12th CIRP Conference on Intelligent Computation in Manufacturing Engineering* , Gulf of Naples, Italy, 2018.

[7] A. Varga, *OMNeT++ Discrete Event Simulation System Version 3.2 User Manual,* 2005.

[8] J. Barrett , B. Jayaraman, D. Patel and J. Skolnik, "Kinetic modeling and simulation: Discrete Event Simulation," University of Pennsylvania, [Online]. Available: https://www.med.upenn.edu/kmas/DES.htm. [Accessed May 2022].

[9] S. Bangsow, Use Cases of Discrete Event Simulation: Appliance and Research, Zwickau: Springer, 2012.

[10] A. Çöltekin, I. Lochhead, M. Madden, S. Christophe, A. Devaux, C. Pettit, O. Lock, S. Shukla, L. Herman, Z. Stachon, P. Kubícek, D. Snopková, S. Bernardes and N. Hedley, "Extended Reality in Spatial Sciences: A Review of Research Challanges and Future Directions," *International Journal of Geo-Information,* 15 July 2020.

[11] Virtual Reality Society, "History of Virtual Reality," 2019. [Online]. Available: https://www.vrs.org.uk/virtual-reality/history.html. [Accessed May 2022].

[12] J. Oyekan, W. Hutabarat, C. Turner, A. Tiwari, N. Prajapat, N. Ince, X.-P. Gan and T. Waller, "A 3D immersive Discrete Event Simulator for enabling prototyping of factory

layouts," in *The Fourth International Conference on Through-life Engineering Services*, 2015.

[13] V. Harvard, B. Jeanne, M. Lacomblez and D. Baudry, "Digital twin and virtual reality: a co-simulation environment for design and assessment of industrial workstations," *Production & Manufacturing Research,* pp. 472-489, 2019.

[14] V. Kuts, G. E. Modoni, T. Otto, M. Sacco, T. Tähemaa, Y. Bondarenko and R. Wang, "Synchronizing physical factory and its digital twin through an IIoT middleware: a case study," *Proceedings of the Estonian Academy of Sciences,* pp. 364-370, 22 October 2019.

[15] X. Xin, "Research on Digital Manufacturing of Lithium Battery Pilot Production Line Based on Virtual Reality," in *Journal of Physics: Conference Series*, 2021.

[16] D. Gajsek, "Unity vs Unreal Engine for XR Development: Which one is better?," Circuit Stream, 12 February 2022. [Online]. Available: https://circuitstream.com/blog/unity-vs-unreal/. [Accessed May 2022].

[17] Unity, "Unity Asset Store," Unity, May 2022. [Online]. Available: https://assetstore.unity.com/3d.

[18] Unity, "Inspiring examples of extended reality," [Online]. Available: https://unity.com/pages/industrial-stories. [Accessed May 2022].

[19] Epic Games, "Unreal Engine for extended reality (XR)," [Online]. Available: https://www.unrealengine.com/en-US/xr. [Accessed May 2022].

[20] Wikipedia, "Python (programming language)," 2022. [Online]. Available: https://en.wikipedia.org/wiki/Python_(programming_language). [Accessed May 2022].

[21] B. Jain, "Simulating a Queueing System in Python," Towards Data Science, 6 June 2020. [Online]. Available: https://towardsdatascience.com/simulating-a-queuing-system-in-python-8a7d1151d485. [Accessed May 2022].

[22] M. A. d. Mesquita, F. B. d. A. Rocha Mariz and J. V. Tomotani, "The Skateboard Factory: A teaching case on discrete-event simulation," *Production,* vol. 2017, no. v27, 2017.

[23] K. Popp, "Moddy Discrete Event Simulator," [Online]. Available: https://klauspopp.github.io/Moddy/. [Accessed May 2022].

[24] Wikipedia, "List of discrete event simulation software," April 2022. [Online]. Available: https://en.wikipedia.org/wiki/List_of_discrete_event_simulation_software.

[25] A. Øvern, "Industy 4.0 - Digital Twins and OPC UA," Norwegian University of Science and Technology, Trondheim, 2018.

[26] M. V. Masdani and D. Darlis, "A comprehensive study on MQTT as low power protocol for internet of things application," in *3rd Annual Applied Science and Engineering Conference*, 2018.

[27] G. P. Viganò, "M2MQTT for Unity," [Online]. Available: https://github.com/gpvigano/M2MqttUnity. [Accessed May 2022].

[28] ØMQ, "ØMQ - The Guide," [Online]. Available: https://zguide.zeromq.org/. [Accessed May 2022].

[29] Unity, "Unity Personal," 2022. [Online]. Available: https://store.unity.com/products/unity-personal. [Accessed 2022].

[30] M. Alba, "Intro to the Unity Industrial Collection, Part 1: Importing and Optimizing 3D Data," engineering.com, 26 Aug 2021. [Online]. Available: https://www.engineering.com/story/intro-to-the-unity-industrial-collection-part-1-importing-and-optimizing-3d-data. [Accessed May 2022].

[31] J. Faure, "Unity Reflect evolves into a suite of purpose-built applkications for AEC," Unity, 22 April 2021. [Online]. Available: https://blog.unity.com/aec/unity-reflect-evolves-into-a-suite-of-purpose-built-applications-for-aec. [Accessed May 2022].

[32] Microsoft, "Game Development with Visual Studio," 2022. [Online]. Available: https://visualstudio.microsoft.com/vs/features/game-development/. [Accessed May 2022].

[33] FreeCAD, "FreeCAD," 2022. [Online]. Available: https://www.freecadweb.org/. [Accessed May 2022].

[34] Team SimPy, "SimPy Discrete event simulation for Python: Overview," [Online]. Available: https://simpy.readthedocs.io/en/latest/. [Accessed May 2022].

[35] Wikipedia, "Matplotlib," [Online]. Available: https://en.wikipedia.org/wiki/Matplotlib. [Accessed May 2022].

[36] pandas development team, "pandas: package overview," 2022. [Online]. Available: https://pandas.pydata.org/docs/getting_started/overview.html. [Accessed May 2022].

[37] NumPy project, "NumPy," [Online]. Available: https://numpy.org/. [Accessed May 2022].

[38] Python Software Foundation, "tkinter - Python interface to Tcl/Tk," [Online]. Available: https://docs.python.org/3/library/tkinter.html. [Accessed May 2022].

[39] B. E. Granger and M. Ragan-Kelley, "PyZMQ Documentation," [Online]. Available: https://pyzmq.readthedocs.io/en/latest/. [Accessed May 2022].

[40] Valve, "SteamVR Unity Plugin," [Online]. Available: https://valvesoftware.github.io/steamvr_unity_plugin/. [Accessed May 2022].

[41] NetMQ, "NetMQ Documentation," [Online]. Available: https://netmq.readthedocs.io/en/latest/. [Accessed May 2022].

[42] N. S., "ZeroMQ in Unity," 29 September 2020. [Online]. Available: https://tech.uqido.com/2020/09/29/zeromq-in-unity/. [Accessed May 2022].

[43] P. Viala, *Presentation of the battery production process,* 2022.

[44] M. Ellis, "stack overflow: How can I use send_json with pyzmq PUB SUB - Answer," [Online]. Available: https://stackoverflow.com/a/25190798. [Accessed May 2022].

[45] M. Kaplan, "GitHubGist moshekaplan/TextHandler.py," [Online]. Available: https://gist.github.com/moshekaplan/c425f861de7bbf28ef06. [Accessed May 2022].

[46] Iris VR, "The Importance of Frame Rates," [Online]. Available: https://help.irisvr.com/hc/en-us/articles/215884547-The-Importance-of-Frame-Rates. [Accessed May 2022].

[47] V. Raulet, "Overview of VR headsets Technology in 2022," 2 Mars 2022. [Online]. Available: https://vraulet.medium.com/overview-of-vr-headsets-technology-in-2022-f586692c3d3d. [Accessed May 2022].

Appendices

Appendix A: FMH606 Master's Thesis: Project topic description
Appendix B: PlantFlowSim.py code
Appendix C: SimParameters.cfg file
Appendix D: SimControlAndGraph.py code
Appendix E: ZMQPub.py code
Appendix F: Client.cs code
Appendix G: Sender.cs code
Appendix H: MachineHandler.cs code
Appendix I: StorageHandler.cs code
Appendix J: PerformanceTest.cs
Appendix K: ZMQ_StressTest.py
Appendix L: PerfAnalysis.py

Appendix A: FMH606 Master's Thesis: Project topic description

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Digital Twin for monitoring, optimization and training in battery production

USN supervisor: Ole Magnus Brastein

External partner: Norwegian battery manufacturer represented by Pascal Viala

Task background:
A battery manufacturer in Norway with initial production starting already in 2022, intends to be the most environmentally friendly battery production and the most technologically advanced workplace globally. To achieve these goals, the manufacturer is in the process of developing a 1D and 3D digital twin to model and optimize all production processes.

Task description:
The objective of the thesis is to model parts of the battery production processes and visualize them in a 3D environment where the goal of the solution is to identify production bottlenecks and inefficiencies, optimize factory layout, train personnel and in the future the solution should be possible to reuse for factory monitoring.

The project consists of the following activities:

- Give an overview of research on combining data-driven models and 3D representations in digital twins, virtual reality (VR) solutions for digital twin presentation and factory optimization using digital twins
- Discuss best practice methods, and benefits of building digital twin solutions and methods that can be used to develop material flow model of a battery plant production line
- Implement a dynamic material flow model of parts of the battery plant in python, with the possibility to switch between simulated model-based values and actual process information
- Develop a VR navigational environment implemented in a Unity game engine solution using actual Building Information Modeling (BIM) files and other 3D models
- Integrate material flow model and VR solution using easy to configure network-based communication, to animate the process and overlay process information
- Evaluate the performance, versability and usability of the developed solution
- Discuss future changes that can be made to improve the solution, identify possible missing data/information that is needed to reach the goal of the solution

Student category: IIA students

Is the task suitable for online students (not present at the campus)? Yes

Practical arrangements:
The battery manufacturer can provide a workspace in Oslo, remote work is accepted. Necessary resources in terms of software, computing or mixed reality equipment will be provided.

