

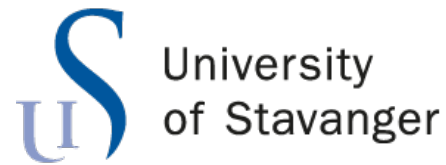


University of
Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study Program/Specialization:	Spring semester 2022
Master of Science in Computer Science Data Science	Open / Confidential
Author: Sondre Tennø	
Supervisor: Ass. Prof. Naeem Khademi	
Title of Master's Thesis: Further Developing a Procedural Digital Twin for Road Tunnels	
ECTS: 30	
Keywords: Road Tunnels Digital Twins Sensor Communications	Number of Pages: 74 Stavanger, 15 June 2022



Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Further Developing a Procedural Digital Twin for Road Tunnels

Master's Thesis in Computer Science - Data Science by

Sondre Tennø

Supervisor

Ass. Prof. Naeem Khademi

June 15, 2022

Abstract

A Digital Twin is a close to as possible replica of a real world application, in the digital world. The Digital Twin aims to simulate a process, in real time so the user can generate information and value from the process. It is a representation of a physical asset, object or service. Often the Digital Twin takes data from real sensor to mimic the live process in a digital representation. In this thesis, the focus is on Digital Twin representations of road tunnels in Norway. This thesis builds further on previously designed implementation and thesis written at University of Stavanger. The project creates tunnel twins for any given Norwegian tunnel, by using Satens Vegvesen public API for tunnel information.

Extending work made previously that connects static models with generated sensor data to display sensors in a tunnel made in Unity. This thesis furthers develops this project, by aiming to find ways to improve the architecture made in the current implementation while also finding new features to add based on what is seen in other Digital Twins in the academic world. Implementing these new features and changes to see how they are helpful to the continuous work of this Digital Twin project.

Acknowledgements

I would like to thank my supervisor, Ass. Prof.Naeem Khademi for supporting me through writing this thesis and giving guidance when needed.

I would also like this time to thank my friends and family for always supporting me in my journey.

Contents

Abstract	ii
Acknowledgements	iii
Contents	iv
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Thesis Structure	3
1.4 Key Concepts	4
1.4.1 Previous Stack	4
1.4.2 Stack to be Implemented	7
2 Background	8
2.1 Related Work	8

CONTENTS

2.1.1	Digital Twins	9
2.1.2	Digital Twins Architecture	9
2.1.3	Cloud in Digital Twins	10
2.1.4	InfluxDB in Digital Twins	13
3	Methodology	18
3.1	Idea	18
3.1.1	Academia	19
3.2	Methodology	19
4	Design	21
4.1	Current Architecture	21
4.2	Digital Twins Layers of Architecture	23
4.3	Proposed features to be Added to the Digital Twin	25
4.3.1	Monitoring	25
4.3.2	Security	26
4.3.3	Graphical User Interface	27
4.4	Digital Twin Areas of Improvement	30
4.4.1	Integration Layer	30
4.4.2	Information Layer	30
4.4.3	Communication Layer	35

CONTENTS

4.4.4	Functional Layer	36
4.4.5	Security Layer	36
4.4.6	New Proposed Architecture Layer Model	37
5	Implementation	38
5.1	GUI	38
5.2	Login System	42
5.3	Realistic OPC Emulator	45
5.4	Kafka Broker	46
5.5	Monitoring	48
6	Results	50
6.1	Experimental Setup	50
6.1.1	Requirements	51
6.1.2	Current State	53
6.1.3	OPC Emulator	53
6.1.4	Broker	55
6.2	Research Questions	56
6.3	Visualization	58
7	Conclusion	68

CONTENTS

7.1	Future Work	69
7.1.1	Complete failed implementations	69
7.1.2	Platform	69
7.1.3	Quality of textures	70
7.1.4	More sensors	70
7.1.5	Cloud	70
7.1.6	Security	71
	References	75

List of Figures

1.1	Component Diagram taken from the thesis "Digital Tunnel Twin Using Procedurally Made 3D Models" [5]	5
2.1	Replica of model described in the six layered cloud based Digital Twin [12]	12
3.1	Gantt Chart	19
4.1	Component Diagram taken from the thesis "Digital Tunnel Twin Using Procedurally Made 3D Models" [5]	22
4.2	Previous Architecture in Layers	24
4.3	GUI Wireframe	28
4.4	Login Screen	29
4.5	Sign Up Screen	29
4.6	Current and Proposed Layer Architecture	37
5.1	Running the OPC Emulator	39

LIST OF FIGURES

5.2	Running the Broker	39
5.3	Running the Digital Twin Framework	40
5.4	Screenshot of the GUI main screen	41
5.5	Screenshot of the GUI login screen	43
5.6	Screenshot of the GUI register screen	43
5.7	SQL Table for user data	44
5.8	Database flow	44
5.9	Current OPC data flow	45
5.10	Figure showing example of OPC Mainserver with OPC Servers connected and sensor	46
5.11	Publisher subscriber with Broker, taken from Nohut thesis [5]	47
5.12	View from Grafana with table view for a specific sensor . . .	48
5.13	View from Grafana with bar view for a specific sensor . . .	48
6.1	Screenshot of the GUI login screen	59
6.2	Screenshot of the main GUI screen	60
6.3	View of the zoomed out Virtual Twin in Unity	61
6.4	Immediate view of "Kleppetunnelen" as of running the Dig- ital Twin	62
6.5	View of the entrance of "Kleppetunnelen"	63
6.6	View of the entrance of "Kleppetunnelen", taken from No- hut's thesis with the old implementation [5]	64

LIST OF FIGURES

6.7 View of "Kleppetunnelen" midway in, with some objects . . . 65

6.8 View of selecting objects inside the Virtual Twin 65

6.9 View of the car fire simulation 66

6.10 View from Grafana with bar view for a specific sensor . . . 66

List of Tables

- 2.1 Table Showing Cloud Based Digital Twins in Academia. . . 13
- 2.2 Table Showing Digital Twins in Academia Using InfluxDB . 15

- 4.1 Table Showing Time-Series Databases based on popularity
from db-engine.com 33
- 4.2 Table Showing Benchmarks Tests for Several Time-Series
Databases 34

- 6.1 Software Dependencies 52
- 6.2 Computer hardware used for testing 53
- 6.3 Testing done by Nohut on old implementation, Reaction Time
[5] 55
- 6.4 Testing done by Nohut on old implementation, Notification
Time [5] 56

Chapter 1

Introduction

1.1 Motivation

The ability to simulate a process, or the lifetime of an object can be very valuable to many different industries. To simulate everything that would also happen to a physical asset, in a digital space. While a simulation usually studies one particular process, a digital twin can run any number of useful simulations in order to study multiple processes at a time [7]. Digital twins also benefit from the two-way communication with real time data, which simulations usually do not [7]. To study the entire process of a production line, or the performance of a motor in a vehicle in real time can be very valuable to companies.

The concept of a digital twin has been known for almost twenty years now, but the growth in the field has increased exponentially in the last five years. This growth is largely due to the increased use of IoT and access to quality networking. Digital twin was predicted in IEEE(Institute of Electrical and Electronics Engineers) as a top three tech trend to be adopted in 2020.

A digital twin is a digital representation of a physical asset, object or service. A digital twin will replicate the instance in a digital space, in essence, a computer program that uses real world data to create simulations that can predict outcomes of the physical asset. There are a lot of different use cases

1.2 Objectives

for digital twins, and for each year that goes by, the concept will be tested in new fields [6].

In recent years, a digital twin for Norwegian tunnel systems has been developed at the UiS(University of Stavanger). This framework is built on the data collection of the NVDB database, which is a database containing information about every tunnel in Norway. Given digital twins' dependency on data, this project wouldn't have been possible without this data. Motivation for this thesis is to further develop the application previously made at UiS and see if any improvement can be made to its architecture and what features could be added.

1.2 Objectives

The main goal of this project is to focus on further developing a procedural method devised for the creation of digital twins for tunnels at a low cost. Previously at UiS, there have been developed methods to procedurally create low-fidelity 3D models for Norwegian tunnels using NVDB's public data. These 3D models have then been used to procedurally create a digital twin framework that uses these models to replicate Norwegian tunnels.

Given that the problems I want to solve aren't directly related to each other, I have divided them into separate milestones:

- Studying and understanding the state of the art of digital twins. How are they made up from an architectural standpoint. Identify which parts of the digital twin framework can be further developed or re-designed to increase performance and easier implement new features.
- Study the currently used sensor communication protocol and see if the framework can be improved by using a different one.
- Adding analytics or monitoring and visualization interface for the database.
- Adding GUI(Graphical User Interface) to the Digital Twin.

1.3 Thesis Structure

By implementing these objectives as stated above, I hope to could answer the following research questions:

- Are there improvements to be made to the architecture of the Digital Twin?
- What features are there to be added to the Digital Twin that could improve functionality?
- Are the InfluxDB a suitable database to be used for this project?
- It is important to get people to use this application, how can changes be done to the implementation to ease the barrier to entry?

1.3 Thesis Structure

The structure of the chapters of this thesis will be as followed:

- **Chapter 1** - Introduction - Introduces what the thesis will be about and what the goal of this thesis is.
- **Chapter 2** - Background - Introduces similar types of projects from academia and industry. Comparing how they are similar to this thesis and what makes them different.
- **Chapter 3** - Methodology - Presents the idea behind the project, and the steps taken accomplish the goals set for this thesis.
- **Chapter 4** - Design - Presents the design of the project. All design and architectural choices that were taken will be discussed in this chapter.
- **Chapter 5** - Implementation - Explains all the implementations in detail that were added in this thesis.
- **Chapter 6** - Evaluation - Evaluates all the new changes that were made to the original work with tests and showing the given results.

1.4 Key Concepts

- **Chapter 7** - Conclusion - Gives a conclusion of the work that was done in this thesis. Discuss what was done, what went well, what went bad, what can be done in the future.

1.4 Key Concepts

This section of the thesis will introduce some methods and technologies used in this thesis. Briefly mentioning what has been used, and will be explained in more detail later in the report. All the concepts mentioned here are closely related to the work of this project.

Starting with the technologies and methods used in the previous work that this thesis will build on, I will briefly explain what has been used followed by what I will implement.

1.4.1 Previous Stack

The previous stack is built upon this component diagram and these technologies and methods are already implemented:

1.4 Key Concepts

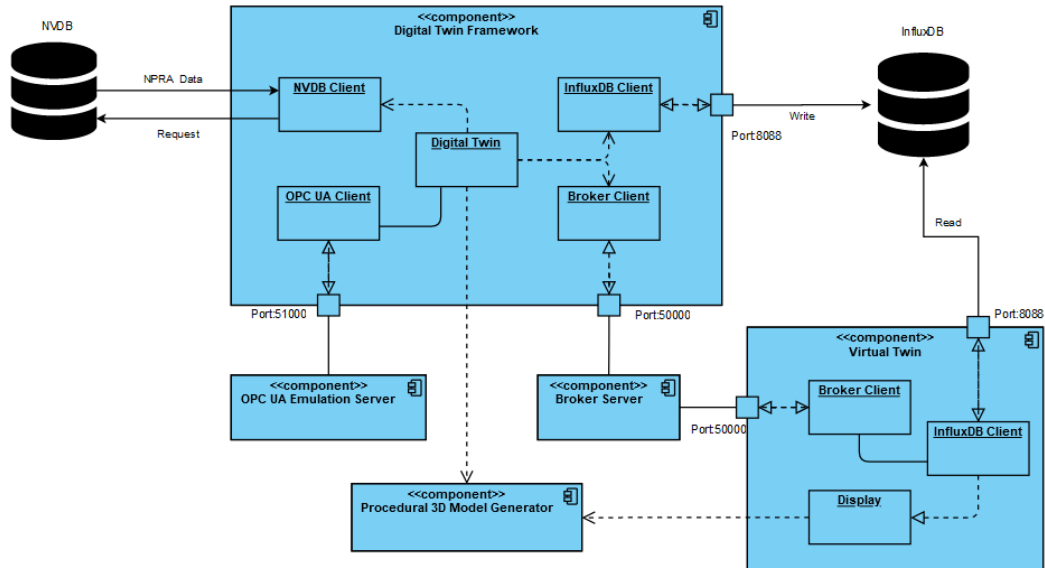


Figure 1.1: Component Diagram taken from the thesis "Digital Tunnel Twin Using Procedurally Made 3D Models" [5]

NVDB

NVDB (Nasjonal Vegdatabank) is a database containing information about all Norwegian roads and tunnels. NVDB is a free to use database by Statens Vegvesen. The application should be able to generate and run any tunnels from this database, given the tunnel id number and is dependent on information from this database.

Influx Database

The database used is the InfluxDB database, sensor data base. It is an open-source time-series database. Main use case for using InfluxDB are monitoring, analytics and sensor data [8]. The digital twin will write all the information that is happening to this database, and the data will then be

1.4 Key Concepts

read by the virtual twin which handles all the visualization for the user to see.

MQTT Communication

MQTT is a network protocol to transport messages between devices. MQTT was developed to work on any type of device, designed for cases where resource constraints can exist, or cases with limited bandwidth. It's main purpose is for reliable communication and is often used in sensor communication. It supports most types of network protocols, but since it's purpose is reliable communication, it usually will be used for TCP/IP.

MQTT uses a publish-subscribe pattern, where the sender of the message are called the publishers, and the receivers of the messages are called subscribers. The idea here is that the publishers only publish the messages without knowing who will be listening(subscribing) to them, so the messages are not tailored for a specific subscriber, but to any subscriber that wants to listen to that publisher.

Unity

Unity was originally a game engine that was mostly used for video-game development. However in the 2010s Unity began its transition into other industries as well, using it as a real time 3D platform. Unity is now used a lot in simulation projects where it's engine is used for visualization for the end user. Its non-gaming simulation and modeling are used in a lot of industries, including automotive, architecture, construction and engineering. It is also used in this project to simulate the digital twin in a visual and interactive space.

1.4 Key Concepts

1.4.2 Stack to be Implemented

Apache Kafka

Apache Kafka is an open-source software platform which is used for handling real-time data feeds. One of the things Kafka provides is a publish-subscribe pattern like MQTT, however Kafka is a software platform that also provides many other services than only the publish-subscribe pattern. Kafka focuses on storage and reading of data and data streaming processing scenarios in real-time with high performance and high scalability.

Grafana

Grafana is an open-source analytics and interactive visualization web application. Grafana is a popular component when it comes to monitoring stacks, and is often used in combination with time-series databases. Grafana's use is to provide charts, graphs and alerts when connected to a time-series database [9]. Users can create monitoring dashboards as a visualization tool to track their monitoring data.

Chapter 2

Background

2.1 Related Work

In this section I will talk about previous academic work that will present an overview of the research that has been done on digital twins. Specifically digital twin work related to tunnel systems and sensor communications. The academic work mentioned here has been used as inspiration for this thesis.

The digital twins have so far had a brief history and have only been around for closer to twenty years. Digital twins first entered the academic world in 2003 by Michael Grieves, where the concept was introduced as a product life cycle management system for manufacturing [22]. Digital twins were theorized earlier, and anticipated in 1991 by David Gelernter [23]. Digital twins got its name in 2010 when it was shown on NASA's road map report, and has been the usual name for this technology since [24].

Digital twins rely on getting information from the physical world, usually through sensors, to be able to display a realistic digital view. Digital twin technology was probably a little premature when it was first introduced given that it relies on IoT(Internet of Things) and good networking for all the data to be displayed in real time. The last ten years, IoT has had an exceptional growth in the industry, paired with increasingly good internet

2.1 Related Work

connections, which makes digital twins now more relevant than ever.

2.1.1 Digital Twins

Digital Twins were first introduced as a product life cycle management system where it was used to simulate manufacturing, but has since then been adopted by many different industries.

Examples of Digital Twins can be found in city planning and construction industry. Where Digital Twins can simulate parts of a city to plan out with construction work. With the upcoming of smart cities, where the use of sensors for every part of the infrastructure is used, we can see the potential of creating entire cities as digital twins. In smart cities sensors can gather data collected from citizens, devices, buildings and assets that is processed and analyzed to monitor and manage traffic and transportation systems, power plants, utilities, water supply networks, etc. to get vast information that can be used to simulate the city[2]. In the UK, Centre for Digital Built Britain(CDBB) are working on a connected network of Digital Twins platform, to connect the UK's digital twins to work together[3].

In healthcare we can find that use of Digital Twins can be used in almost any part of the industry, from simulating the operations of a hospital and risk management, to Digital Twins of human bodies or healthcare applications. In the paper "Digital Twin for Intelligent Context-Aware IoT Healthcare Systems" a Digital Twin framework has been proposed to detect ECG heart rhythms to diagnose heart disease and detect heart problems[4].

2.1.2 Digital Twins Architecture

In 2018, Fei Tao et al. [1] went through the academic papers about Digital Twins from 2003 to 2015. This paper reviews DT applications in industry from this time span. To understand the development that has been done on Digital Twins and their application in industry, this paper thoroughly reviews the state-of-the-art of Digital Twin research concerning the key components of Digital Twins, the current development of them, and their major applications in industry [1].

2.1 Related Work

The research and development of DTs has rapidly increased in the last few years, so given the last papers from this review is from 2015, the tech may be a little outdated. However, what is still being used today from this paper, is the way of thinking of the DT architecture in forms of either three or five layers. In this paper, they introduced the idea of thinking of Digital Twins as a five layer architecture instead of the previous three layer architecture implemented by Grieves.

In 2020, Wolfgang Kastner et al. [10] reviewed concepts, architectures, and frameworks for Digital Twins in literature in order to develop a Generic Digital Twin Architecture(GDTA) based on five layers of a digital twin which was not dependent on any technologies [10]. The layers are Asset, Integration, Communication, Information and Functional. Asset, being the physical entity of the digital twin, how the sensors are set up in the real world. Integration holds the run-time data or engineering data, which are given by the sensors, which are usually time-series data and holds the state of the physical object. Communication layer is a representation of how the digital twin is communicating with its components, the concrete protocols it is running or if it has publish-subscribe pattern support. Information layer acts as a semantic integration layer for run-time data, engineering data and historical data. Functional is the last layer of this model. The functional layer is what implements interaction with the digital twin and gives insight. Here we have the simulation, the monitoring and AI predictions etc. These five layers of architecture are no standards for how a digital twin should be implemented, but are a suggestion of the architecture of a general digital twin.

2.1.3 Cloud in Digital Twins

In the last few years, cloud computing has also had an extreme growth due to its higher availability and lower costs than what cloud computing has been offering in the past. With this growth, cloud computing has also entered the digital twin ecosystem, and the idea of connecting the digital twin directly to the cloud. The cloud can operate as a storage provider, run the computations and provide analytics.

In 2017, Alam et al. [11] released a paper on a cloud-based digital twin reference model with an approach to implement cloud in a digital twin. This

2.1 Related Work

paper explains the general use of this architecture, and with an example of how it can be used in telemetries-based prototype driving assistance applications for the vehicular domain [11].

Redelinghuys et al. [12] released in 2018 a paper on a six layered cloud-based digital twin. The Digital Twin infrastructure was meant for a manufacturing cell in Germany but could also act as a general architecture for other applications. This model is somewhat similar to the five layer architecture, except here the communication layer has been removed as a layer, and replaced with two layers, IoT gateway and Cloud-based information repositories as layers [12].

2.1 Related Work

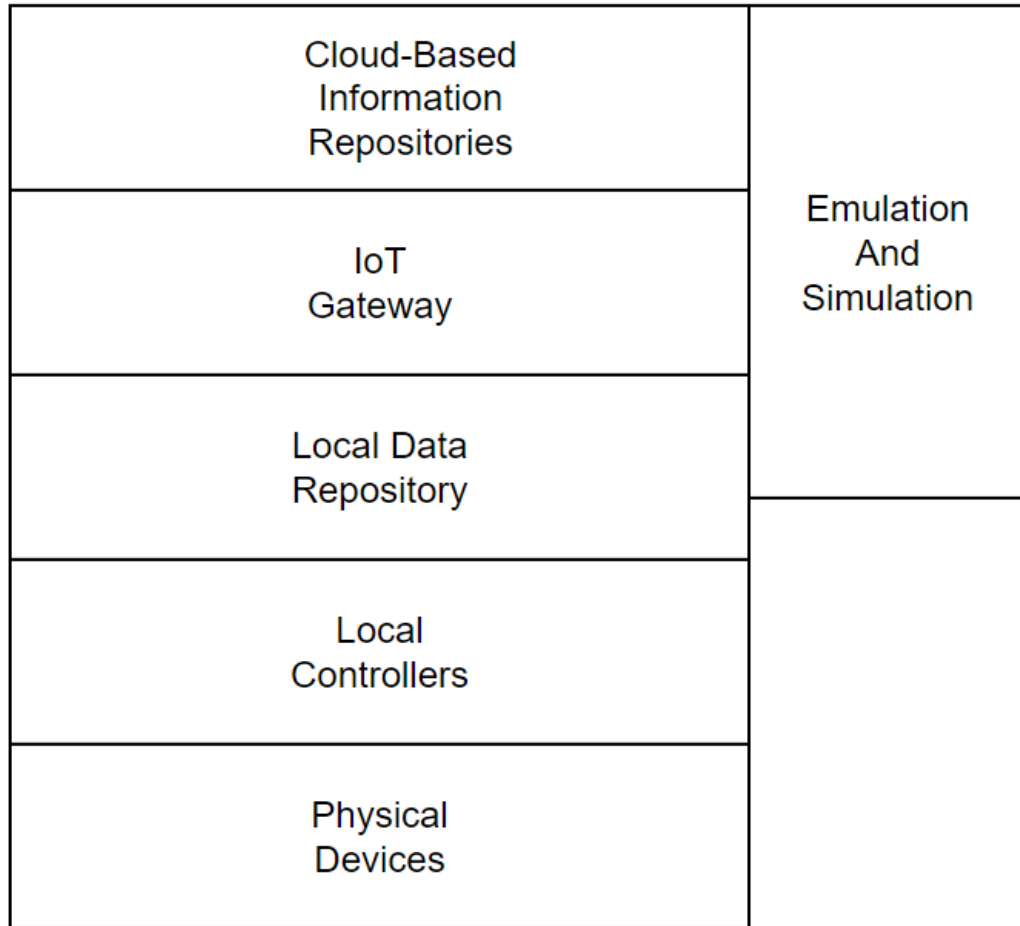


Figure 2.1: Replica of model described in the six layered cloud based Digital Twin [12]

2.1 Related Work

Table 2.1: Table Showing Cloud Based Digital Twins in Academia.