Supervision:
As a general rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

Ole Magnus Brastein, 04.02.22

Signatures:
Supervisor (date and signature): Ole M. Brastein

Student (write clearly in all capitalized letters): RONNIE ANDRÉ HORNEMOE

Ronnie A. Moe 1/2-22

Appendix B: PlantFlowSim.py code

```
import simpy
import simpy.rt
import threading
import zmq
import json
import uuid
from numpy import random

from zmq.eventloop import ioloop
ioloop.install()

message_count = 0
connection_count = 0

start = False
exitflag = False

# initialize simulation environment
env = simpy.rt.RealtimeEnvironment(factor=1, strict=False)

storages = []
machines = []

def mogrify(topic, msg):
    """ json encode the message and prepend the topic """
    return topic + ' ' + json.dumps(msg)

def demogrify(topicmsg):
    """ Inverse of mogrify() """
    json0 = topicmsg.find('{')
    topic = topicmsg[0:json0].strip()
    msg = json.loads(topicmsg[json0:])
    return topic, msg

def getcommand(topic, msg):
    global exitflag, start
    print("Received: {topic} {message}".format(topic=topic, message=msg))
    if topic == "COMMAND":
        if msg['command'] == "Exit":
            print("Received exit command, client will stop receiving messages")
            exitflag = True
        elif msg['command'] == "Start":
            print("Received Start command, Simulation starts")
            start = True
    elif topic == "CHANGE":
        name = msg['name']
        if any(elem.name == name for elem in machines):
            machine = next(machine for machine in machines if machine.name == name)
            try:
                machine.container_chg = True
            except:
                print('Container change failed')
    else:
        print("Unknown topic recieved")

context = zmq.Context()
socketPub = context.socket(zmq.PUB)  # <- the PUBLISH socket
socketSub = context.socket(zmq.SUB)  # <- the SUBSCRIBE socket
socketSub.setsockopt_string(zmq.SUBSCRIBE, "C") # <- topic subscription

socketSub.bind("tcp://*:5556")
socketPub.bind('tcp://*:5555')

def sub_handler():
    while True:
        topic, msg = demogrify(socketSub.recv_string())
        getcommand(topic, msg)
```

```
class LoadParameters(object):
    def __init__(self, config_file):
        self.readConfigFile(config_file)

    def readConfigFile(self, file):
        try:
            with open(file, 'r') as cfg:
                lines = list(cfg)
            print("Read", len(lines), "lines from file", file)
            for line in lines:
                if not line == "\n" and not line.startswith("#"):
                    key, value = line.split('=')
                    setattr(self, key.strip(), eval(value.strip()))
        except:
            print('Configuration file read error')
            raise

def update_status(self, new_status):
    self.status = new_status
    if self.last_status != self.status:
        print(self.name, 'STATUS:', self.status)
        data = {
            'name': self.name,
            'status': self.status,
            'time': self.env.now

        }
        socketPub.send_string(mogrify("STATUS", data))

def update_storage(from_to, storage_name, level, direction):
    if direction == "from":
        print(from_to, " <- ", storage_name, ": ", level)
    else:
        print(from_to, " -> ", storage_name, ": ", level)
    data = {
        'from_to': from_to,
        'storage': storage_name,
        'level': level,
        'direction': direction,
        'time': env.now
    }
    socketPub.send_string(mogrify("STORAGE", data))

def end_process(env):
    global exitflag
    while not exitflag:
        yield env.timeout(1)
    exitflag = False

class Product:
    def __init__(self, env, name):
        self.env = env
        self.unique_id = uuid.uuid4()
        self.time_created = env.now
        self.name = name

class ContainerProduct(Product):
    def __init__(self, env, name, init_level, request=None, quality="PASS"):
        super().__init__(env, name)
        self.container = simpy.Container(
            env, init_level, init=init_level)
        self.request = request
        self.quality = quality

class UnitProduct(Product):
    def __init__(self, env, name, parts=[], calculate_quality=False, quality="PASS", passRate=1.0):
        super().__init__(env, name)
        self.parts = []
        self.quality = quality
```

```
        for part in parts:
            self.parts.append((part.name, part.unique_id, part.quality))
            if(part.quality == "FAIL"):
                self.quality == "FAIL"
        if self.quality == "PASS" and calculate_quality and not random.binomial(n=1, p=passRate,
size=1):
            self.quality = "FAIL"

class Machine:
    def __init__(self, env, name, MTBF, MTTR):
        self.env = env
        self.name = name
        self.action = env.process(self.run())
        self.MTBF = MTBF
        self.MTTR = MTTR
        self.TBF = random.normal(MTBF, MTBF/4)
        self.TTR = random.normal(MTTR, MTTR/2)
        self.start_time = env.now
        self.idle_time = 0
        self.status = "IDLE"
        self.last_status = "IDLE"

    def run(self):
        pass

    def idle(self):
        update_status(self, "IDLE")
        self.idle_time += Parameters.SIM_STEP
        yield env.timeout(Parameters.SIM_STEP)

class SingleRawMaterialMachine(Machine):
    def __init__(self, env, productType, productName, productStorage, PRODUCT_PER_CYCLE, cycle_time,
product_container_max, material1_name, MATERIAL1_USE_PR_CYCLE, material1_capacity, material1_storage,
CONTAINER_CHG_MIN, MTBF, MTTR, name="ROLLER", remote_change=False):
        super().__init__(env, name, MTBF, MTTR)
        self.productType = productType
        self.productName = productName
        self.productStorage = productStorage
        self.material1_name = material1_name
        self.material1_capacity = material1_capacity
        self.material1_container = simpy.Container(
            env, capacity=material1_capacity+MATERIAL1_USE_PR_CYCLE, init=0)
        self.material1_storage = material1_storage
        self.MATERIAL1_USE_PR_CYCLE = MATERIAL1_USE_PR_CYCLE
        self.PRODUCT_PER_CYCLE = PRODUCT_PER_CYCLE
        self.CONTAINER_CHG_MIN = CONTAINER_CHG_MIN
        self.remote_change = remote_change
        self.container_chg = False
        self.product_produced = simpy.Container(env, init=0)
        self.product_produced_total = 0
        self.product_container_max = product_container_max
        self.cycle_time = cycle_time

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif self.material1_container.level >= self.MATERIAL1_USE_PR_CYCLE and
(self.productStorage.capacity - len(self.productStorage.items) > 0):
                update_status(self, "RUN")
                self.material1_container.get(self.MATERIAL1_USE_PR_CYCLE)
                yield env.timeout(self.cycle_time)
                self.product_produced.put(self.PRODUCT_PER_CYCLE)
                self.product_produced_total = self.product_produced_total + self.PRODUCT_PER_CYCLE
```

```
                if self.product_produced.level >= self.product_container_max:
                    self.product_produced.get(self.product_container_max)
                    if(self.productType == ContainerProduct):
                        product = self.productType(
                            self.env, self.productName, self.product_container_max)
                    else:
                        product = self.productType(
                            self.env, self.productName)
                    yield self.productStorage.put(product)
                    update_storage(self.name, product.name, len(
                        self.productStorage.items), "to")
            elif self.material1_container.level < self.MATERIAL1_USE_PR_CYCLE and
self.material1_storage.level > 0:
                yield self.env.process(self.material1_change())
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

    def material1_change(self):
        update_status(self, "{material_name}_CONTAINER_CHANGE".format(
            material_name=self.material1_name))
        yield self.material1_storage.get(1)
        update_storage(self.name, self.material1_name,
                       self.material1_storage.level, "from")
        if self.remote_change:
            while not self.container_chg:
                yield env.timeout(1)
            self.container_chg = False
            yield self.material1_container.put(self.material1_capacity)
        else:
            yield env.timeout(self.CONTAINER_CHG_MIN)
            yield self.material1_container.put(self.material1_capacity)


class TwoRawMaterialMachine(SingleRawMaterialMachine):
    def __init__(self, env, productType, productName, productStorage, PRODUCT_PER_CYCLE, cycle_time,
product_container_max, material1_name, MATERIAL1_USE_PR_CYCLE, material1_capacity, material1_storage,
material2_name, MATERIAL2_USE_PR_CYCLE, material2_capacity, material2_storage, CONTAINER_CHG_MIN,
MTBF, MTTR, name="MILL", remote_change=False):
        super().__init__(env, productType, productName, productStorage, PRODUCT_PER_CYCLE, cycle_time,
product_container_max,
                         material1_name, MATERIAL1_USE_PR_CYCLE, material1_capacity,
material1_storage, CONTAINER_CHG_MIN, MTBF, MTTR, name, remote_change)
        self.material2_name = material2_name
        self.material2_capacity = material2_capacity
        self.material2_container = simpy.Container(
            env, capacity=material2_capacity+MATERIAL2_USE_PR_CYCLE, init=0)
        self.material2_storage = material2_storage
        self.MATERIAL2_USE_PR_CYCLE = MATERIAL2_USE_PR_CYCLE

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif self.material1_container.level >= self.MATERIAL1_USE_PR_CYCLE and
self.material2_container.level >= self.MATERIAL2_USE_PR_CYCLE and (self.productStorage.capacity -
len(self.productStorage.items) > 0):
                update_status(self, "RUN")
                self.material1_container.get(self.MATERIAL1_USE_PR_CYCLE)
                self.material2_container.get(self.MATERIAL2_USE_PR_CYCLE)
                yield env.timeout(self.cycle_time)
                self.product_produced.put(self.PRODUCT_PER_CYCLE)
                self.product_produced_total = self.product_produced_total + self.PRODUCT_PER_CYCLE
                if self.product_produced.level >= self.product_container_max:
```

```
                    self.product_produced.get(self.product_container_max)
                    product = self.productType(
                        self.env, self.productName, self.product_container_max)
                    yield self.productStorage.put(product)
                    update_storage(self.name, product.name, len(
                        self.productStorage.items), "to")
            elif self.material1_container.level < self.MATERIAL1_USE_PR_CYCLE and
self.material1_storage.level > 0:
                yield self.env.process(self.material1_change())
            elif self.material2_container.level < self.MATERIAL2_USE_PR_CYCLE and
self.material2_storage.level > 0:
                yield self.env.process(self.material2_change())
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

    def material2_change(self):
        update_status(self, "{material_name}_CONTAINER_CHANGE".format(
            material_name=self.material2_name))
        yield self.material2_storage.get(1)
        self.material2_container.put(self.material2_capacity)
        yield env.timeout(self.CONTAINER_CHG_MIN)
        update_storage(self.name, self.material2_name,
                       self.material2_storage.level, "from")

class CartridgeLoadMachine(SingleRawMaterialMachine):
    def __init__(self, env, productType, productName, productStorage, PRODUCT_PER_CYCLE, cycle_time,
product_container_max, material1_name, MATERIAL1_USE_PR_CYCLE, material1_capacity, material1_storage,
MATERIAL2_USE_PR_CYCLE, material2_storage, cartridge_storage, CONTAINER_CHG_MIN, MTBF, MTTR,
name="COMPRESS AND LOAD"):
        super().__init__(env, productType, productName, productStorage, PRODUCT_PER_CYCLE, cycle_time,
product_container_max,
                         material1_name, MATERIAL1_USE_PR_CYCLE, material1_capacity,
material1_storage, CONTAINER_CHG_MIN, MTBF, MTTR, name)
        self.material2 = []
        self.material2_storage = material2_storage
        self.MATERIAL2_USE_PR_CYCLE = MATERIAL2_USE_PR_CYCLE
        self.cartridge_storage = cartridge_storage

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif self.material1_container.level >= self.MATERIAL1_USE_PR_CYCLE and
isinstance(self.material2, ContainerProduct) and self.material2.container.level >=
self.MATERIAL2_USE_PR_CYCLE and (self.productStorage.capacity - len(self.productStorage.items) > 0):
                update_status(self, "RUN")
                self.material1_container.get(self.MATERIAL1_USE_PR_CYCLE)
                yield self.material2.container.get(self.MATERIAL2_USE_PR_CYCLE)
                yield env.timeout(self.cycle_time)
                self.product_produced.put(self.PRODUCT_PER_CYCLE)
                self.product_produced_total = self.product_produced_total + self.PRODUCT_PER_CYCLE
                if self.product_produced.level >= self.product_container_max:
                    self.product_produced.get(self.product_container_max)
                    request = self.cartridge_storage.request()
                    yield request
                    product = self.productType(
                        self.env, self.productName, self.product_container_max, request=request)
                    yield self.productStorage.put(product)
                    update_storage(self.name, product.name, len(
                        self.productStorage.items), "to")
            elif self.material1_container.level < self.MATERIAL1_USE_PR_CYCLE and
self.material1_storage.level > 0:
                yield self.env.process(self.material1_change())
```

```
            elif not (isinstance(self.material2, ContainerProduct) and self.material2.container.level
>= self.MATERIAL2_USE_PR_CYCLE) and len(self.material2_storage.items) > 0:
                yield self.env.process(self.material2_change())
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

    def material2_change(self):
        self.material2 = self.material2_storage.get().value
        update_status(self, "{material_name}_CONTAINER_CHANGE".format(
            material_name=self.material2.name))
        yield env.timeout(self.CONTAINER_CHG_MIN)
        update_storage(self.name, self.material2.name, len(
            self.material2_storage.items), "from")

class SingleProductRefineMachine(Machine):
    def __init__(self, env, productType, productName, productStorage, PRODUCT_PER_CYCLE, cycle_time,
product_container_max, MATERIAL1_USE_PR_CYCLE, material1_storage, CONTAINER_CHG_MIN, MTBF, MTTR,
name="ROLLER", cartridge_storage=None):
        super().__init__(env, name, MTBF, MTTR)
        self.productType = productType
        self.productName = productName
        self.productStorage = productStorage
        self.material1 = []
        self.material1_storage = material1_storage
        self.MATERIAL1_USE_PR_CYCLE = MATERIAL1_USE_PR_CYCLE
        self.PRODUCT_PER_CYCLE = PRODUCT_PER_CYCLE
        self.CONTAINER_CHG_MIN = CONTAINER_CHG_MIN
        self.product_produced = simpy.Container(env, init=0)
        self.product_produced_total = 0
        self.product_container_max = product_container_max
        self.cartridge_storage = cartridge_storage
        self.cycle_time = cycle_time

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif isinstance(self.material1, ContainerProduct) and self.material1.container.level >=
self.MATERIAL1_USE_PR_CYCLE and (self.productStorage.capacity - len(self.productStorage.items) > 0):
                update_status(self, "RUN")
                yield self.material1.container.get(self.MATERIAL1_USE_PR_CYCLE)
                yield env.timeout(self.cycle_time)
                self.product_produced.put(self.PRODUCT_PER_CYCLE)
                self.product_produced_total = self.product_produced_total + self.PRODUCT_PER_CYCLE
                if self.product_produced.level >= self.product_container_max:
                    self.product_produced.get(self.product_container_max)
                    product = self.productType(
                        self.env, self.productName, self.product_container_max)
                    yield self.productStorage.put(product)
                    update_storage(self.name, product.name, len(
                        self.productStorage.items), "to")