Table Showing Cloud Based Digital Twins in Academia.				
Digital Twin	Published	Industry	Layers	Cloud
C2PS: A Digital Twin Architecture Reference Model for the Cloud-Based Cyber-Physical Systems [11]	2017	General	5	Yes
Digital Twin Driven Smart Manufacturing [20]	2019	Manufacturing	5	Yes
A Six-Layer Digital Twin Architecture for a Manufacturing Cell [12]	2018	General	6	Yes
Generic Digital Twin Architecture for Industrial Energy Systems [10]	2020	General	5	No
A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems [21]	2014	Manufacturing	5	No

2.1.4 InfluxDB in Digital Twins

In order to look into if the database we have has been used for this type of work, we will look into how this database has been used in other Digital Twin projects.

Kamath et al. [13] published in 2020 a paper about using industrial IoT in

2.1 Related Work

a Digital Twin Smart Factory, by using InfluxDB as the database, Grafana as the analytics tool and Kafka as the publish-subscribe pattern. The aim was to use only open-source tools for their DT architecture [13].

Zhou et al. [14] wrote a paper in 2020 for reducing energy consumption in a iron making facility by using a digital twin for replicating the manufacturing. This paper used InfluxDB as their database for the output data of their cloud-based digital twin architecture [14].

Vering et al. [15] published in 2021 a paper about a digital twin for calibrating HVAC systems in buildings. The goal was to create a DT to increase efficiency through operational optimization and predictive maintenance of real processes [15]. The project used InfluxDB as their database, and a non-specified dashboard tool [15].

2.1 Related Work

Table 2.2: Table Showing Digital Twins in Academia Using InfluxDB

Table Showing Digital Twins in Academia Using InfluxDB					
Digital Twin	Published	Industry	InfluxDB	Grafana	Kafka
Industrial IoT and Digital Twins for a Smart Factory[13]	2020	Manufacturing	Yes	Yes	Yes
A Collaborative Optimization Strategy for Energy Reduction in Iron making Digital Twin[14]	2020	Manufacturing	Yes	No	No
Digital Twin Design with On-Line Calibration for HVAC Systems in Buildings [15]	2021	Buildings	Yes	No	No
An open source approach to the design and implementation of Digital Twins for Smart Manufacturing [16]	2019	Manufacturing	Yes	Yes	Yes
AI environment for predictive maintenance in a manufacturing scenario [17]	2021	Manufacturing	Yes	Yes	Yes
An IIoT Solution for SME's [18]	2020	Manufacturing	Yes	Yes	Yes
IoTwins: Design and Implementation of a Platform for the Management of Digital Twins in Industrial Scenarios [19]	2021	General	Yes	Yes	No

2.1 Related Work

Digital Twins in Tunnel Systems

In this thesis I will focus on the Digital Twin for tunnel systems.

A very similar project to this thesis is a project done by Jeroen van Hegelsom and published in 2021[7]. Which is a project designing a Digital Twin of Swalmen Tunnel in the Netherlands. Swalmen Tunnel is set to be renovated between 2023 and 2028, and by proving how a Digital Twin can be utilized, the project aims to support the need for more and reliable sensor systems in the renovation. This project also uses Unity and BIM models to visualize the tunnel. However this project only seeks to create a Digital Twin for this specific tunnel only, and not every tunnel from a database.

Tunnelware is a company that aims to accelerate construction by improving information flow between contractors. Tunnelware is a digital twin of a tunnel construction system that monitors everything there is to monitor about this system, through its entire lifetime[9]. The goal is to create a digital twin of a tunnel construction site which enables architects and planners to come together and work in the digital space. This in return, creates an interactive project which promotes real-time collaboration. This is not exactly what we try to do in this project, but it is in the same kind of similarity.

In a paper published in 2020 by Jinwooung Kim et al. the authors create a Digital Twin of a Noise Barrier Tunnel to predict the lifespan of the components inside the tunnel[8]. The possibility of analyzing the life and damage caused to components using a digital twin was verified, thereby proving that a digital twin can be used in the component reuse scenario via visualization[8]. This project does not aim to achieve the same goals as we are, but they are using a Digital Twin of a tunnel using sensors to arrive at a conclusion of their goal.

Related Work from UiS

This thesis is based on and will continue the work from a previous thesis written at the University of Stavanger(UiS), by Berke Kağan Nohut [5]. We will use this previous work as a basis to expand on to further develop the

2.1 Related Work

project[5]. The introduces a framework to create digital twins for any tunnel in Norway, gathered from the NVDB(Nasjonal vegdatabank) database.

Chapter 3

Methodology

This chapter will present the idea behind this project and how I did my research to come up with the ideas and solutions I will suggest in further sections.

3.1 Idea

In the current implementation of the Digital Twin in road tunnels, the application runs a few sensors and replicates them as object inside a procedural made tunnel. The idea behind my thesis is to find ways to improve on this solution in several different ways. I need to find out if the currently used architecture is the best use cases for each component, as it was not really studied in the previous thesis [5] why choices were made to choose different kinds of software or frameworks. I will also try to look at what more functionality can be added to the current implementation for it to be a better version and provide a better use of it.

3.2 Methodology

3.1.1 Academia

The main way I have been researching to find the answer to my questions is by looking through many, many, published papers online where Digital Twins are discussed. Some of the articles are very vague about implementation, but are good at explaining their use case, while other focuses more on how they are architecturally made up. By looking through these papers I get more of an idea which types of features and functionality many of the Digital Twins have, and a lot of it are the same for many of them even though they are in vastly different technical fields. That makes sense given that Digital Twins are made to replicate a real world scenario, it does not really matter what kind of scenario needs to be replicated, as long as you can feed the Digital Twin information, and monitor how it behaves.

3.2 Methodology

Here I will show the timeline I will be basing my work on. I have made a Gantt chart describing my progress. The timeline is based on milestones that will be completed in certain time frames. Gantt chart of the work is being displayed below.

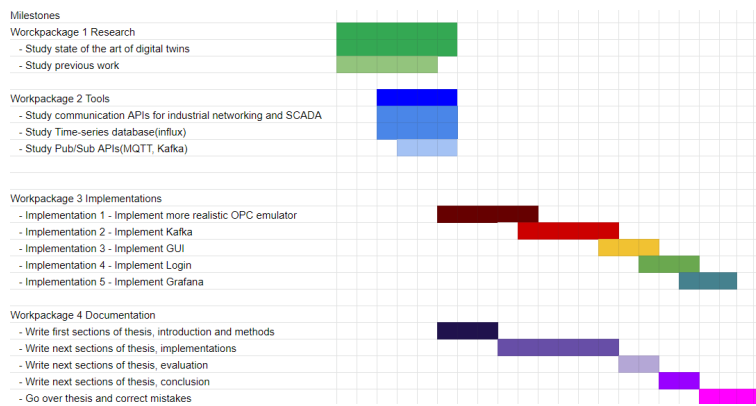


Figure 3.1: Gantt Chart

As seen in the Gantt chart, the work is represented by milestones. The thesis

3.2 Methodology

work start off in the first milestone, in Workpackage 1 Research. Basically saying that the first weeks of the thesis goes to researching the topic and mostly do the work represented in Chapter 1 and Chapter 2. Really get to know the topic and find about as much about it as possible before going in to how this information can be used. Then I move on to Workpackage 2 Tools. In this work package I learn about all the technologies used in the previous project work, and about the new ones that I want to implement. It is important that I learn how to use them to their full extent before I just throw them into the architecture.

After Workpackage 2 is done, I start on the implementation part of the project in Workpackage 3. Here I go through one implementation at the time, to be sure that I have the time to go through all five of them, instead of starting on all of them and doing little by little on each. I start on implementing the more realistic sensor emulator, as this takes big chunk of the time as seen in the Gantt chart. Next up I implement the Kafka broker, which also takes a big chunk of the time. The three next implementations are less time consuming. I then implement GUI, Login and Grafana, which all takes about the same time. At the end of the project, and also kind of overlapping Workpackage 3, I do the last work package, Workpackage 4. The last work package is all about writing the thesis to document the research, findings and the work that has been done in the project.

Chapter 4

Design

In this thesis the main objective is to explore how I can improve the previously made Digital Twin for Norwegian road tunnels. Until now, introduction and related work has been presented. This coming chapter will go over the methods used to improve different aspects of the Digital Twin and how they are implemented. The complete setup and results of these implementations will be described fully in chapter five and six respectively.

4.1 Current Architecture

As this thesis is continuing the work by a previous thesis, there is already a complex architecture in place for the Digital Twin. The current architecture is built by the components as shown in the figure below, Figure 4.1. The main part of this thesis is to find ways to improve this already established architecture, while also introducing new features to compliment the Digital Twin and improve its usefulness.

4.1 Current Architecture

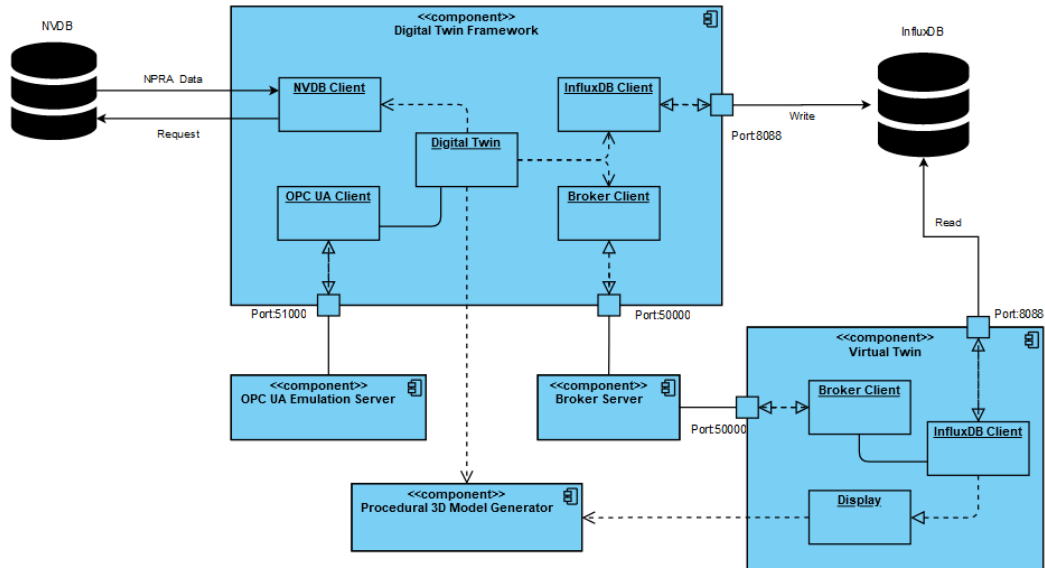


Figure 4.1: Component Diagram taken from the thesis "Digital Tunnel Twin Using Procedurally Made 3D Models" [5]

The Digital Twin framework is built up by five main components, internal communication, sensor communication, database, virtual twin and NVDB communication.

4.2 Digital Twins Layers of Architecture

- Internal Communication MQTT(Message Queuing Telemetry Transport):
 - Broker Server
 - Broker Clients
- Sensor Communication
 - OPC UA Emulation Server
 - OPC UA Client
- Database
 - Influx Server
 - Influx Client
- Virtual Twin
 - Unity Project
- NVDB
 - NVDB client

4.2 Digital Twins Layers of Architecture

If we look at the architecture of the Digital Twin by using layers of architecture, like we looked at the different Digital Twins in the related work section. We can see the architecture of a Digital Twin is usually divided into five layers. The layers can be asset, integration, information, communication and functional. If we take the components from Figure 4.1 above, we can divide them into a layer model as seen below in Figure 4.2.

4.2 Digital Twins Layers of Architecture

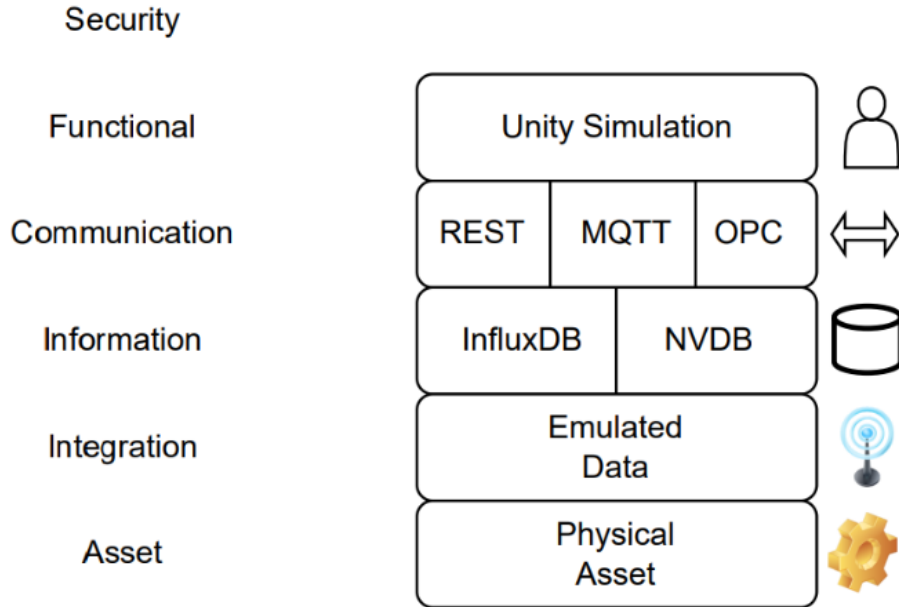


Figure 4.2: Previous Architecture in Layers

Asset layer contains the physical assets of a Digital Twin. How the real world sensors are set up. In this case we don't actually have access directly to the hardware, as we are just getting NVDB data from an API, and emulating the sensor data. Asset layer is not something we can do anything about at this point.

Integration layer contains the data we are gathering to the Digital Twin to work with. It holds the run-time data or engineering data which are given by the sensors in the tunnels. In our case, this data is emulated since we don't have access to the sensors in the tunnels. The data is emulated by the OPC emulation server.

Information layer contains all the data from all different sources of the application and how they are stored. In our case, it holds the data gathered from the NVDB API and the data we are emulating from the OPC emulation server. All this data is combined and stored in an Influx time-series database that will be read by our Virtual Twin to be shown in Unity.

4.3 Proposed features to be Added to the Digital Twin

Communication layer is a representation of how the Digital Twin communicates with its components in the application. We have three main communications in this Digital Twin, HTTP communication with NVDB API, OPC TCP communication for the emulated data, and MQTT communication between the Digital Twin and the Virtual Twin.

Functional layer is a representation of functionality that can interact with the user of the application. The functional layer is what implements interaction with the digital twin and gives insight. Here we have the simulation, the monitoring and AI predictions etc. In this application, here is where we have the Unity application where we can interact with the tunnel in a 3D space.

4.3 Proposed features to be Added to the Digital Twin

The main objective of this thesis is studying what can be done to improve this Procedural Digital Twin. Improving the Digital Twin can come in different ways, either finding ways to improve on already implemented features, and finding new features we can add to further develop the application. In this section we will look through all the features we are going to add to the Digital Twin to increase functionality.

4.3.1 Monitoring

When it comes to Digital Twins information is key. The whole idea is to have as much information about the processes and everything that is going on as possible. With the current architecture we don't really have that much information except for what we can see in Unity and the database, but it's hard to see what is going on in the database in real-time. Therefore I want to implement some way to monitor the data and have real-time monitoring of the database in a way that it is easy to observe for the user. To solve this task I have decided to add a real-time dashboard system, or interactive visualization application. For this project I have decided to go with Grafana as the monitoring application. As previously mentioned in

4.3 Proposed features to be Added to the Digital Twin

the Introduction, I have explained what Grafana is, and what it can do. There are several reasons why I have chosen Grafana. First, it is free to use, which is important for this project since it is a non-profit research project and it is important that it is not expensive to be able to run it. This also comes in handy when trying to get new people to try out this application, free to use makes it an easier choice for people to try it out for themselves. Grafana is also open-source, which is important so that we can always know what is in the program and what changes are being made to the application. And as discovered in the related work section, Grafana is often used in combination with InfluxDB, as we can see in Table 2.2. Both are free to use, open-source applications with very good connection to each other. Grafana was mainly used for time-series databases but has widen it's horizon to also be used for relational databases. But since Grafana was first mainly made for time-series databases, it was focusing on being easy to use for time-series databases, and there is a lot of configurations made specifically for InfluxDB, which makes them a good combination. Adding monitoring would add a feature to the functional layer, since it gives more functionality for the user in the Digital Twin.

4.3.2 Security

It is desired to add some kind of authentication system to the Digital Twin. Right now there is no confidential information or specific reason to hinder usage of the Digital Twin, but this project is continuously being developed for future use and future use cases so having authentication system is needed in the future. Right now an authentication system will stop people with unauthorized access to misuse the NVDB API from being used a lot without purpose and for people without authorization to use the features we have added to this project. For now we will add a simple authorization system that just stores the users name, username and password but in the future there might be more information added to the user accounts, for example, remember user setting and configurations for future use, or the accounts may be used in the future to be used against a cloud solution for the project. The optimal solution would be to host the authorization application and database on the UiS servers and send calls to this web server. But since I don't have access to this system and getting that setup and authorized would take a long time, I have decided to just have the database locally for now and work more as a temporary concept for now. I have decided to

4.3 Proposed features to be Added to the Digital Twin

use SQLite as the database for this authorization. SQLite is an embedded database that follows PostgreSQL syntax.

Adding this authorization system to the Digital Twin will add a feature to a sixth layer of the architecture, security layer. Usually Digital Twins operate with five layers to describe the architecture, but when adding authorization or cloud connection, they often are added as their own layer. Here we will add it as the sixth layer.

4.3.3 Graphical User Interface

With the current build, the Digital Twin is somewhat clunky to use and get started with. You have to open three different command windows and run the correct python files with the correct parameters and using the same tunnel ID on all of them. A suggested implementation is to add a GUI to run the Digital Twin from, where we can fill in the parameters once and run all python files in the correct order from. To create the GUI I have chosen to use the python package Tkinter which is a standard GUI library for Python applications. Adding GUI to the application would add a feature to the functional layer since it gives the user an easier way to use and configure the Digital Twin.

It is also important to consider that one of the goals of the research team working on Digital Twins at UiS is to get people to use it. If a program is very hard to use and to get people to try it, it is going to be hard to convince new people to try it. Adding a GUI which decreases a lot of the steps to get the Digital Twin going, and at the same time looks more appealing to use than running a lot of commands, might be more appealing for people to try it out.

4.3 Proposed features to be Added to the Digital Twin

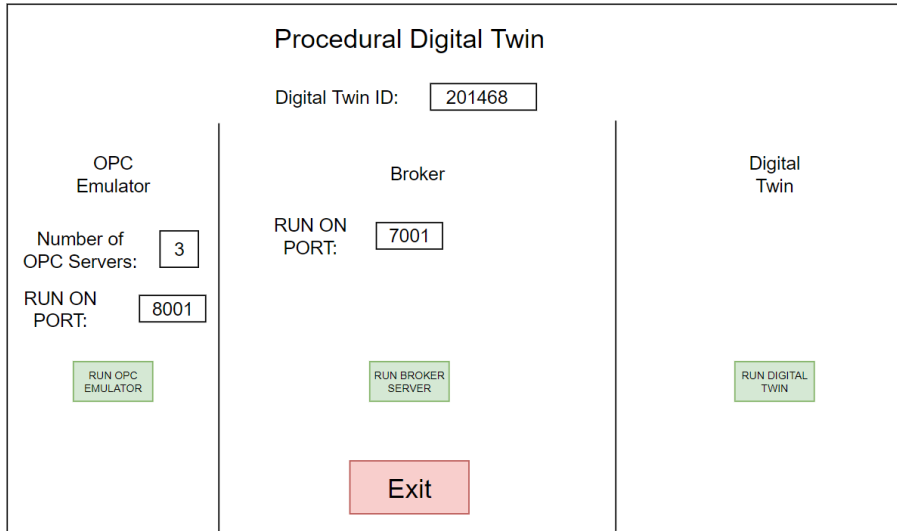


Figure 4.3: GUI Wireframe

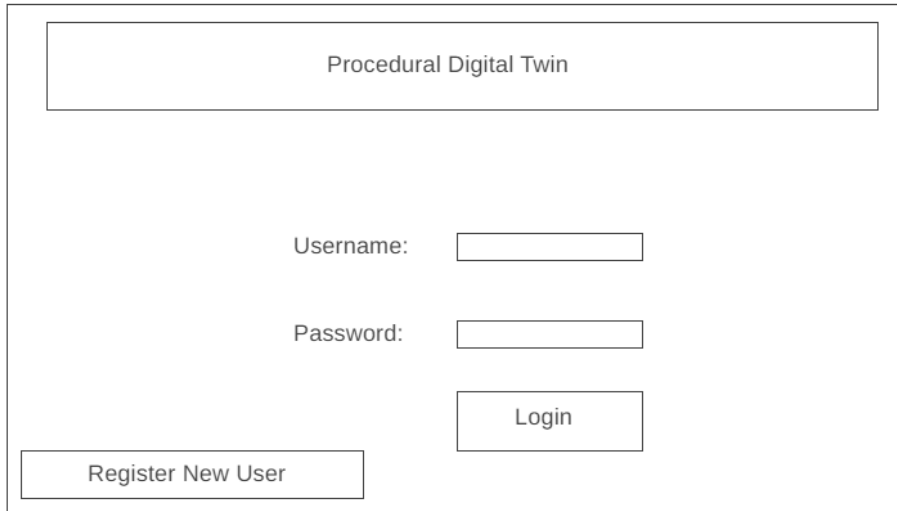
In Figure 4.3 above is a suggested design for how the GUI will look like, and what I will try to replicate when I design the GUI part of the application. It will have an input at the top for the Digital Twin ID, so we don't have to rewrite it several times when running all the python files. The OPC Emulator will have its own input for the number of OPC Servers should be deployed and what port to communicate on. The Broker will only have an input for which port to run on. The Digital Twin will only have the general input for tunnel ID. All three of them will have its individual button to run each part, since they need to be run individually in a specific order.

We will also have GUI for the login system discussed previously. So the first thing that meets the user when launching the program will be the login screen, in which the user will have to login to continue to the Digital Twin GUI where the user can run the applications. The GUI for the login system will have a input field for username and password, and a login button which will confirm the input fields to be correct before it sends the user further.