            elif not (isinstance(self.material1, ContainerProduct) and self.material1.container.level
>= self.MATERIAL1_USE_PR_CYCLE) and len(self.material1_storage.items) > 0:
                yield self.env.process(self.material1_change())
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

    def material1_change(self):
        if(isinstance(self.cartridge_storage, simpy.Resource) and isinstance(self.material1,
ContainerProduct) and self.material1.request != None):
            self.cartridge_storage.release(self.material1.request)
        self.material1 = self.material1_storage.get().value
```

```
        update_status(self, "{material_name}_CONTAINER_CHANGE".format(
            material_name=self.material1.name))
        yield env.timeout(self.CONTAINER_CHG_MIN)
        update_storage(self.name, self.material1.name, len(
            self.material1_storage.items), "from")

class TwoProductRefineMachine(SingleProductRefineMachine):
    def __init__(self, env, productType, productName, productStorage, PRODUCT_PER_CYCLE, cycle_time,
product_container_max, MATERIAL1_USE_PR_CYCLE, material1_storage, MATERIAL2_USE_PR_CYCLE,
material2_storage, CONTAINER_CHG_MIN, MTBF, MTTR, name="ASSEMBLY", cartridge_storage=None):
        super().__init__(env, productType, productName, productStorage, PRODUCT_PER_CYCLE, cycle_time,
product_container_max,
                         MATERIAL1_USE_PR_CYCLE, material1_storage, CONTAINER_CHG_MIN, MTBF, MTTR,
name, cartridge_storage)
        self.material2 = []
        self.material2_storage = material2_storage
        self.MATERIAL2_USE_PR_CYCLE = MATERIAL2_USE_PR_CYCLE

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif isinstance(self.material1, ContainerProduct) and self.material1.container.level >=
self.MATERIAL1_USE_PR_CYCLE and isinstance(self.material2, ContainerProduct) and
self.material2.container.level >= self.MATERIAL2_USE_PR_CYCLE and (self.productStorage.capacity -
len(self.productStorage.items) > 0):
                update_status(self, "RUN")
                self.material1.container.get(self.MATERIAL1_USE_PR_CYCLE)
                self.material2.container.get(self.MATERIAL2_USE_PR_CYCLE)
                yield env.timeout(self.cycle_time)
                self.product_produced.put(self.PRODUCT_PER_CYCLE)
                self.product_produced_total = self.product_produced_total + self.PRODUCT_PER_CYCLE
                if self.product_produced.level >= self.product_container_max:
                    self.product_produced.get(self.product_container_max)
                    product = self.productType(
                        self.env, self.productName, parts=[self.material1, self.material2])
                    yield self.productStorage.put(product)
                    update_storage(self.name, product.name, len(
                        self.productStorage.items), "to")
            elif not (isinstance(self.material1, ContainerProduct) and self.material1.container.level
>= self.MATERIAL1_USE_PR_CYCLE) and len(self.material1_storage.items) > 0:
                yield self.env.process(self.material1_change())
            elif not (isinstance(self.material2, ContainerProduct) and self.material2.container.level
>= self.MATERIAL2_USE_PR_CYCLE) and len(self.material2_storage.items) > 0:
                yield self.env.process(self.material2_change())
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

    def material2_change(self):
        self.material2 = self.material2_storage.get().value
        update_status(self, "{material_name}_CONTAINER_CHANGE".format(
            material_name=self.material2.name))
        yield env.timeout(self.CONTAINER_CHG_MIN)
        update_storage(self.name, self.material2.name, len(
            self.material2_storage.items), "from")

class AssembleMachine(SingleProductRefineMachine):
    def __init__(self, env, productType, productName, productStorage, scrap_storage,
PRODUCT_PER_CYCLE, cycle_time, product_container_max, MATERIAL1_USE_PR_CYCLE, material1_storage,
material2_storage, material3_storage, CONTAINER_CHG_MIN, MTBF, MTTR, name="ASSEMBLY"):
        super().__init__(env, productType, productName, productStorage, PRODUCT_PER_CYCLE, cycle_time,
product_container_max,
```

```
                        MATERIAL1_USE_PR_CYCLE, material1_storage, CONTAINER_CHG_MIN, MTBF, MTTR,
name)
        self.scrap_storage = scrap_storage
        self.material2_storage = material2_storage
        self.material3_storage = material3_storage

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif isinstance(self.material1, ContainerProduct) and self.material1.container.level >=
self.MATERIAL1_USE_PR_CYCLE and len(self.material2_storage.items) > 0 and
len(self.material3_storage.items) > 0 and (self.productStorage.capacity -
len(self.productStorage.items) > 0):
                update_status(self, "RUN")
                self.material1.container.get(self.MATERIAL1_USE_PR_CYCLE)
                material2 = self.material2_storage.get().value
                update_storage(self.name, material2.name, len(
                    self.material2_storage.items), "from")
                material3 = self.material3_storage.get().value
                update_storage(self.name, material3.name, len(
                    self.material3_storage.items), "from")
                yield env.timeout(self.cycle_time)
                self.product_produced.put(self.PRODUCT_PER_CYCLE)
                self.product_produced_total = self.product_produced_total + self.PRODUCT_PER_CYCLE
                if self.product_produced.level >= self.product_container_max:
                    self.product_produced.get(self.product_container_max)
                    product = self.productType(
                        self.env, self.productName, parts=[self.material1, material2, material3])
                    if(isinstance(self.scrap_storage, simpy.Store) and product.quality == "FAIL"):
                        yield self.scrap_storage.put(product)
                        update_storage(
                            self.name, product.name + "_scrap", len(self.scrap_storage.items), "to")
                    else:
                        yield self.productStorage.put(product)
                        update_storage(self.name, product.name, len(
                            self.productStorage.items), "to")
            elif not (isinstance(self.material1, ContainerProduct) and self.material1.container.level
>= self.MATERIAL1_USE_PR_CYCLE) and len(self.material1_storage.items) > 0:
                yield self.env.process(self.material1_change())
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

class StackMachine(Machine):
    def __init__(self, env, productType, productName, productStorage, scrap_storage, cycle_time,
stack_amount, material1_storage, MTBF, MTTR, name="STACKER"):
        super().__init__(env, name, MTBF, MTTR)
        self.productType = productType
        self.productName = productName
        self.productStorage = productStorage
        self.scrap_storage = scrap_storage
        self.stack_amount = stack_amount
        self.material1_storage = material1_storage
        self.product_produced_total = 0
        self.cycle_time = cycle_time

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
```

```
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif len(self.material1_storage.items) >= self.stack_amount and
(self.productStorage.capacity - len(self.productStorage.items) > 0):
                update_status(self, "RUN")
                parts = []
                for i in range(self.stack_amount):
                    parts.append(self.material1_storage.get().value)
                update_storage(self.name, parts[0].name, len(
                    self.material1_storage.items), "from")
                yield env.timeout(self.cycle_time)
                self.product_produced_total = self.product_produced_total + 1
                product = self.productType(
                    self.env, self.productName, parts=parts)
                if(isinstance(self.scrap_storage, simpy.Store) and product.quality == "FAIL"):
                    yield self.scrap_storage.put(product)
                    update_storage(self.name, product.name + "_scrap",
                                   len(self.scrap_storage.items), "to")
                else:
                    yield self.productStorage.put(product)
                    update_storage(self.name, product.name, len(
                        self.productStorage.items), "to")
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

class TabMachine(Machine):
    def __init__(self, env, productType, productName, productStorage, scrap_storage, cycle_time,
unit_storage, tab1_storage, tab2_storage, MTBF, MTTR, name="TAB"):
        super().__init__(env, name, MTBF, MTTR)
        self.productType = productType
        self.productName = productName
        self.productStorage = productStorage
        self.scrap_storage = scrap_storage
        self.unit_storage = unit_storage
        self.tab1_storage = tab1_storage
        self.tab2_storage = tab2_storage
        self.product_produced_total = 0
        self.cycle_time = cycle_time

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif len(self.unit_storage.items) > 0 and self.tab1_storage.level > 0 and
self.tab2_storage.level > 0 and (self.productStorage.capacity - len(self.productStorage.items) > 0):
                update_status(self, "RUN")
                unit = self.unit_storage.get().value
                yield self.tab1_storage.get(1)
                update_storage(self.name, "ANTAB",
                               self.tab1_storage.level, "from")
                yield self.tab2_storage.get(1)
                update_storage(self.name, "CATAB",
                               self.tab2_storage.level, "from")
                yield env.timeout(self.cycle_time)
                self.product_produced_total = self.product_produced_total + 1
                product = self.productType(
                    self.env, self.productName, parts=[unit])
                if(isinstance(self.scrap_storage, simpy.Store) and product.quality == "FAIL"):
                    yield self.scrap_storage.put(product)
                    update_storage(self.name, product.name+"_scrap",
                                   len(self.scrap_storage.items), "to")
                else:
```

```
                    yield self.productStorage.put(product)
                    update_storage(self.name, product.name, len(
                        self.productStorage.items), "to")
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

class CellUnitMachine(Machine):
    def __init__(self, env, productType, productName, productStorage, scrap_storage, cycle_time,
unit_storage, pouch_storage, tape_storage, MTBF, MTTR, name="ASSEMBLY"):
        super().__init__(env, name, MTBF, MTTR)
        self.productType = productType
        self.productName = productName
        self.productStorage = productStorage
        self.scrap_storage = scrap_storage
        self.unit_storage = unit_storage
        self.pouch_storage = pouch_storage
        self.tape_storage = tape_storage
        self.product_produced_total = 0
        self.cycle_time = cycle_time

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif len(self.unit_storage.items) > 0 and len(self.pouch_storage.items) > 0 and
len(self.tape_storage.items) > 0 and (self.productStorage.capacity - len(self.productStorage.items) >
0):
                update_status(self, "RUN")
                unit = self.unit_storage.get().value
                update_storage(self.name, unit.name, len(
                    self.unit_storage.items), "from")
                pouch = self.pouch_storage.get().value
                update_storage(self.name, pouch.name, len(
                    self.pouch_storage.items), "from")
                tape = self.tape_storage.get().value
                update_storage(self.name, tape.name, len(
                    self.tape_storage.items), "from")
                yield env.timeout(self.cycle_time)
                self.product_produced_total = self.product_produced_total + 1
                product = self.productType(
                    self.env, self.productName, parts=[unit, pouch, tape])
                if(isinstance(self.scrap_storage, simpy.Store) and product.quality == "FAIL"):
                    yield self.scrap_storage.put(product)
                    update_storage(self.name, product.name+"_scrap",
                                   len(self.scrap_storage.items), "to")
                else:
                    yield self.productStorage.put(product)
                    update_storage(self.name, product.name, len(
                        self.productStorage.items), "to")
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

class SingleUnitProcesser(Machine):
    def __init__(self, env, productType, productName, productStorage, scrap_storage, pass_rate,
cycle_time, unit_storage, MTBF, MTTR, name="PROCESSER"):
        super().__init__(env, name, MTBF, MTTR)
        self.productType = productType
        self.productName = productName
        self.productStorage = productStorage
        self.scrap_storage = scrap_storage
        self.pass_rate = pass_rate
        self.unit_storage = unit_storage
```

```
        self.product_produced_total = 0
        self.cycle_time = cycle_time

    def run(self):
        while True:
            # Downtime
            if env.now >= self.start_time + self.TBF + self.idle_time:
                update_status(self, "DOWN")
                yield env.timeout(self.TTR)
                self.start_time = env.now
                self.idle_time = 0
                self.TBF = random.normal(self.MTBF, self.MTBF/4)
                self.TTR = random.normal(self.MTTR, self.MTTR/2)
            elif len(self.unit_storage.items) > 0 and (self.productStorage.capacity -
len(self.productStorage.items) > 0):
                update_status(self, "RUN")
                unit = self.unit_storage.get().value
                update_storage(self.name, unit.name, len(
                    self.unit_storage.items), "from")
                yield env.timeout(self.cycle_time)
                self.product_produced_total = self.product_produced_total + 1
                product = self.productType(
                    self.env, self.productName, parts=[unit], calculate_quality=self.pass_rate)
                if(isinstance(self.scrap_storage, simpy.Store) and product.quality == "FAIL"):
                    yield self.scrap_storage.put(product)
                    update_storage(self.name, product.name+"_scrap",
                                   len(self.scrap_storage.items), "to")
                else:
                    yield self.productStorage.put(product)
                    update_storage(self.name, product.name, len(
                        self.productStorage.items), "to")
            else:
                yield self.env.process(self.idle())
            self.last_status = self.status

class RawMaterialRefill:
    def __init__(self, env, storage, reorder_amount, reorder_min, delivery_time_min, name="STUFF"):
        self.env = env
        self.action = env.process(self.run())
        self.storage = storage
        self.name = name
        self.reorder_amount = reorder_amount
        self.reorder_min = reorder_min
        self.delivery_time_min = delivery_time_min
        self.status = "IDLE"

    def run(self):
        while True:
            # Downtime
            if isinstance(self.storage, simpy.Container) and self.storage.level <= self.reorder_min:
                self.status = "REFILL ORDERED"
                print(self.name, 'ORDERED')
                yield env.timeout(self.delivery_time_min)
                self.status = "ORDER ARRIVED"
                print(self.name, 'ARRIVED')
                yield self.storage.put(self.reorder_amount)
                update_storage("Supply", self.name, self.storage.level, "to")
            else:
                yield self.env.process(self.idle())

    def idle(self):
        self.status = "IDLE"
        yield env.timeout(10*Parameters.SIM_STEP)

def SetupStorages():
    global hd01_storage, c45_storage, electrolyte_storage, anode_foil_roll_storage,
carrier_film_roll_storage, posconmc_storage, cond_carbon_storage, cathode_foil_roll_storage,
seperator_roll_storage, insulation_tape_roll_storage, aluminized_pouch_roll_storage,
anode_tab_storage, cathode_tab_storage
    global clean_empty_cartridges
```

```
    global anode_blend_storage, anode_slurry_cartridge_storage, anode_laminate_roll_storage,
anode_storage, cathode_blend_storage, dried_cathode_blend_storage, cathode_slurry_cartridge_storage,