It is also necessary to have a sign up screen for new users to create an account that they can later login with. This screen will have fields for users to enter the username they want and the password they want for that account and a register button. In the figures below, Figure 4.4 and 4.5,

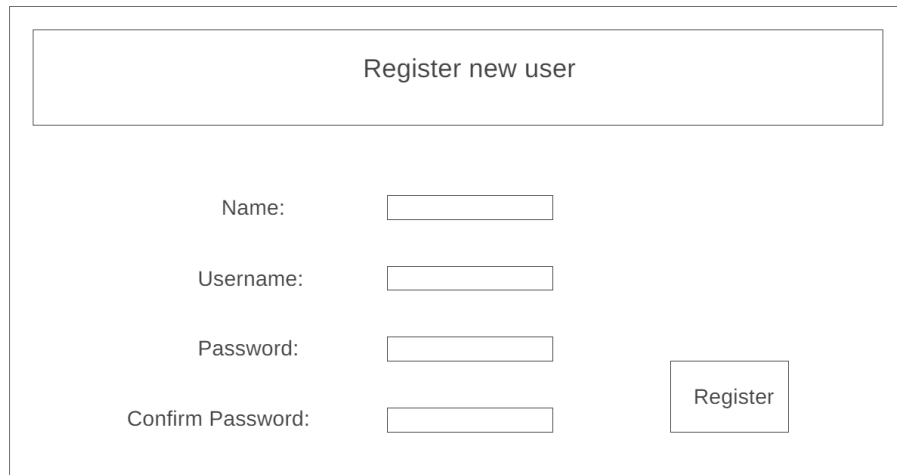
4.3 Proposed features to be Added to the Digital Twin

are a suggested layout for the login screen and sign up screen.



The diagram shows a login screen layout. At the top, there is a header box containing the text "Procedural Digital Twin". Below the header, there are two input fields: "Username:" followed by a text box, and "Password:" followed by a text box. Below the password field is a "Login" button. In the bottom left corner, there is a "Register New User" button.

Figure 4.4: Login Screen



The diagram shows a sign up screen layout. At the top, there is a header box containing the text "Register new user". Below the header, there are four input fields: "Name:" followed by a text box, "Username:" followed by a text box, "Password:" followed by a text box, and "Confirm Password:" followed by a text box. To the right of the "Confirm Password" field is a "Register" button.

Figure 4.5: Sign Up Screen

4.4 Digital Twin Areas of Improvement

4.4 Digital Twin Areas of Improvement

In this section I will look through all layers of architecture from Figure 4.1 and see which section could get an improvement or new feature. Since I can't really do anything about the physical layer, I will skip that one for now.

4.4.1 Integration Layer

The Integration Layer is currently containing the emulation data from the OPC Emulation server. The only data that I are interesting in using at the moment, are data from NVDB and OPC Emulated data, so here is not really anything I want to add at the moment. However, there are things that I can change or improve on from the features that are previously implemented.

Emulated Data

The OPC UA Emulation Server currently creates all the emulated sensor data, and sends it over to the Digital Twin through OPC communication. This implementation skips the steps that sensors are creating the data, sending it to OPC Servers, then sending them all to the same application, and then sending them to the desired user of this data. The OPC UA Emulation Server is just an application that generates all this data and sends it over. I want to create an emulation server that is more realistic to how it is done in the real world, creating each sensor that will send information on its own with the configurations specified in Statens Vegvesen. It is also desired to be able to specify the number of OPC Servers in the tunnel that gathers the information from the sensors. Current implementation assumes only one OPC Server in the tunnel.

4.4.2 Information Layer

The Information Layer is the layer that is responsible for the information being contained. In this case it is where I store the emulated data and the

4.4 Digital Twin Areas of Improvement

data I gather from NVDB. In the current architecture NVDB is stored at Statens Vegvesen and I gather it with an API and use it to create tunnels. Emulated data is sent to the Digital Twin, processed and sent to an InfluxDB database. There are not really anything to add here at this point, since I do have the types of data I want. However, as in the Integration Layer, there are things to improve on.

NVDB

When it comes to the NVDB data, there is not a huge change I want to make. The change that I would like to implement is updating the API from NVDB API v2 to NVDB API v3. NVDB API v2 stopped being support by August 2021, so it will no longer receive updates. This is the version currently being used in this application. Therefore it is desired to update it to the newest version, which is NVDB API v3. The structure will mostly be the same, with some new elements, and continuous updates.

InfluxDB

The question in this section will be to find out if InfluxDB is a viable database for this kind of project. In the current implementation, the choice of database was not really justified, and since there is a lot of databases in the world, I will take a look into if this database is a good choice for this kind of project.

First I will look into some of the functional reasons for why to choose InfluxDB. Cost, InfluxDB is a free to use database, which as stated earlier, is very important in this project. This project need to be as cost effective as possible since it is not a commercial product that makes money. A free database is very important. Secondly, InfluxDB is a open-source application, as stated in the Monitoring section 4.3.1, open-source is important to always have control of what is actually in the application.

Time-series databases are the best when I have data that are continually capturing metrics combined with a timestamp for analytical purposes. Relational databases are usually for general use and not optimized for time-series

4.4 Digital Twin Areas of Improvement

data and tend to be slower at inserting and retrieving time-series data [25]. However, relational databases are more flexible for other cases and have more use cases. But, as I will only use this database for one specific thing, and that is gathering sensor data for metric use, combined with a timestamp, a dedicated time-series database looks to be the optimal choice [25].

However, there is a lot of different time-series databases, and I need to figure out if this one is the best for this use case. It is important to know if the selected database is used a lot by other programmers. This will give a sign that it is a popular database because how well it is made, and by picking a widely used database, it will be easier to find people that has vast knowledge using this database. In the table below, Table 4.1, I have gathered data from www.db-engines.com [26]. This data shows a ranking of popularity from all time-series databases. Popularity ranking are based of many different metrics, some being number of search queries on Google and Stackoverflow [26]. For a full method of ranking the scores, go to https://db-engines.com/en/ranking_definition [27]. The table below shows the top 10 time-series databases based on popularity as of June 2022 [26]. Table 4.1.

4.4 Digital Twin Areas of Improvement

Table 4.1: Table Showing Time-Series Databases based on popularity from db-engine.com

Table Showing Time-Series Databases based on popularity from db-engine.com			
June 2022 Rank	June 2022 Score	Database	Type
1.	29.86	InfluxDB	Time-series
2.	9.12	Kdb+	Time-series
3.	6.32	Prometheus	Time-series
4.	5.35	Graphite	Time-series
5.	4.56	TimescaleDB	Time-series
6.	2.94	Apache Druid	Time-series
7.	2.43	RRDtool	Time-series
8.	1.86	OpenTSDB	Time-series
9.	1.65	DolphinDB	Time-series
10.	1.33	Fauna	Time-series

Even though this is a very popular database, and fits the use case of this project, it is important to find out how well it compares to other databases on a performance level. It is not easy to find a well made performance metric test that use the same test for all the databases to consider, however

4.4 Digital Twin Areas of Improvement

I did find a test used for six different time-series databases using the same test and metrics. Below is a table, Table 4.2, showing the scores of metrics test done by the InfluxDB team, so this test might be a test to show off specifically where InfluxDB does the best. However it does show that in this specific case, InfluxDB does have best performance, and it is a case that is similar to the own in this project. The benchmark test focused on a system that models a common DevOps monitoring and metrics use case. The system had a fleet of servers that were monitoring and reporting application metrics at given intervals. The data set had 100 servers with 100 values measured per server, on 10s measure interval, over a period of 24 hours. The experiment gathered a total value of 87M samples of data per day. The exact same experiment were done on all the databases in Table 4.2 below. The benchmark scores tested write throughput, on-disk compression and query performance. Write throughput had a bulk load performance of the 24-hour data set for 100 hosts with four concurrent writers, here the higher score is the better performer. Table 4.2 shows how many writes are written to the database per second. On-disk compression data represents 24 hours of metric for 100 hosts, here the lower score is the better performer since it shows the amount of space used for the given data. Query performance shows the maximum value across random 1-hour intervals, grouped by minute, here the higher number is the best performer as it shows amount of queries per second[28, 29, 30, 31].

Table 4.2: Table Showing Benchmarks Tests for Several Time-Series Databases

Table Showing Benchmarks Tests for Several Time-Series Databases			
Database	Write Throughput	On-Disk Storage	Query Performance
InfluxDB[28]	2.674.948/s	155MB	925/s
Graphite[28]	180.606/s	1120MB	92.8/s
Splunk[29]	161.578/s	3146MB	37/s
Elasticsearch[30]	702.825/s	1400MB	120/s
Cassandra[31]	625.711/s	375MB	275/s

4.4 Digital Twin Areas of Improvement

In all of the benchmark tests done on these databases, I can see that the InfluxDB outperforms the other databases it was tested on by a very good margin. Since it was not tested on all time-series databases, just a few specific ones, I can not guarantee that it is the best performing database, but I can safely say that it does perform a lot better than many of them, and it has a similar use case to this project, monitoring sensor data.

By now it has been established that InfluxDB seems like a pretty good choice for this project. It is convenient when it comes to cost and open-source, it seems like a very good performer as I talked about in the previous paragraph, and by looking at the Table 2.2 in Chapter 2 it is shown to also be a popular database when it comes to Digital Twins. By the research done by now, it does seem that InfluxDB is a very good database for this project, and it will not be necessary to change the architecture to be using a different database.

4.4.3 Communication Layer

The Communication Layer is the layer that is responsible for the way information travels in the application. In the current architecture, I have three different type of communications, REST, MQTT and OPC.

REST is the communication used to talk to Statens Vegvesen and gather the NVDB data. Since I can not change this API, this communication can not be changed in any way and will remain the same. Using OPC communication to talk to the sensors in a tunnel is the way Statens Vegvesen is doing it now in the real world, so changing this would decrease the realism of the project, so this communication is going to remain the same.

MQTT

MQTT protocol is the pub/sub system that is currently being used to notify the Virtual Twin whenever there is an update in the database so that the Virtual Twin can update itself. By researching the architecture in the Digital Twin, I believe that MQTT is not the best use for this, and I should be looking into changes to this pub/sub system.

4.4 Digital Twin Areas of Improvement

As many sources are referencing, MQTT protocol is the pub/sub system you want to use when you are working with low power machines that needs a as lightweight as possible application to transfer the data [32, 33, 34]. For example if you are places with bad network connection or low compute power, MQTT seems like the pub/sub system to use [32, 33, 34]. While Kafka is to be preferred in situations with high power, and stable connection [32, 33, 34]. As the pub/sub system are used between the Digital Twin and the Virtual Twin, I can be sure that it will always be a high power and stable connection so it does seem like Kafka is a more natural choice to go for. Kafka is made for high throughput and high availability.

RabbitMQ is also an option to look into when it comes to pub/sub systems. However, by the research I have done so far, RabbitMQ is more of a niche pub/sub system, when special routing needs or special file types are needed to be sent[35]. Kafka seems to be the best option for high availability and best throughput, and RabbitMQ are better for special routing and special file types[35].

4.4.4 Functional Layer

In the Functional Layer is where the Unity feature is placed. Unity let the user go around in a virtual representation of the Digital Twin and interact with objects. This feature is the core of the project, and will not be replaced or improved upon architecture wise. However I will look into features to add to compliment the application to gain additional functionality to the project. Here two sections previously mentioned comes in. I will do no changes to what has been previously made, but it is here I will add monitoring and GUI to the functional layer of the architecture.

4.4.5 Security Layer

Usually Digital Twins are represented as layers of five as I have talked about several times earlier in this thesis, however when applications have either cloud or security implemented in them, they usually add another layer, the sixth layer. In this case I have added the security layer to make it a six layer application and added the login system in this layer.

4.4 Digital Twin Areas of Improvement

4.4.6 New Proposed Architecture Layer Model

By now I have gone through several changes to be made to the architecture of the Digital Twin, whether it is adding new functionality with new features, or changing what has been made previously with the current implementation. With that said I have made a proposed six layer architecture model as seen in the figure below, Figure 4.6, where all the proposed changes have been added. The core of the architecture will remain the same, with some changes added. In the next chapter I will go through how I will implement them to the application.

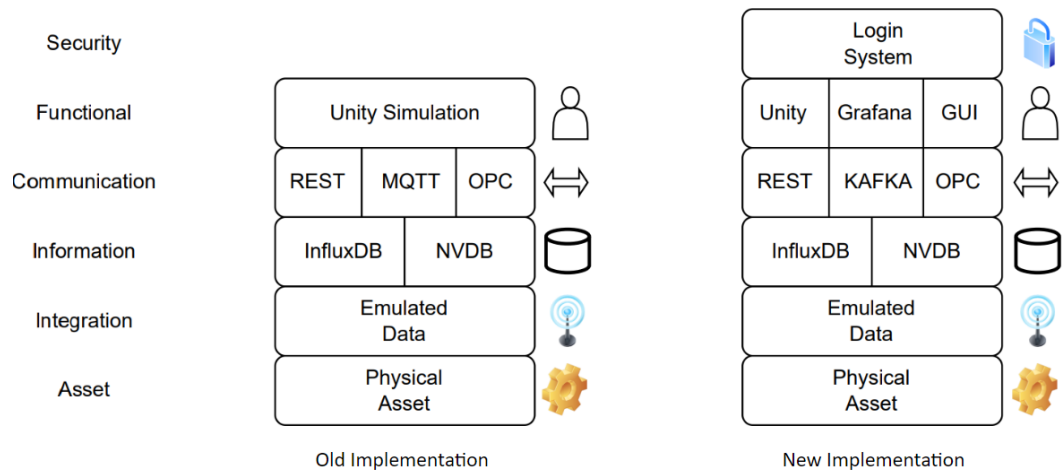


Figure 4.6: Current and Proposed Layer Architecture

Chapter 5

Implementation

Until now I have talked about what changes and implementations can be done to alter the Digital Twin to make it a better application to use. I have talked about the design and how it should be altered. In this chapter I will talk about how these implementations will be implemented and how they are going to be working. I will only go over what I will implement, and not all the components of the Digital Twin. Parts of this section might be accompanied by diagrams and charts to further provide understanding. I will start with the individual parts that will be added, and go through sections that was changed last. Instead of talking about the current architecture or implementation, it will now be referred to as the previous architecture or implementation.

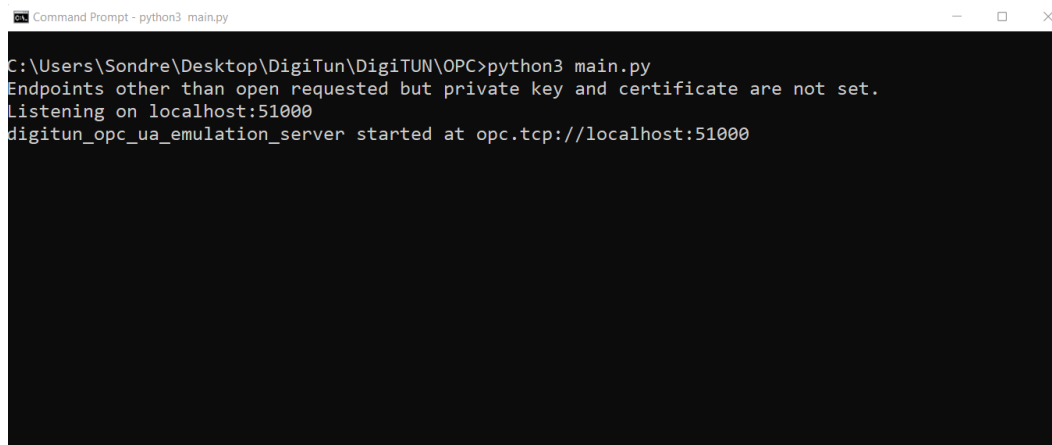
5.1 GUI

In order to increase the functionality and the ease of use of the Digital Twin, the GUI was implemented. This GUI eliminates the need for the user to open up three different command windows and run the specific python scripts in the desired order with desired parameters that you have to repeat several times.

In the previous implementation the user would first need to locate the OPC

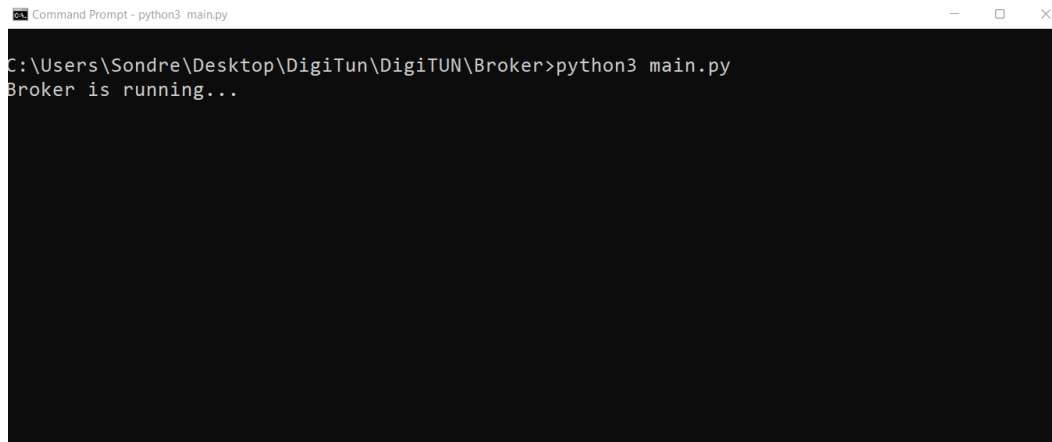
5.1 GUI

project files and with a command window, run "python main.py". This would start up the OPC Emulator. Then the user would need to locate the broker project files and with a command window, run "python main.py". This would start up the MQTT broker. Next the user would need to locate the "digitwinframework" project files and with a command window, run "python main.py <tunnel id>". The tunnel ID for the OPC Emulator was also hard coded into the code, making the user have to go inside the code and change tunnel ID. Figures below show the process of running the previous implementation. Figure 5.1, 5.2, 5.3.



```
Command Prompt - python3 main.py
C:\Users\Sondre\Desktop\DigiTun\DigiTUN\OPC>python3 main.py
Endpoints other than open requested but private key and certificate are not set.
Listening on localhost:51000
digitun_opc_ua_emulation_server started at opc.tcp://localhost:51000
```

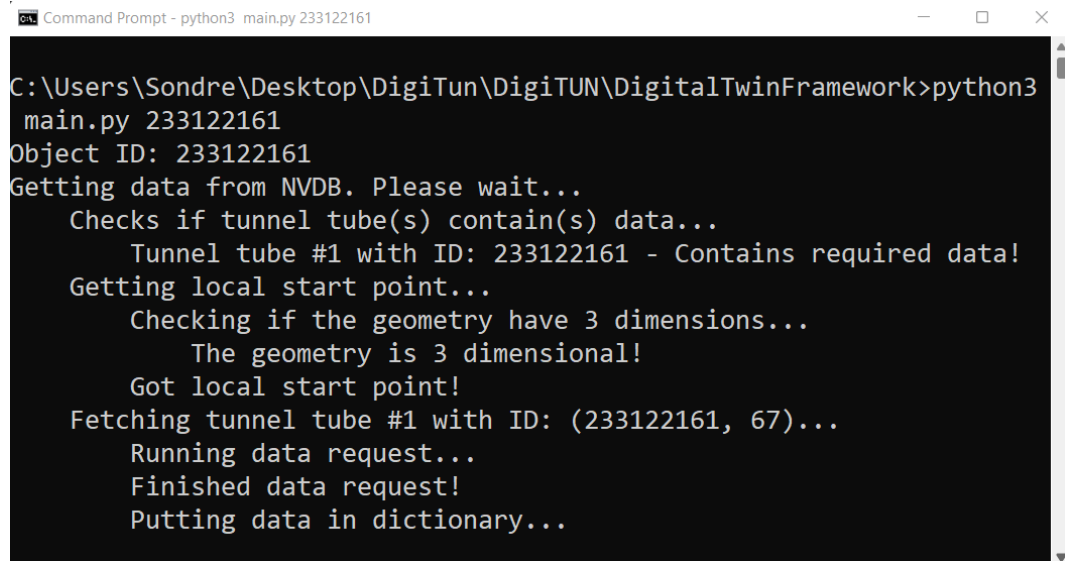
Figure 5.1: Running the OPC Emulator



```
Command Prompt - python3 main.py
C:\Users\Sondre\Desktop\DigiTun\DigiTUN\Broker>python3 main.py
Broker is running...
```

Figure 5.2: Running the Broker

5.1 GUI



```
Command Prompt - python3 main.py 233122161
C:\Users\Sondre\Desktop\DigiTun\DigiTUN\DigitalTwinFramework>python3
main.py 233122161
Object ID: 233122161
Getting data from NVDB. Please wait...
  Checks if tunnel tube(s) contain(s) data...
    Tunnel tube #1 with ID: 233122161 - Contains required data!
Getting local start point...
  Checking if the geometry have 3 dimensions...
    The geometry is 3 dimensional!
  Got local start point!
  Fetching tunnel tube #1 with ID: (233122161, 67)...
  Running data request...
  Finished data request!
  Putting data in dictionary...
```

Figure 5.3: Running the Digital Twin Framework

By developing a GUI based on the proposed design Figures in chapter 4, I have developed a GUI that can run all these three applications from the same window, and removing the need to write in the same parameter several times, for example the tunnel ID. Figure below shows a screenshot of the GUI that I have developed with Tkinter in Python. Figure 5.4.

5.1 GUI

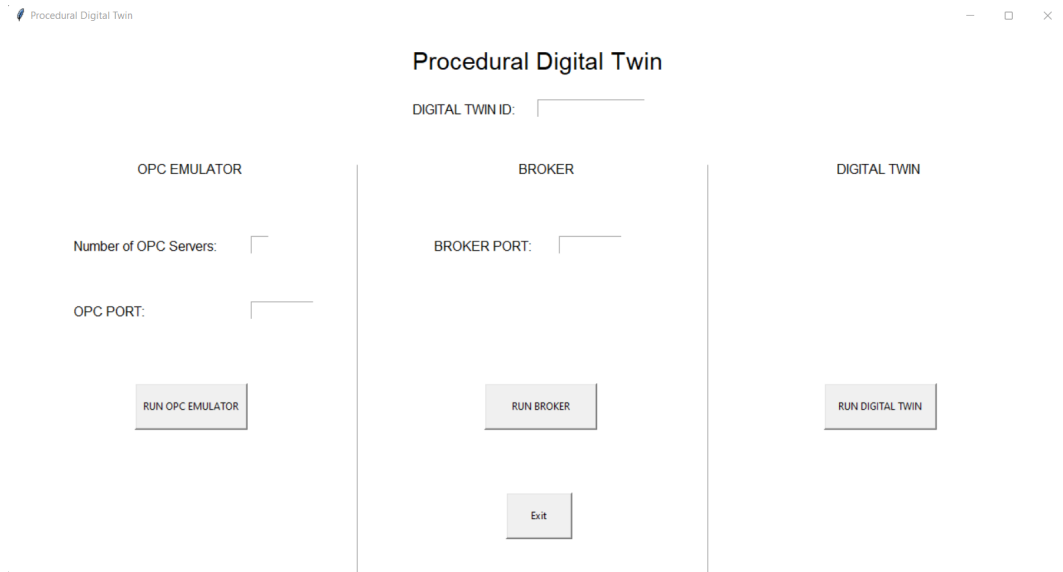


Figure 5.4: Screenshot of the GUI main screen

In this main screen of the GUI, we have four main sections. The top part of the screen is the title of the application, and just below it are the input field for inputting the tunnel ID. To the left we have the OPC Emulator section. Here we have two input fields, where I can input the number of desired OPC Servers and choose the wanted port to run the OPC Emulator on. In the middle section is the Broker section, here the user can input the desired port for the Broker application to run on. To the right we have the Digital Twin section, which does not have any input fields, it only requires the tunnel ID input from the top section.