cathode_laminate_roll_storage, cathode_storage, seperator_roll_shutdown_frame_storage,
unit_cell_storage, unit_cell_stack_storage, insulation_tape_roll_storage, insulation_tape_storage,
formed_pouch_storage, unit_cell_stack_tabbed_storage, cell_preformation_storage
    global scrap_cell_storage, scrap_cell_preformation_storage, scrap_unit_cell_stack_tabbed_storage,
scrap_unit_cell_storage
    global finished_cell_storage, cell_box_storage, pallet_of_cell_boxes_storage

    # initialize raw material storages
    hd01_storage = simpy.Container(
        env, capacity=Parameters.HD01_CAP, init=Parameters.HD01_INIT)
    storages.append(hd01_storage)
    c45_storage = simpy.Container(
        env, capacity=Parameters.C45_CAP, init=Parameters.C45_INIT)
    storages.append(c45_storage)
    electrolyte_storage = simpy.Container(
        env, capacity=Parameters.ELEC_CAP, init=Parameters.ELEC_INIT)
    storages.append(electrolyte_storage)
    anode_foil_roll_storage = simpy.Container(
        env, capacity=Parameters.ANFOILROLL_CAP, init=Parameters.ANFOILROLL_INIT)
    storages.append(anode_foil_roll_storage)
    carrier_film_roll_storage = simpy.Container(
        env, capacity=Parameters.CARFILMROLL_CAP, init=Parameters.CARFILMROLL_INIT)
    storages.append(carrier_film_roll_storage)
    posconmc_storage = simpy.Container(
        env, capacity=Parameters.POSCONMC_CAP, init=Parameters.POSCONMC_INIT)
    storages.append(posconmc_storage)
    cond_carbon_storage = simpy.Container(
        env, capacity=Parameters.COND_CARB_CAP, init=Parameters.COND_CARB_INIT)
    storages.append(cond_carbon_storage)
    cathode_foil_roll_storage = simpy.Container(
        env, capacity=Parameters.CAFOILROLL_CAP, init=Parameters.CAFOILROLL_INIT)
    storages.append(cathode_foil_roll_storage)
    seperator_roll_storage = simpy.Container(
        env, capacity=Parameters.SEPROLL_CAP, init=Parameters.SEPROLL_INIT)
    storages.append(seperator_roll_storage)
    insulation_tape_roll_storage = simpy.Container(
        env, capacity=Parameters.INSULTAPEROLL_CAP, init=Parameters.INSULTAPEROLL_INIT)
    storages.append(insulation_tape_roll_storage)
    aluminized_pouch_roll_storage = simpy.Container(
        env, capacity=Parameters.ALPOUCHROLL_CAP, init=Parameters.ALPOUCHROLL_INIT)
    storages.append(aluminized_pouch_roll_storage)
    anode_tab_storage = simpy.Container(
        env, capacity=Parameters.ANTAB_CAP, init=Parameters.ANTAB_INIT)
    storages.append(anode_tab_storage)
    cathode_tab_storage = simpy.Container(
        env, capacity=Parameters.CATAB_CAP, init=Parameters.CATAB_INIT)
    storages.append(cathode_tab_storage)

    # initialize limited resources
    clean_empty_cartridges = simpy.Resource(
        env, capacity=Parameters.CARTRIDGE_CAP)

    # initialize half fabricated storages
    anode_blend_storage = simpy.Store(env, capacity=Parameters.ANBL_CAP)
    storages.append(anode_blend_storage)
    anode_slurry_cartridge_storage = simpy.Store(
        env, capacity=Parameters.ANSLCART_CAP)
    storages.append(anode_slurry_cartridge_storage)
    anode_laminate_roll_storage = simpy.Store(
        env, capacity=Parameters.ANLAMROLL_CAP)
    storages.append(anode_laminate_roll_storage)
    anode_storage = simpy.Store(env, capacity=Parameters.AN_CAP)
    storages.append(anode_storage)
    cathode_blend_storage = simpy.Store(env, capacity=Parameters.CABL_CAP)
    storages.append(cathode_blend_storage)
    dried_cathode_blend_storage = simpy.Store(
        env, capacity=Parameters.CABLDR_CAP)
    storages.append(dried_cathode_blend_storage)
```

```
    cathode_slurry_cartridge_storage = simpy.Store(
        env, capacity=Parameters.CASLCART_CAP)
    storages.append(cathode_slurry_cartridge_storage)
    cathode_laminate_roll_storage = simpy.Store(
        env, capacity=Parameters.CALAMROLL_CAP)
    storages.append(cathode_laminate_roll_storage)
    cathode_storage = simpy.Store(env, capacity=Parameters.CA_CAP)
    storages.append(cathode_storage)
    seperator_roll_shutdown_frame_storage = simpy.Store(
        env, capacity=Parameters.SEPSHROLL_CAP)
    storages.append(seperator_roll_shutdown_frame_storage)
    unit_cell_storage = simpy.Store(env, capacity=Parameters.UNCE_CAP)
    storages.append(unit_cell_storage)
    unit_cell_stack_storage = simpy.Store(env, capacity=Parameters.UNCEST_CAP)
    storages.append(unit_cell_stack_storage)
    insulation_tape_storage = simpy.Store(
        env, capacity=Parameters.INSULTAPE_CAP)
    storages.append(insulation_tape_storage)
    formed_pouch_storage = simpy.Store(env, capacity=Parameters.FOPOUCH_CAP)
    storages.append(formed_pouch_storage)
    unit_cell_stack_tabbed_storage = simpy.Store(
        env, capacity=Parameters.UNCESTTAB_CAP)
    storages.append(unit_cell_stack_tabbed_storage)
    cell_preformation_storage = simpy.Store(env, capacity=Parameters.CEPRE_CAP)
    storages.append(cell_preformation_storage)

    # initialize scrap storages
    scrap_unit_cell_storage = simpy.Store(env)
    storages.append(scrap_unit_cell_storage)
    scrap_unit_cell_stack_tabbed_storage = simpy.Store(env)
    storages.append(scrap_unit_cell_stack_tabbed_storage)
    scrap_cell_preformation_storage = simpy.Store(env)
    storages.append(scrap_cell_preformation_storage)
    scrap_cell_storage = simpy.Store(env)
    storages.append(scrap_cell_storage)

    # initialize end products storages
    finished_cell_storage = simpy.Store(env, capacity=Parameters.FINCE_CAP)
    storages.append(finished_cell_storage)
    cell_box_storage = simpy.Store(env, capacity=Parameters.CEBOX_CAP)
    storages.append(cell_box_storage)
    pallet_of_cell_boxes_storage = simpy.Store(
        env, capacity=Parameters.PALLET_CAP)
    storages.append(pallet_of_cell_boxes_storage)

def SetupMachines():
    # Production equipment
    anode_powder_mill = TwoRawMaterialMachine(env, ContainerProduct, "ANBL", anode_blend_storage,
Parameters.ANODE_POWDER_MILLED_PER_CYCLE, Parameters.ANODE_POWDER_CYCLE,
                                              Parameters.POWDER_CONTAINER_MAX, "HD01",
Parameters.HD01_USE_PR_CYCLE, 300, hd01_storage, "C45", Parameters.C45_USE_PR_CYCLE, 200, c45_storage,
Parameters.CONTAINER_CHG, Parameters.MTBF_MILL, Parameters.MTTR_MILL, "ANMI", True)
    machines.append(anode_powder_mill)
    cathode_powder_mill = TwoRawMaterialMachine(env, ContainerProduct, "CABL", cathode_blend_storage,
Parameters.CATHODE_POWDER_MILLED_PER_CYCLE, Parameters.CATHODE_POWDER_CYCLE,
                                                Parameters.POWDER_CONTAINER_MAX, "PONMC",
Parameters.POSCONMC_USE_PR_CYCLE, 200, posconmc_storage, "COCA", Parameters.COND_CARBON_USE_PR_CYCLE,
100, cond_carbon_storage, Parameters.CONTAINER_CHG, Parameters.MTBF_MILL, Parameters.MTTR_MILL,
"CAMI")
    machines.append((cathode_powder_mill))
    cathode_powder_drier = SingleProductRefineMachine(env, ContainerProduct, "CABLDR",
                                                      dried_cathode_blend_storage,
Parameters.CATHODE_POWDER_DRIED_PER_CYCLE, Parameters.CATHODE_POWDER_DRYING_CYCLE,
Parameters.POWDER_CONTAINER_MAX, Parameters.CATHODE_POWDER_USE_PR_CYCLE, cathode_blend_storage,
Parameters.CONTAINER_CHG, Parameters.MTBF_DRYER, Parameters.MTTR_DRYER, "CABLDRY")
    machines.append(cathode_powder_drier)
    anode_cartridge_compress_and_load = CartridgeLoadMachine(env, ContainerProduct, "ANSLCART",
anode_slurry_cartridge_storage, Parameters.SLURRY_COMPRESS_PER_CYCLE, Parameters.ANODE_SLURRY_CYCLE,
Parameters.ANODE_BRICK, "ELEC",
```

```
                                                        Parameters.ELECTROLYTE_USE_PR_CYCLE, 100,
electrolyte_storage, Parameters.ANODE_POWDER_USE_PR_CYCLE, anode_blend_storage,
clean_empty_cartridges, Parameters.CART_CONTAINER_CHG_TIME, Parameters.MTBF_CART,
Parameters.MTTR_CART, name="ANSLCO")
    machines.append(anode_cartridge_compress_and_load)
    cathode_cartridge_compress_and_load = CartridgeLoadMachine(env, ContainerProduct, "CATSLCART",
cathode_slurry_cartridge_storage, Parameters.SLURRY_COMPRESS_PER_CYCLE,
Parameters.CATHODE_SLURRY_CYCLE, Parameters.CATHODE_BRICK, "ELEC",
                                                            Parameters.ELECTROLYTE_USE_PR_CYCLE,
100, electrolyte_storage, Parameters.CATHODE_DRIED_USE_PR_CYCLE, dried_cathode_blend_storage,
clean_empty_cartridges, Parameters.CART_CONTAINER_CHG_TIME, Parameters.MTBF_CART,
Parameters.MTTR_CART, name="CASLCO")
    machines.append(cathode_cartridge_compress_and_load)
    anode_foil_cut_and_laminate = TwoRawMaterialMachine(env, ContainerProduct, "ANLAMRO",
anode_laminate_roll_storage, Parameters.FOIL_PER_CYCLE, Parameters.FOIL_LAMINATE_CYCLE,
Parameters.FOIL_ROLL_MAX, "ANFORO", 1,
                                                        Parameters.FOIL_ROLL_MAX,
anode_foil_roll_storage, "CARFIRO", 1, Parameters.FOIL_ROLL_MAX, carrier_film_roll_storage,
Parameters.CONTAINER_CHG, Parameters.MTBF_LAMINATE, Parameters.MTTR_LAMINATE, name="ANFOLA")
    machines.append(anode_foil_cut_and_laminate)
    cathode_foil_cut_and_laminate = TwoRawMaterialMachine(env, ContainerProduct, "CALAMRO",
cathode_laminate_roll_storage, Parameters.FOIL_PER_CYCLE, Parameters.FOIL_LAMINATE_CYCLE,
Parameters.FOIL_ROLL_MAX, "CAFORO", 1,
                                                          Parameters.FOIL_ROLL_MAX,
cathode_foil_roll_storage, "CARFIRO", 1, Parameters.FOIL_ROLL_MAX, carrier_film_roll_storage,
Parameters.CONTAINER_CHG, Parameters.MTBF_LAMINATE, Parameters.MTTR_LAMINATE, name="CAFOLA")
    machines.append(cathode_foil_cut_and_laminate)
    anode_casting = TwoProductRefineMachine(env, UnitProduct, "ANEL", anode_storage, 1,
Parameters.CAST_CYCLE, 1, Parameters.ANODE_SLURRY_USE_PR_CYCLE,
                                            anode_slurry_cartridge_storage, 1,
anode_laminate_roll_storage, Parameters.CONTAINER_CHG, Parameters.MTBF_CAST, Parameters.MTTR_CAST,
name="ANCA", cartridge_storage=clean_empty_cartridges)
    machines.append(anode_casting)
    cathode_casting = TwoProductRefineMachine(env, UnitProduct, "CAEL", cathode_storage, 1,
Parameters.CAST_CYCLE, 1, Parameters.CATHODE_SLURRY_USE_PR_CYCLE,
                                              cathode_slurry_cartridge_storage, 1,
cathode_laminate_roll_storage, Parameters.CONTAINER_CHG, Parameters.MTBF_CAST, Parameters.MTTR_CAST,
name="CACA", cartridge_storage=clean_empty_cartridges)
    machines.append(cathode_casting)
    seperator_roll_shutdownframer = SingleRawMaterialMachine(env, ContainerProduct, "SEPSHRO",
seperator_roll_shutdown_frame_storage, 1, Parameters.FOIL_PER_CYCLE, Parameters.FOIL_ROLL_MAX,
                                                             "SEPRO", 1, Parameters.FOIL_ROLL_MAX,
seperator_roll_storage, Parameters.CONTAINER_CHG, Parameters.MTBF_SEPERATOR,
Parameters.MTTR_SEPERATOR, "SEPSHFR")
    machines.append(seperator_roll_shutdownframer)
    unit_cell_assembler = AssembleMachine(env, UnitProduct, "UNCE", unit_cell_storage,
scrap_cell_storage, 1, Parameters.ASSEMBLE_CYCLE, 1, 1, seperator_roll_shutdown_frame_storage,
                                          anode_storage, cathode_storage, Parameters.CONTAINER_CHG,
Parameters.MTBF_ASSEMBLE, Parameters.MTTR_ASSEMBLE, "UNCEAS")
    machines.append(unit_cell_assembler)
    unit_cell_stacker = StackMachine(env, UnitProduct, "UNCEST", unit_cell_stack_storage, None,
Parameters.STACK_CYCLE,
                                     Parameters.CELL_STACK_MAX, unit_cell_storage,
Parameters.MTBF_STACK, Parameters.MTTR_STACK, "UNCESK")
    machines.append(unit_cell_stacker)
    unit_cell_stack_tab_weld = TabMachine(env, UnitProduct, "UNCESTTAB",
unit_cell_stack_tabbed_storage, scrap_unit_cell_stack_tabbed_storage,
                                          Parameters.TAB_CYCLE, unit_cell_stack_storage,
anode_tab_storage, cathode_tab_storage, Parameters.MTBF_TAB, Parameters.MTTR_TAB, "UNCESTWE")
    machines.append(unit_cell_stack_tab_weld)
    insulation_tape_proc = SingleRawMaterialMachine(env, UnitProduct, "INTA", insulation_tape_storage,
1, Parameters.INSULATION_TAPE_CYCLE, 1, "INTARO",
                                                    1, Parameters.FOIL_ROLL_MAX,
insulation_tape_roll_storage, Parameters.CONTAINER_CHG, Parameters.MTBF_INSULATION,
Parameters.MTTR_INSULATION, "INTAPR")
    machines.append(insulation_tape_proc)
    formed_pouch_proc = SingleRawMaterialMachine(env, UnitProduct, "FOPO", formed_pouch_storage, 1,
Parameters.FORMED_POUCH_CYCLE, 1, "ALPORO",
```

```
                                                1, Parameters.FOIL_ROLL_MAX,
aluminized_pouch_roll_storage, Parameters.CONTAINER_CHG, Parameters.MTBF_POUCH, Parameters.MTTR_POUCH,
"POFOPR")
    machines.append(formed_pouch_proc)
    cell_assembly_proc = CellUnitMachine(env, UnitProduct, "CEPRE", cell_preformation_storage,
scrap_cell_preformation_storage, Parameters.CELL_ASSEMBLY_CYCLE, unit_cell_stack_tabbed_storage,
formed_pouch_storage, insulation_tape_storage, Parameters.MTBF_CELL_ASSEMBLY,
Parameters.MTTR_CELL_ASSEMBLY, name="CEAS")
    machines.append(cell_assembly_proc)

    mach1 = next(filter(lambda m: m.name == "UNCEAS", machines), None)

def SetupReorderProcesses():
    # Restocking of raw materials
    hd01_refill = RawMaterialRefill(env, hd01_storage, Parameters.HD01_ORDER_AMOUNT,
                                    Parameters.HD01_MIN_AMOUNT, Parameters.HD01_DELIVER_TIME, "HD01")
    c45_refill = RawMaterialRefill(env, c45_storage, Parameters.C45_ORDER_AMOUNT,
                                   Parameters.C45_MIN_AMOUNT, Parameters.C45_DELIVER_TIME, "C45")
    posconmc_refill = RawMaterialRefill(env, posconmc_storage, Parameters.POSCONMC_ORDER_AMOUNT,
                                        Parameters.POSCONMC_MIN_AMOUNT,
Parameters.POSCONMC_DELIVER_TIME, "PONMC")
    cond_carbon_refill = RawMaterialRefill(env, cond_carbon_storage,
Parameters.COND_CARB_ORDER_AMOUNT,
                                           Parameters.COND_CARB_MIN_AMOUNT,
Parameters.COND_CARB_DELIVER_TIME, "COCA")
    electrolyte_refill = RawMaterialRefill(env, electrolyte_storage, Parameters.ELEC_ORDER_AMOUNT,
                                           Parameters.ELEC_MIN_AMOUNT, Parameters.ELEC_DELIVER_TIME,
"ELEC")
    carrier_film_refill = RawMaterialRefill(env, carrier_film_roll_storage,
Parameters.CARRIER_FILM_ORDER_AMOUNT,
                                            Parameters.CARRIER_FILM_MIN_AMOUNT,
Parameters.CARRIER_FILM_DELIVER_TIME, "CARFIRO")
    anode_foil_roll_refill = RawMaterialRefill(env, anode_foil_roll_storage,
Parameters.FOIL_ROLL_ORDER_AMOUNT,
                                               Parameters.FOIL_ROLL_MIN_AMOUNT,
Parameters.FOIL_ROLL_DELIVER_TIME, "ANFORO")
    cathode_foil_roll_refill = RawMaterialRefill(env, cathode_foil_roll_storage,
Parameters.FOIL_ROLL_ORDER_AMOUNT,
                                                 Parameters.FOIL_ROLL_MIN_AMOUNT,
Parameters.FOIL_ROLL_DELIVER_TIME, "CAFORO")
    seperator_roll_refill = RawMaterialRefill(env, seperator_roll_storage,
Parameters.SEPERATOR_ORDER_AMOUNT,
                                              Parameters.SEPERATOR_MIN_AMOUNT,
Parameters.SEPERATOR_DELIVER_TIME, "SEPRO")
    insulation_tape_refill = RawMaterialRefill(env, insulation_tape_roll_storage,
Parameters.INSULATION_ORDER_AMOUNT,
                                               Parameters.INSULATION_MIN_AMOUNT,
Parameters.INSULATION_DELIVER_TIME, "INTARO")
    aluminized_pouch_refill = RawMaterialRefill(env, aluminized_pouch_roll_storage,
Parameters.POUCH_ORDER_AMOUNT,
                                                Parameters.POUCH_MIN_AMOUNT,
Parameters.POUCH_DELIVER_TIME, "ALPORO")
    anode_tab_refill = RawMaterialRefill(env, anode_tab_storage, Parameters.TAB_ORDER_AMOUNT,
                                         Parameters.TAB_MIN_AMOUNT, Parameters.TAB_DELIVER_TIME,
"ANTAB")
    cathode_tab_refill = RawMaterialRefill(env, cathode_tab_storage, Parameters.TAB_ORDER_AMOUNT,
                                           Parameters.TAB_MIN_AMOUNT, Parameters.TAB_DELIVER_TIME,
"CATAB")

def runsimulation():
    global exitflag, start
    endproc = env.process(end_process(env))
    while True:
        if start:
            env.run(until=endproc)
            exitflag = False
            start = False


def main():
```

```
    global Parameters
    Parameters = LoadParameters('SimParameters.cfg')
    SetupStorages()
    SetupMachines()
    SetupReorderProcesses()
    reciever = threading.Thread(target=sub_handler)
    reciever.start()
    runsimulation()
    reciever.join()

if __name__ == "__main__":
    main()
```