The three sections, OPC Emulator, Broker and Digital Twin has their own run button. When the run button for the OPC Emulator is clicked, it will take the input from the tunnel ID, Number of OPC Servers and OPC Port, and will with these inputs run a command window that runs the OPC Emulator with these specific parameters. When the run Broker button is clicked, it will take the input from the Broker Port input only, as the broker does not need the tunnel ID to run. When the button for the Digital Twin is clicked, it will take the input from Broker Port and the tunnel ID, open a command window, and run the Digital Twin with these specific parameters. The button on the bottom which states "Exit" will exit the application when

5.2 Login System

clicked.

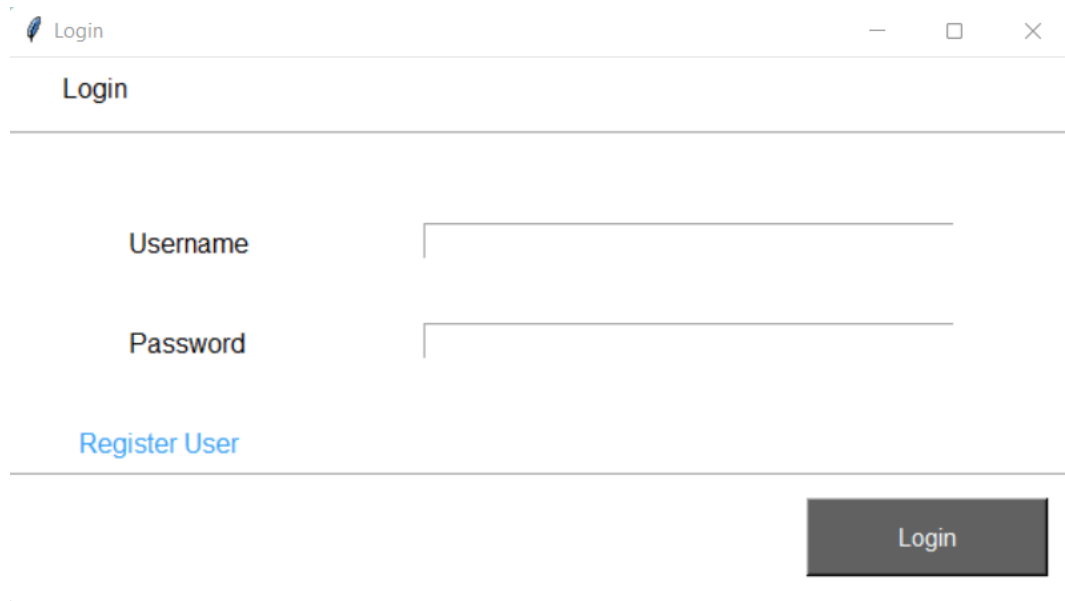
With this logic in place for the GUI, it is much easier for the user to start running Digital Twins without opening a lot of command windows manually and locating each specific file in different directories to run. This will also make it faster to run several Digital Twins with the same configurations, or small changes. The goal of this implementation is to make the life easier for the user, and increase that chance that a user will use this program or that it is so intuitive that a user will continue to use it.

5.2 Login System

The Login System is designed with a GUI part similar to the previous section. It is designed with Tkinter in Python so users can log in by plotting in inputs in the GUI. From the GUI the user can both log in and create an account to log in with. Given that there is no sensitive information and the log in system is mostly made for testing purposes at the moment, anyone can register as a new user without needing to have admin rights. This makes the project easier to work on without every person involved will need to contact an admin to have their rights given. The main idea of adding this login system is to show more ways to add functionality to a Digital Twin.

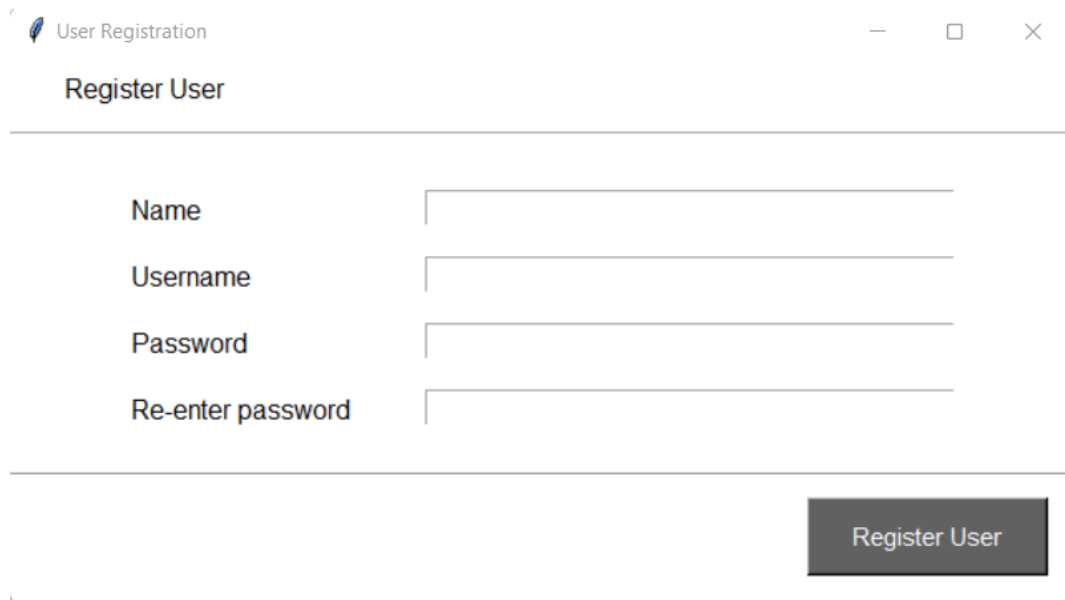
The first thing that meets the user after running the application, is the login screen. Here the user can enter their username and password to log in and get to the main window as explained in the previous section. Also on this screen is the option to register a user, which takes the user to a new window, where a user can be registered. In the Figures below, you can see how these two screens looks. Figure 5.5, 5.6.

5.2 Login System



The screenshot shows a window titled "Login" with a feather icon on the left and standard window controls on the right. Below the title bar, the word "Login" is centered. There are two input fields: "Username" and "Password". Below these fields, the text "Register User" is displayed in blue. At the bottom right, there is a dark grey button labeled "Login".

Figure 5.5: Screenshot of the GUI login screen



The screenshot shows a window titled "User Registration" with a feather icon on the left and standard window controls on the right. Below the title bar, the text "Register User" is centered. There are four input fields: "Name", "Username", "Password", and "Re-enter password". At the bottom right, there is a dark grey button labeled "Register User".

Figure 5.6: Screenshot of the GUI register screen

5.2 Login System

Since the optimal solution would be to create a login system that ran on the University of Stavanger servers with a database and an API system, the system implemented here is very easy and lightweight since it most likely will be changed when getting a hosting system. Also the current Digital Twin does not store any data that is user specific as of now, so the database that connects to the login system will only hold username, name and password for now. The database is a SQLite database running locally on the computer, so users who does not have the database on their computer can not use it to log in as of now. As mentioned, this is more to show a concept as of now since I do not have hosting available, so it is a pretty naive implementation. Table 5.7 showing user data table.

<i>user_info</i>
Name
Username
Password

Figure 5.7: SQL Table for user data

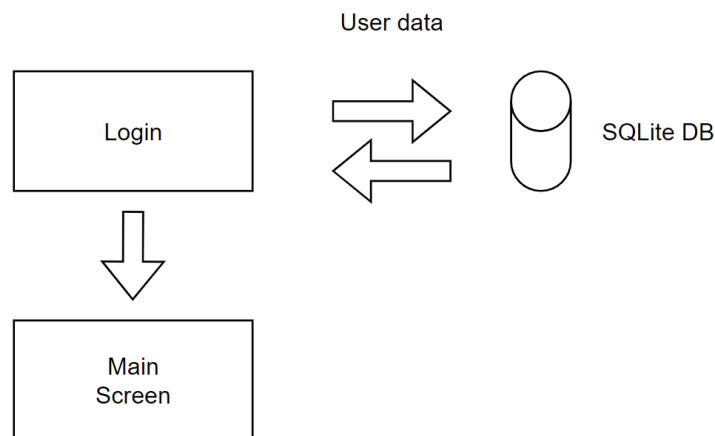


Figure 5.8: Database flow

5.3 Realistic OPC Emulator

Figure 5.8 shows the simple flow of the login system. In the Login screen the user will call the database to see check if the username and password match. If there is a match the user will be sent to the Main screen where the user can start working with the Digital Twin.

5.3 Realistic OPC Emulator

With the current implementation of the OPC Emulator, the emulator generates sensor data, and then sends it directly to the Digital Twin. But I wanted to create an implementation that is more realistic to the real world, where each sensor generates its own data, sends it to an OPC Server, which then sends it to an OPC Mainserver, which then can send it to be used, like a Digital Twin.

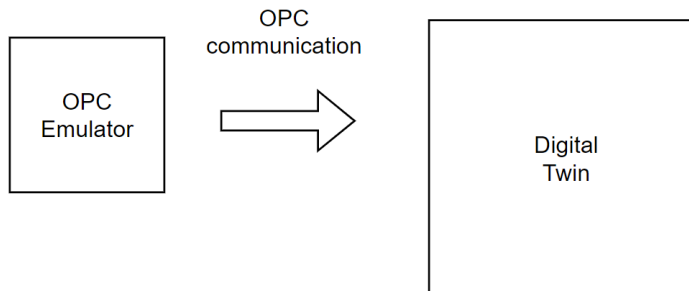


Figure 5.9: Current OPC data flow

Figure 5.9 shows how the current implementation works, very simple one step communication. With the new implementation I have developed there is a much more realistic approach to the communication. The data generation is being done in the exact same way as the previous implementation, except for that one program making all sensor data, it makes only the sensor data for itself. The data is then sent to an OPC Server, which is sent to an OPC Mainserver, which is then sent to the Digital Twin. This may create a little latency for the Digital Twin to receive its data, but it is important to create an implementation that is as realistic as possible. This approach also gives more power over each individual sensor, so it is easier

5.4 Kafka Broker

to create different working sensors that does not have to be created by a generic program that generates them all. Sensors will now also be in charge of their own OPC connection and can be found in the network as a single sensor instead of the entire OPC Emulator. With the new implementation I can also specify the amount of OPC Servers within a tunnel, since this can change from tunnel to tunnel in the real world.