Appendix C: SimParameters.cfg file

```
SIM_STEP = 1/1000
SIM_DURATION = 1440

# Storage capacity and start level
# Raw materials
HD01_CAP = 50
HD01_INIT = 3
C45_CAP = 40
C45_INIT = 5
ELEC_CAP = 40
ELEC_INIT = 10
ANFOILROLL_CAP = 50
ANFOILROLL_INIT = 20
CARFILMROLL_CAP = 50
CARFILMROLL_INIT = 20
POSCONMC_CAP = 40
POSCONMC_INIT = 6
COND_CARB_CAP = 40
COND_CARB_INIT = 5
CAFOILROLL_CAP = 50
CAFOILROLL_INIT = 30
SEPROLL_CAP = 60
SEPROLL_INIT = 30
INSULTAPEROLL_CAP = 30
INSULTAPEROLL_INIT = 10
ALPOUCHROLL_CAP = 40
ALPOUCHROLL_INIT = 10
ANTAB_CAP = 200
ANTAB_INIT = 100
CATAB_CAP = 200
CATAB_INIT = 100
# Limited resources
CARTRIDGE_CAP = 100
# Half fabricate
ANBL_CAP = 50
ANSLCART_CAP = 50
ANLAMROLL_CAP = 50
AN_CAP = 1
CABL_CAP = 50
CABLDR_CAP = 50
CASLCART_CAP = 50
CALAMROLL_CAP = 50
CA_CAP = 1
SEPSHROLL_CAP = 50
UNCE_CAP = 100
UNCEST_CAP = 50
INSULTAPE_CAP = 100
FOPOUCH_CAP = 100
UNCESTTAB_CAP = 50
CEPRE_CAP = 200
# End products
FINCE_CAP = 100
CEBOX_CAP = 100
PALLET_CAP = 50

# Production consumption values
HD01_USE_PR_CYCLE = .7
C45_USE_PR_CYCLE = .3
POSCONMC_USE_PR_CYCLE = .6
COND_CARBON_USE_PR_CYCLE = .4
ELECTROLYTE_USE_PR_CYCLE = 1.0
ANODE_POWDER_USE_PR_CYCLE = 1.0
CATHODE_POWDER_USE_PR_CYCLE = 1.0
CATHODE_DRIED_USE_PR_CYCLE = 1.0
CATHODE_SLURRY_USE_PR_CYCLE = 45 / 1000
ANODE_SLURRY_USE_PR_CYCLE = 40 / 1000
```

```
# Production per cycle values
ANODE_POWDER_MILLED_PER_CYCLE = 1
CATHODE_POWDER_MILLED_PER_CYCLE = 1
CATHODE_POWDER_DRIED_PER_CYCLE = 1
SLURRY_COMPRESS_PER_CYCLE = 1
FOIL_PER_CYCLE = 1

# Product values
POWDER_CONTAINER_MAX = 500
CATHODE_BRICK = 35
ANODE_BRICK = 35
FOIL_ROLL_MAX = 100
CELL_STACK_MAX = 5

# Processing / cycle times
CATHODE_POWDER_CYCLE = 0.15
CATHODE_POWDER_DRYING_CYCLE = 0.1
ANODE_POWDER_CYCLE = 0.2
CATHODE_SLURRY_CYCLE = 0.15
ANODE_SLURRY_CYCLE = 0.16
CAST_CYCLE = 0.01
FOIL_LAMINATE_CYCLE = 0.1
ASSEMBLE_CYCLE = 1
STACK_CYCLE = 0.1
INSULATION_TAPE_CYCLE = 1
FORMED_POUCH_CYCLE = 2
TAB_CYCLE = 3
CELL_ASSEMBLY_CYCLE = 3
CELL_PREFORMATION_CYCLE = 30

# Machine - Container Change times
CONTAINER_CHG = 2
BLEND_CONTAINER_CHG_TIME = 12.2
CART_CONTAINER_CHG_TIME = 12.2
CAST_CART_CHG = 12.2 / 60

# Machine - Mean Time Between Failures - Mean Time To Repair
MTBF_MILL = (3 * 24 * 60)
MTTR_MILL = 99
MTBF_DRYER = (5 * 24 * 60)
MTTR_DRYER = 60
MTBF_CART = (5 * 24 * 60)
MTTR_CART = 90
MTBF_CAST = (4 * 24 * 60)
MTTR_CAST = 100
MTBF_LAMINATE = (3 * 24 * 60)
MTTR_LAMINATE = 80
MTBF_SEPERATOR = (6 * 24 * 60)
MTTR_SEPERATOR = 100
MTBF_ASSEMBLE = (4.5 * 24 * 60)
MTTR_ASSEMBLE = 90
MTBF_STACK = (3 * 24 * 60)
MTTR_STACK = 120
MTBF_INSULATION = (10 * 24 * 60)
MTTR_INSULATION = 30
MTBF_POUCH = (12 * 24 * 60)
MTTR_POUCH = 40
MTBF_TAB = (12 * 24 * 60)
MTTR_TAB = 40
MTBF_CELL_ASSEMBLY = (12 * 24 * 60)
MTTR_CELL_ASSEMBLY = 50

# Product quality values
ELECTRODE_QUALITY = .91
STACK_QUALITY = .95
STACK_TAB_QUALITY = .97
CELL_PREFORMATION_QUALITY = .98
FINISHED_CELL_QUALITY = .99

# Raw material order information
```

```
HD01_MIN_AMOUNT = 1
HD01_ORDER_AMOUNT = 10
HD01_DELIVER_TIME = 120
C45_MIN_AMOUNT = 0
C45_ORDER_AMOUNT = 5
C45_DELIVER_TIME = 140
POSCONMC_MIN_AMOUNT = 1
POSCONMC_ORDER_AMOUNT = 10
POSCONMC_DELIVER_TIME = 90
COND_CARB_MIN_AMOUNT = 2
COND_CARB_ORDER_AMOUNT = 5
COND_CARB_DELIVER_TIME = 66
ELEC_MIN_AMOUNT = 10
ELEC_ORDER_AMOUNT = 20
ELEC_DELIVER_TIME = 200
CARRIER_FILM_MIN_AMOUNT = 15
CARRIER_FILM_ORDER_AMOUNT = 25
CARRIER_FILM_DELIVER_TIME = 400
FOIL_ROLL_MIN_AMOUNT = 10
FOIL_ROLL_ORDER_AMOUNT = 30
FOIL_ROLL_DELIVER_TIME = 420
SEPERATOR_MIN_AMOUNT = 6
SEPERATOR_ORDER_AMOUNT = 25
SEPERATOR_DELIVER_TIME = 250
INSULATION_MIN_AMOUNT = 8
INSULATION_ORDER_AMOUNT = 20
INSULATION_DELIVER_TIME = 120
POUCH_MIN_AMOUNT = 5
POUCH_ORDER_AMOUNT = 20
POUCH_DELIVER_TIME = 200
TAB_MIN_AMOUNT = 10
TAB_ORDER_AMOUNT = 30
TAB_DELIVER_TIME = 300
```

Appendix D: SimControlAndGraph.py code

```
from tkinter import ttk
import tkinter as tk
import tkinter.scrolledtext as ScrolledText
from matplotlib import style
import matplotlib.animation as animation
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
import logging
import zmq
import json
import threading
import pandas as pd
import matplotlib

matplotlib.use("TkAgg")

LARGE_FONT = ("Verdana", 24)
MEDIUM_FONT = ("Verdana", 18)
style.use("ggplot")

f = Figure(figsize=(5, 5), dpi=100)
a = f.add_subplot(111)
a.autoscale(enable=True, axis='both', tight=None)

plotdata = pd.DataFrame()
machines = []
storages = []

statustree = None
storagetree = None

run_consumer = False

context = zmq.Context()
# recieve work
receiver = context.socket(zmq.SUB)
receiver.setsockopt_string(zmq.SUBSCRIBE, "ST")

receiver.connect("tcp://127.0.0.1:5555")
sender = context.socket(zmq.PUB)
sender.connect("tcp://127.0.0.1:5556")

class TextHandler(logging.Handler):
    # This class allows you to log to a Tkinter Text or ScrolledText widget
    # Adapted from Moshe Kaplan: https://gist.github.com/moshekaplan/c425f861de7bbf28ef06

    def __init__(self, text):
        # run the regular Handler __init__
        logging.Handler.__init__(self)
        # Store a reference to the Text it will log to
        self.text = text

    def emit(self, record):
        msg = self.format(record)
        def append():
            self.text.configure(state='normal')
            self.text.insert(tk.END, msg + '\n')
            self.text.configure(state='disabled')
            # Autoscroll to the bottom
            self.text.yview(tk.END)
        # This is necessary because we can't modify the Text from other threads
        self.text.after(0, append)

def animate(i):
    a.clear()

    if len(plotdata) > 0:
```

```
        for name, group in plotdata.groupby('name'):
            group.plot(x='time', y='level', ax=a, label=name)
        a.legend(bbox_to_anchor =(1.1, 1.0))

class SimControlApp(tk.Tk):
    def __init__(self, *args, **kwargs):

        tk.Tk.__init__(self, *args, **kwargs)

        tk.Tk.wm_title(self, "Battery Plant Simulator")

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        for F in (StartPage, StatusPage, GraphPage, LogPage):

            frame = F(container, self)

            self.frames[F] = frame

            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(StartPage)

    def show_frame(self, cont):

        frame = self.frames[cont]
        frame.tkraise()

class Machine():
    def __init__(self, name, status):
        self.name = name
        self.status = status

class Storage():
    def __init__(self, name, level):
        self.name = name
        self.level = level

def mogrify(topic, msg):
    return topic + ' ' + json.dumps(msg)

def demogrify(topicmsg):
    """ Inverse of mogrify() """
    json0 = topicmsg.find('{')
    topic = topicmsg[0:json0].strip()
    msg = json.loads(topicmsg[json0:])
    return topic, msg


def consumer():
    global plotdata

    while run_consumer:
        topic, msg = demogrify(receiver.recv_string())
        data = ""
        for key, value in msg.items():
            data += key + ": " + str(value) + " "
        logging.info('{0} {1} {2} {3}'.format('Topic:', topic, "Data:", data))
        if 'STORAGE' in topic:
            name = msg['storage']
            level = int(msg['level'])
            time = float(msg['time'])
            new_data = pd.DataFrame({"name": name, "level": level, "time": time}, index=[0])
            plotdata = plotdata.append(new_data, ignore_index=True)
            if any(elem.name == name for elem in storages):
```

```
                storage = next(storage for storage in storages if storage.name == name)
                storage.level = level
            else:
                storage = Storage(name, level)
                storages.append(storage)
        elif 'STATUS' in topic:
            name = msg['name']
            status = msg['status']
            time = float(msg['time'])
            if any(elem.name == name for elem in machines):
                machine = next(machine for machine in machines if machine.name == name)
                machine.status = status
            else:
                machine = Machine(name, status)
                machines.append(machine)

listen_thread = threading.Thread(target=consumer)

def send_stop():
    command = {'command': 'Exit'}
    msg = mogrify("COMMAND", command)
    sender.send_string(msg)

def send_start():
    command = {'command': 'Start'}
    msg = mogrify("COMMAND", command)
    sender.send_string(msg)

def start_consumer():
    global run_consumer, listen_thread
    run_consumer = True
    listen_thread.start()

def sub_topic_filter(filter):
    receiver.setsockopt_string(zmq.SUBSCRIBE, filter)

def unsub_topic_filter(filter):
    receiver.setsockopt_string(zmq.UNSUBSCRIBE, filter)

class StartPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        top_frame = tk.Frame(self)
        top_frame.pack(side='top')
        mid_frame = tk.Frame(self)
        mid_frame.pack(side='top', fill=tk.BOTH, expand=True)
        bot_frame = tk.Frame(self)
        bot_frame.pack(side='top')

        label = tk.Label(top_frame, text="Controls", font=LARGE_FONT)
        label.pack(pady=10, padx=10)

        nav_button = ttk.Button(top_frame, text="Show Status Page",
                                command=lambda: controller.show_frame(StatusPage))
        nav_button.pack(side='left', pady=20)

        nav_button2 = ttk.Button(top_frame, text="Show Storage Graph Page",
                                command=lambda: controller.show_frame(GraphPage))
        nav_button2.pack(side='left', pady=20)

        nav_button3 = ttk.Button(top_frame, text="Show Logging Page",
                                command=lambda: controller.show_frame(LogPage))
        nav_button3.pack(side='left', pady=20)

        button1 = ttk.Button(mid_frame, text="Start listener",
                             command=start_consumer)
        button1.pack(pady=20)

        button2 = ttk.Button(mid_frame, text="Start Simulator", command=send_start)
```

```
        button2.pack(pady=20)

        button3 = ttk.Button(mid_frame, text="Stop Simulator", command=send_stop)
        button3.pack(pady=20)

        entry1 = ttk.Entry(bot_frame)
        entry1.insert(0, "ST")
        entry1.pack(side='left', pady=20)

        entry_button = ttk.Button(
            bot_frame, text="SUBSCRIBE", command=lambda: sub_topic_filter(entry1.get()))
        entry_button.pack(side='left')

        entry2 = ttk.Entry(bot_frame)
        entry2.insert(0, "UNSUB")
        entry2.pack(side='left', pady=20)

        entry2_button = ttk.Button(
            bot_frame, text="UNSUBSCRIBE", command=lambda: unsub_topic_filter(entry2.get()))
        entry2_button.pack(side='left')

class StatusPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        global statustree, storagetree
        label = tk.Label(self, text="Status", font=LARGE_FONT)
        label.pack(pady=10, padx=10)

        button1 = ttk.Button(self, text="Back to Controls",
                             command=lambda: controller.show_frame(StartPage))
        button1.pack(pady=20)

        label1 = tk.Label(self, text="Machines", font=MEDIUM_FONT)
        label1.pack(pady=10, padx=10)
        statustree = ttk.Treeview(self, columns=('name', 'status'), show='headings')
        statustree.heading('name', text='Name')
        statustree.heading('status', text='Status')
        statustree.pack(pady=20, padx=10, fill=tk.BOTH, expand=True)

        label2 = tk.Label(self, text="Storages", font=MEDIUM_FONT)
        label2.pack(pady=10, padx=10)

        storagetree = ttk.Treeview(self, columns=('name', 'level'), show='headings')
        storagetree.heading('name', text='Name')
        storagetree.heading('level', text='Level')
        storagetree.pack(pady=20, padx=10, fill=tk.BOTH, expand=True)

        self.after(0, self.refresh())
    def refresh(self):
        for item in statustree.get_children():
            statustree.delete(item)
        for machine in machines:
            statustree.insert('', 'end', values=(machine.name, machine.status))

        for item in storagetree.get_children():
            storagetree.delete(item)
        for storage in storages:
            storagetree.insert('', 'end', values=(storage.name, storage.level))
        self.after(1000, self.refresh)

class GraphPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Storage Graph", font=LARGE_FONT)
        label.pack(pady=10, padx=10)

        button1 = ttk.Button(self, text="Back to Controls",
                             command=lambda: controller.show_frame(StartPage))
```

```
        button1.pack(pady=20)

        canvas = FigureCanvasTkAgg(f, self)
        canvas.draw()
        canvas.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)

        toolbar = NavigationToolbar2Tk(canvas, self)
        toolbar.update()
        canvas._tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

class LogPage(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Logging", font=LARGE_FONT)
        label.pack(pady=10, padx=10)

        button1 = ttk.Button(self, text="Back to Controls",
                             command=lambda: controller.show_frame(StartPage))
        button1.pack(pady=20)

        logtext = ScrolledText.ScrolledText(self, state='disabled')
        logtext.configure(font='TkFixedFont')
        logtext.pack(pady=20, padx=10, fill=tk.BOTH, expand=True)

        text_handler = TextHandler(logtext)

        # Logging configuration
        logging.basicConfig(filename='Updates.log',
            level=logging.INFO,
            format='%(asctime)s - %(levelname)s - %(message)s')

        logger = logging.getLogger()
        logger.addHandler(text_handler)
        logging.info('------------------New Run------------------')

app = SimControlApp()
ani = animation.FuncAnimation(f, animate, interval=1000)
app.mainloop()
```

Appendix E: ZMQPub.py code

```
import zmq
import json

def mogrify(topic, msg):
    return topic + ' ' + json.dumps(msg)

def demogrify(topicmsg):
    """ Inverse of mogrify() """
    json0 = topicmsg.find('{')
    topic = topicmsg[0:json0].strip()
    msg = json.loads(topicmsg[json0:])
    return topic, msg

def publisher():
    context = zmq.Context()
    # recieve work
    socket = context.socket(zmq.PUB)

    socket.connect("tcp://127.0.0.1:5556")

    while True:

        send_command = input("Command to send: ")
        command = {'command': send_command}
        msg = mogrify("COMMAND", command)
        socket.send_string(msg)

publisher()
```

Appendix F: Client.cs code

```
using System;
using UnityEngine;

namespace PubSub
{
    public class Client : MonoBehaviour
    {
        public enum ClientStatus
        {
            Inactive,
            Activating,
            Active,
            Deactivating
        }

        [SerializeField] private string host;
        [SerializeField] private string port;
        [SerializeField] private string topic;

        [SerializeField] private string pushPort;
        private Listener _listener;
        private Sender _sender;
        private ClientStatus _clientStatus = ClientStatus.Inactive;

        private void Start()
        {
            _listener = new Listener(host, port, topic, HandleMessage);
            _sender = new Sender(host, pushPort);
            EventManager.Instance.onStartClient.AddListener(OnStartClient);
            EventManager.Instance.onClientStarted.AddListener(() => _clientStatus =
ClientStatus.Active);
            EventManager.Instance.onStopClient.AddListener(OnStopClient);
            EventManager.Instance.onClientStopped.AddListener(() => _clientStatus =
ClientStatus.Inactive);
            EventManager.Instance.onSendChange.AddListener(OnSendChange);
            EventManager.Instance.onSendStartSim.AddListener(OnSendStartSim);
        }

        private void Update()
        {
            if (_clientStatus == ClientStatus.Active)
                _listener.DigestMessage();
        }

        private void OnDestroy()
        {
            if (_clientStatus != ClientStatus.Inactive)
                OnStopClient();
        }

        private void HandleMessage(string message)
        {
            Debug.Log(message);
            if (message.Contains("STORAGE"))
            {
                string jsonString = message.Substring(message.IndexOf("STORAGE") + "STORAGE ".Length);
                StorageUpdate storageUpdate = JsonUtility.FromJson<StorageUpdate>(jsonString);
                GameObject storageObject = GameObject.Find(storageUpdate.storage);
                if (storageObject != null)
                {
                    storageObject.GetComponent<StorageHandler>().SetLevel(storageUpdate.level);
                }
            }
            else if (message.Contains("STATUS"))
            {
                string jsonString = message.Substring(message.IndexOf("STATUS") + "STATUS ".Length);
                StatusUpdate statusUpdate = JsonUtility.FromJson<StatusUpdate>(jsonString);
```

```
                GameObject statusObject = GameObject.Find(statusUpdate.name);
                if (statusObject != null)
                {
                    statusObject.GetComponent<MachineHandler>().SetStatus(statusUpdate.status);
                }
            }
        }

        private void OnSendChange()
        {
            Debug.Log("Sending Change...");
            _sender.SendMessage("CHANGE", "{\"name\": \"ANMI\"}");
            Debug.Log("Change Sent...");

        }
        private void OnSendStartSim()
        {
            Debug.Log("Sending Start Sim...");
            _sender.SendMessage("COMMAND", "{\"command\": \"Start\"}");
            Debug.Log("Start Sent...");

        }

        private void OnStartClient()
        {
            Debug.Log("Starting client...");
            _clientStatus = ClientStatus.Activating;
            _listener.Start();
            _sender.Start();
            Debug.Log("Client started!");
        }

        private void OnStopClient()
        {
            Debug.Log("Stopping client...");
            _clientStatus = ClientStatus.Deactivating;
            _sender.Stop();
            _listener.Stop();
            Debug.Log("Client stopped!");
        }

        [Serializable]
        public class StorageUpdate
        {
            public string from_to;
            public string storage;
            public int level;
            public string direction;
            public float time;
        }

        [Serializable]
        public class StatusUpdate
        {
            public string name;
            public string status;
            public float time;
        }
    }
}
```

Appendix G: Sender.cs code

```
using System;
using System.Collections.Concurrent;
using System.Threading;
using NetMQ;
using NetMQ.Sockets;

namespace PubSub
{
    public class Sender
    {
        private readonly string _host;
        private readonly string _port;
        private PublisherSocket _sender;

        public Sender(string host, string port)
        {
            _host = host;
            _port = port;
        }

        public void Start()
        {
            _sender = new PublisherSocket();
            _sender.Connect($"tcp://{_host}:{_port}");
        }

        public void Stop()
        {
            _sender.Close();
        }

        // Start is called before the first frame update
        public void SendMessage(string topic, string message)
        {
            string msg = topic.ToString() + " " + message.ToString();
            _sender.SendFrame(msg);
        }

    }

}
```

Appendix H: MachineHandler.cs code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class MachineHandler : MonoBehaviour
{
    public string machineName = "Machine";
    public string machineId = "M";
    public string machineStatus = "IDLE";
    public GameObject floorIndicator;
    public GameObject machineObject;
    public Transform player;
    public TextMeshPro nameTextMesh;
    public TextMeshPro statusTextMesh;
    public float drawDistance = 100.0f;

    private GameObject machine;
    private Vector3 lastPos;


    // Start is called before the first frame update
    void Start()
    {
        player = GameObject.Find("Player").transform;
        transform.name = machineId;
        nameTextMesh.text = machineName;
        statusTextMesh.text = machineStatus;
        machine = Instantiate(machineObject);
        machine.transform.SetParent(transform);
        machine.transform.localPosition = Vector3.zero;

        switch (machineStatus)
        {
            case "RUN":
                floorIndicator.GetComponent<MeshRenderer>().material.color = Color.green;
                break;
            case "DOWN":
                floorIndicator.GetComponent<MeshRenderer>().material.color = Color.red;
                break;
            case "IDLE":
                floorIndicator.GetComponent<MeshRenderer>().material.color = Color.blue;
                break;
            default:
                floorIndicator.GetComponent<MeshRenderer>().material.color = Color.yellow;
                break;
        }
    }

    // Update is called once per frame
    void Update()
    {
        if(statusTextMesh.text != machineStatus){
            statusTextMesh.text = machineStatus;
            switch (machineStatus)
            {
                case "RUN":
                    floorIndicator.GetComponent<MeshRenderer>().material.color = Color.green;
                    break;
                case "DOWN":
                    floorIndicator.GetComponent<MeshRenderer>().material.color = Color.red;
                    break;
                case "IDLE":
                    floorIndicator.GetComponent<MeshRenderer>().material.color = Color.blue;
                    break;
                default:
                    floorIndicator.GetComponent<MeshRenderer>().material.color = Color.yellow;
```

```
                    break;
            }
        }
        if (lastPos != player.position) {
            if (Vector3.Magnitude(player.position - transform.position) > drawDistance)
            {
                machine.SetActive(false);
            }
            else
            {
                machine.SetActive(true);
            }
            lastPos = player.position;
        }
    }

    public void SetStatus(string status)
    {
        machineStatus = status;
    }
}
```

Appendix I: StorageHandler.cs code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class StorageHandler : MonoBehaviour
{
    public string storageName = "Storage";
    public string storageId = "Storage";
    public int storageLevel = 0;
    public int storageCapacity = 50;
    public int storageWidth = 10;
    public Vector3 objectScale = Vector3.one;
    public float horizontalDistance = 1.0f;
    public float verticalDistance = 1.0f;
    public GameObject storageObjectPrefab;
    public GameObject cube;
    public TextMeshPro textMesh;
    TextMeshPro levelTextMesh;

    int storageHeight;
    Vector2[] locations;
    List<GameObject> storageObjects = new List<GameObject>();
    // Start is called before the first frame update
    void Start()
    {
        transform.name = storageId;
        if(ReadParameters.Instance.setupParameters.ContainsKey(storageId + "_INIT"))
        {
            int.TryParse(ReadParameters.Instance.setupParameters[storageId + "_INIT"], out
storageLevel);
        }

        if (ReadParameters.Instance.setupParameters.ContainsKey(storageId + "_CAP"))
        {
            int.TryParse(ReadParameters.Instance.setupParameters[storageId + "_CAP"], out
storageCapacity);
        }

        if (storageLevel > storageCapacity)
        {
            storageLevel = storageCapacity;
        }
        else if (storageLevel < 0)
        {
            storageLevel = 0;
        }
        levelTextMesh = transform.Find("LevelText").GetComponent<TextMeshPro>();
        levelTextMesh.text = storageLevel.ToString();
        textMesh.text = storageName;
        storageHeight = Mathf.CeilToInt(storageCapacity / storageWidth);
        locations = new Vector2[storageCapacity];
        for(int i = 0; i < storageHeight; i++)
        {
            GameObject shelf = Instantiate(cube);
            shelf.transform.SetParent(transform);
            shelf.transform.localPosition = new Vector3(storageWidth * horizontalDistance / 2.0f -
0.5f, i * verticalDistance + verticalDistance, 0f);
            shelf.transform.localScale = new Vector3(storageWidth * horizontalDistance + 1.0f, 0.1f,
1.4f);
            for(int j = 0; j < storageWidth; j++)
            {
                if((i * storageWidth + j) < storageCapacity)
                {
                    locations[i * storageWidth + j] = new Vector2(j * horizontalDistance, i *
verticalDistance);
                }
```

```
            }
        }
        GameObject shelfPole1 = Instantiate(cube);
        shelfPole1.transform.SetParent(transform);
        shelfPole1.transform.localPosition = new Vector3(-1f, 0.1f + storageHeight * verticalDistance
/ 2, 0f);
        shelfPole1.transform.localScale = new Vector3(0.2f, 0.2f + storageHeight * verticalDistance,
1.4f);

        GameObject shelfPole2 = Instantiate(cube);
        shelfPole2.transform.SetParent(transform);
        shelfPole2.transform.localPosition = new Vector3(storageWidth * horizontalDistance, 0.1f +
storageHeight * verticalDistance / 2, 0f);
        shelfPole2.transform.localScale = new Vector3(0.2f, 0.2f + storageHeight * verticalDistance,
1.4f);

        for (int i = 0; i < storageLevel; i++)
        {
            GameObject gameObject = Instantiate(storageObjectPrefab);
            gameObject.transform.SetParent(transform);
            gameObject.transform.localPosition = locations[i];
            Vector3 localScale = gameObject.transform.localScale;
            gameObject.transform.localScale = new Vector3(localScale.x * objectScale.x, localScale.y *
objectScale.y, localScale.z * objectScale.z);
            storageObjects.Add(gameObject);
        }
    }
    // Update is called once per frame
    void Update()
    {
        if(storageLevel > storageCapacity)
        {
            storageLevel = storageCapacity;
        }
        else if(storageLevel < 0)
        {
            storageLevel = 0;
        }

        if(storageLevel < storageObjects.Count)
        {
            for(int i = storageLevel; i < storageObjects.Count; i++)
            {
                GameObject storageObject = storageObjects[i];
                storageObjects.RemoveAt(i);
                GameObject.Destroy(storageObject);
            }
        }
        else if(storageLevel > storageObjects.Count)
        {
            for (int i = storageObjects.Count; i < storageLevel; i++)
            {
                GameObject gameObject = Instantiate(storageObjectPrefab);
                gameObject.transform.SetParent(transform);
                gameObject.transform.localPosition = locations[i];
                Vector3 localScale = gameObject.transform.localScale;
                gameObject.transform.localScale = new Vector3(localScale.x * objectScale.x,
localScale.y * objectScale.y, localScale.z * objectScale.z);
                storageObjects.Add(gameObject);
            }
        }
    }

    public void SetLevel(int level)
    {
        storageLevel = level;
        levelTextMesh.text = storageLevel.ToString();
    }
}
```

Appendix J: PerformanceTest.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using Unity.Profiling;

public class PerformanceTest : MonoBehaviour
{
    public static PerformanceTest Instance;

    public float updateInterval = 0.5F;

    private float accum = 0; // FPS accumulated over the interval
    private int frames = 0; // Frames drawn over the interval
    private int messagesRx = 0; // Messages recieved over the interval
    public int totalMessagesRx = 0;
    public int lostMessages = 0;
    private int messagesTx = 0; // Messages transmitted over the interval
    private float timeleft;
    private float timePassed;
    public float fps;
    public float mpsr;
    public float mpst;
    public int verts;
    public int tris;
    public int numObjects;
    public string fileName = "PerformanceData.csv";
    private string filePath = "";
    private StreamWriter streamWriter;
    ProfilerRecorder verticesRecorder;
    ProfilerRecorder trianglesRecorder;

    private void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
            this.filePath = Application.dataPath + "/Data/" + fileName;
        }
        else
            Destroy(this);
    }

    // Start is called before the first frame update
    void Start()
    {
        streamWriter = new StreamWriter(filePath);
        streamWriter.WriteLine("Time fps mpsr mpst noo not nov nolm");
        timeleft = updateInterval;
        verticesRecorder = ProfilerRecorder.StartNew(ProfilerCategory.Render, "Vertices Count");
        trianglesRecorder = ProfilerRecorder.StartNew(ProfilerCategory.Render, "Triangles Count");
    }

    // Update is called once per frame
    void Update()
    {
        timeleft -= Time.deltaTime;
        accum += Time.timeScale / Time.deltaTime;
        timePassed += Time.deltaTime;
        ++frames;

        if (timeleft <= 0.0)
        {
            fps = accum / frames;
            mpsr = messagesRx / timePassed;
            mpst = messagesTx / timePassed;
```

```
            timeleft = updateInterval;
            timePassed = 0;
            accum = 0;
            frames = 0;
            messagesRx = 0;
            messagesTx = 0;
            GetObjectInformation();
            WriteToFile(Time.time, fps, mpsr, mpst, numObjects, tris, verts, lostMessages);
        }
        if (verticesRecorder.Valid)
            int.TryParse(verticesRecorder.LastValue.ToString(), out verts);
        if (trianglesRecorder.Valid)
            int.TryParse(trianglesRecorder.LastValue.ToString(), out tris);
    }

    private void WriteToFile(float time, float fps, float mpsr, float mpst, int noo, int not, int nov,
int nolm)
    {
        streamWriter.WriteLine(String.Format("{0:F2}", time) + " " + String.Format("{0:F2}", fps) + "
" + String.Format("{0:F2}", mpsr) + " " + String.Format("{0:F2}", mpst) + " " + noo + " " + not + " "
+ nov + " " + nolm);
    }

    void GetObjectInformation()
    {
        GameObject[] ob = FindObjectsOfType(typeof(GameObject)) as GameObject[];
        numObjects = ob.Length;
    }


    public void addMessageRecieved(int messageSentCount)
    {
        ++messagesRx;
        ++totalMessagesRx;
        lostMessages = messageSentCount - totalMessagesRx;
    }

    public void addMessageSent()
    {
        ++messagesTx;
    }

    void OnApplicationQuit()
    {
        streamWriter.Flush();
        streamWriter.Close();
        verticesRecorder.Dispose();
    }
}
```

Appendix K: ZMQ_StressTest.py

```
import threading
import zmq
import json
import time

from zmq.eventloop import ioloop
ioloop.install()

message_count = 0
connection_count = 0

start = False
exitflag = False
sleeptime = 1
sleepadjustinterval = 10
last_adjust_time = time.time()

def mogrify(topic, msg):
    """ json encode the message and prepend the topic """
    return topic + ' ' + json.dumps(msg)

def demogrify(topicmsg):
    """ Inverse of mogrify() """
    json0 = topicmsg.find('{')
    topic = topicmsg[0:json0].strip()
    msg = json.loads(topicmsg[json0:])
    return topic, msg

def getcommand(topic, msg):
    global exitflag, start
    print("Received: {topic} {message}".format(topic=topic, message=msg))
    if topic == "COMMAND":
        if msg['command'] == "Exit":
            print("Received exit command, client will stop receiving messages")
            exitflag = True
        elif msg['command'] == "Start":
            print("Received Start command, Simulation starts")
            start = True
    else:
        print("Unknown topic recieved")

context = zmq.Context()
socketPub = context.socket(zmq.PUB)  # <- the PUBLISH socket
socketSub = context.socket(zmq.SUB)  # <- the SUBSCRIBE socket
socketSub.setsockopt_string(zmq.SUBSCRIBE, "C") # <- topic subscription

socketSub.bind("tcp://*:5556")
socketPub.bind('tcp://*:5555')

def sub_handler():
    while True:
        topic, msg = demogrify(socketSub.recv_string())
        getcommand(topic, msg)

def pub_handler(msg):
    socketPub.send_string(msg)

def update_status(name, status):
    global last_adjust_time, sleeptime, message_count
    message_count += 1
    data = {
        'name': name,
        'status': status,
        'time': message_count
    }
    msg = mogrify("STATUS", data)
    pub_handler(msg)
```

```
    time.sleep(sleeptime)
    if time.time() - last_adjust_time >= sleepadjustinterval:
        sleeptime *= 0.5
        last_adjust_time = time.time()



def runtest():
    while True:
        if start:
            update_status("TEST", str(time.time()))

def main():
    reciever = threading.Thread(target=sub_handler)
    reciever.start()
    runtest()
    reciever.join()

if __name__ == "__main__":
    main()
```

Appendix L: PerfAnalysis.py

```
import math
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
import plotly.express as px
import plotly.graph_objs as go
from plotly.subplots import make_subplots
from scipy.fft import fftfreq
from scipy.signal.filter_design import normalize
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import seaborn as sns

# Reading of CSV to Dataframe
df = pd.concat([pd.read_csv('PerfData3.csv', sep=';', usecols= ['Time', 'fps', 'mpsr', 'mpst', 'noo',
'not', 'nov'], decimal=',', encoding="UTF-8")])

df = df.fillna(0)

features  = ['fps', 'mpsr', 'mpst', 'noo', 'not', 'nov']
fig = px.line(df, x='Time', y=features)
fig.show()

fig = make_subplots(
    rows=math.ceil(len(features) / 6), cols=6,
    shared_xaxes=True,
    vertical_spacing=0.03
)
plotindex = 0
for feature in features:
    fig.add_trace(
        go.Histogram(x=df[feature], histnorm='probability', name=feature),
        col=(plotindex % 6) + 1,
        row=math.floor(plotindex / 6) + 1
    )
    plotindex += 1
fig.show()

features = ['Time', 'fps', 'mpsr', 'mpst', 'noo', 'not', 'nov']

scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])

pca = PCA(n_components=3)
components = pca.fit_transform(df[features])
labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_ * 100)
}

loadings = pca.components_.T * np.sqrt(pca.explained_variance_)

total_var = pca.explained_variance_ratio_.sum() * 100

fig = px.scatter(
    components, x=0, y=1, color=df['fps'],
    title=f'Total Explained Variance: {total_var:.2f}%',
    labels=labels
)

for i, feature in enumerate(features):
    fig.add_shape(
        type='line',
        x0=0, y0=0,
        x1=loadings[i, 0],
        y1=loadings[i, 1]
    )
```

```
    fig.add_annotation(
        x=loadings[i, 0],
        y=loadings[i, 1],
        ax=0, ay=0,
        xanchor="center",
        yanchor="bottom",
        text=feature,
    )

fig.show()

corr = df[features].corr()
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True
corr[mask] = np.nan
(corr
 .style
 .background_gradient(cmap='coolwarm', axis=None, vmin=-1, vmax=1)
 .highlight_null(null_color='#f1f1f1')  # Color NaNs grey
 .set_precision(2))

plt.figure(figsize=(12,8))
sns.heatmap(corr, cmap="Greens",annot=True)
plt.show()
```