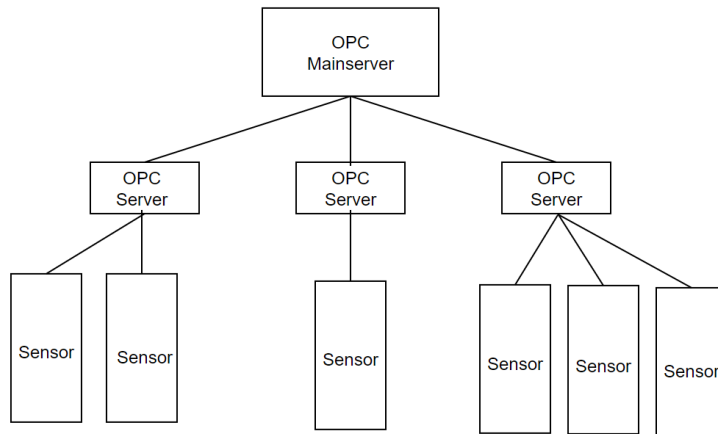


Figure 5.10: Figure showing example of OPC Mainserver with OPC Servers connected and sensor

5.4 Kafka Broker

The current implementation are using MQTT protocol as the pub/sub system to notify the Virtual Twin whenever the database has been updated with new data so the Virtual Twin can send a request to the database to get the newest data. In this implementation implemented here, MQTT protocol will be replaced with Apache Kafka as the pub/sub system.

5.4 Kafka Broker

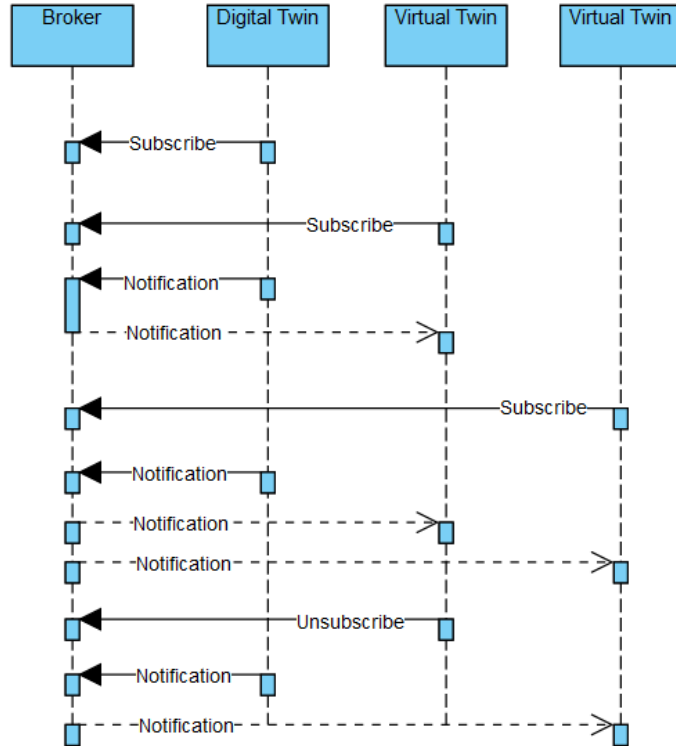


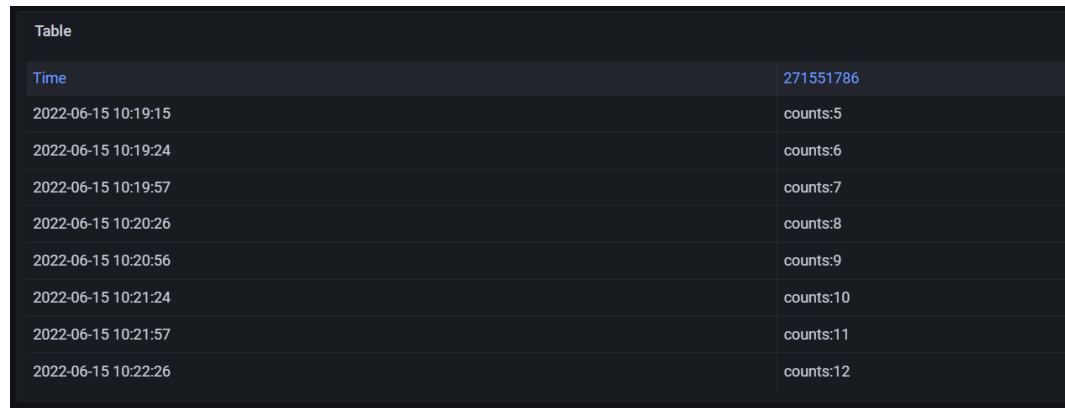
Figure 5.11: Publisher subscriber with Broker, taken from Nohut thesis [5]

The flow of the pub/sub system will work exactly like the pub/sub system with MQTT. As seen in Figure 5.11, it is the same model as MQTT used in Nohut's thesis [5]. The changes to be made are to remove the Broker Client and Broker Server from the application, and instead replace them with running Kafka in the command window. Kafka comes prebuilt with tons of functions and libraries to ease the connection and usage of the framework, so a client and server class are no longer needed. Kafka will implement its pub system in Python and Sub system in Unity.

5.5 Monitoring

5.5 Monitoring

To monitor the Digital Twin, I have connected it to as monitoring dashboard called Grafana. By connecting the Digital Twin to a monitoring visualization web app, I can watch all the sensor values live in a clean and modifiable dashboard which makes it a lot easier to watch the specific sensors I want to watch.



A screenshot of a Grafana dashboard showing a table view for a specific sensor. The table has two columns: 'Time' and 'counts'. The 'Time' column contains timestamps from 2022-06-15 10:19:15 to 2022-06-15 10:22:26. The 'counts' column shows values increasing from 5 to 12. The total count is 271551786.

Time	counts
2022-06-15 10:19:15	counts:5
2022-06-15 10:19:24	counts:6
2022-06-15 10:19:57	counts:7
2022-06-15 10:20:26	counts:8
2022-06-15 10:20:56	counts:9
2022-06-15 10:21:24	counts:10
2022-06-15 10:21:57	counts:11
2022-06-15 10:22:26	counts:12

Figure 5.12: View from Grafana with table view for a specific sensor

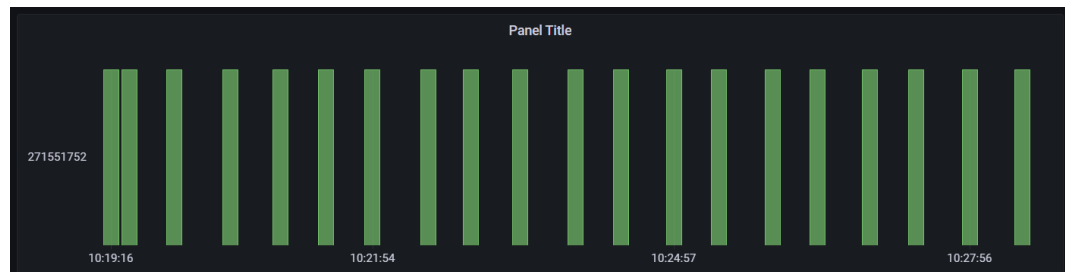


Figure 5.13: View from Grafana with bar view for a specific sensor

In Figure 5.12 I chose to watch a specific sensor and see its counts value in table view. In Figure 5.13 I chose to watch the same sensor but switched up the view form. Grafana makes it really easy to connect the database to the visualization tool and customize the view all I want. This is however connected straight to the database, so it will only work as a ground truth

5.5 Monitoring

for the sensor data, actions done in the Unity that affects sensor values will not be seen in this implementation. To get the live data straight from Unity, a new database needs to be created that live reads the data from Unity, and connect that one also to the Grafana system.

Chapter 6

Results

This chapter I will go over the current state of the project and what results I have found during this thesis and answer the research questions stated in Chapter 1. The proposed architecture will be justified by results and the findings done in Chapter 4 and Chapter 5, to best state if these proposed changes still are considered to be useful.

In the previous thesis that this project is based on [5], it was stated that the results section was based on the Digital Twins ability to produce said Digital Twins and collect data from multiple sources. However, I have looked more in to the architecture choices made in that thesis and how each section could be approved upon from how they originally was implemented. Not all implementations deemed successful, and some I can not really see the full impact of yet, since the Digital Twin project is still in an early stage and there is room for a lot more improvements, features and scalability. I will also in this section do a experimental run of the Digital Twin and how it is setup.

6.1 Experimental Setup

In order to run the Digital Twin application, there is a lot that needs to be set up before the user can run it. Some frameworks needs to be downloaded

6.1 Experimental Setup

and set up correctly, and some parameters needs to be set in different configuration files to be sure to have correct paths to programs on the computer. All of these steps will be walked through in the following section.

6.1.1 Requirements

The Digital Twin application is programmed and tested by people using Windows OS, and therefore it is currently only known to work on this OS since it is not tested on any of the other ones. Currently I'm using Windows 11 to run the application. There are two main programming languages used in the Digital Twin, Python and C#. To be able to run the Digital Twin a working version of both Python2 and Python3 are necessary. Python2 are being used to create the tunnel models, and Python3 are used for all the other components, OPC Emulator, GUI, Broker, etc. C# are being used for the code that are running in Unity to control the objects inside the tunnel. All of these programs should have its own version for different OS, but they are all OS specific, given that I have only used windows specific versions of the needed applications, I can not guarantee it will work on another OS. As seen in the table below, Table 6.1, all programs should be able to be run on Windows, Linux, MacOS except for the Unity 2019.4.5f1 version.

6.1 Experimental Setup

Table 6.1: Software and their platform dependencies

Software and their platform dependencies			
Software	Program	Version	Platform Accessibility
Digital Twin Framework	InfluxDB	1.8.4	Windows/Linux/MacOS
Digital Twin Framework	Unity	2019.4.5f1	Windows/MacOS
Model Generator	Maya	2020	Windows/Linux/MacOS
Broker	Apache Kafka	3.2	Windows/Linux/MacOS
Broker	Zookeeper	3.4.9	Windows/Linux/MacOS
Digital Twin Framework	Python	2	Windows/Linux/MacOS
Entire Application	Python	3	Windows/Linux/MacOS
Broker	JRE	8+	Windows/Linux/MacOS
Monitoring	Grafana	9.0	Windows/Linux/MacOS

6.1 Experimental Setup

I have only used a single computer through this project and therefore only tested the system on this one computer with specific resources. Since everything in the application is running locally, the resources the computer possess can have a big impact on how well and fast the application is running. In Table 6.2 below I have specified which components the computer I have used contains.

Table 6.2: Computer hardware used for testing

Computer hardware used for testing.	
CPU	Intel Core i7 (8. generation) 8565U
GPU	NVIDIA GeForce GTX 1650
Memory	16GB
Drive	512 GB SSD
Operating System	Windows 10 Home

6.1.2 Current State

By doing several tests on the Digital Twin with the new proposed architecture system, I can see on the performance based changes if there are any improvement to the benchmark metrics. By doing the same kinds of tests on the old implementation and replicating the same changes on the new implementation, I can see if there are any improvements to time usage. There are several tests that would be viable to check with the new implementation.

6.1.3 OPC Emulator

OPC Emulator time. Since the generation of sensor data has been changed to a more realistic approach, with more servers and more communication between different communicators, testing the new speed versus the old one is desired. The optimal way of measuring would be to time every step of the

6.1 Experimental Setup

way and compare it to the old implementation. This will give an overall idea of the difference between the implementation, and by logging every single step, it is possible to spot bottlenecks more easily in case of finding further improvements. To start a timer when a sensor starts emulating data, time it when it hits the OPC Server, time it when it hits the OPC Mainserver, time it when it hits the Digital Twin, and lastly time it when it is written into the database. This would give a good overall picture of the performance of which data is generated and stored in the database. Doing this with both the old implementation and new implementation and compare them would give a good benchmark of performance, given the experiment is tested several times to make sure there are as little randomness as possible.

However, the new implementation was not successfully developed into the Digital Twin. The current implementation is not currently working and it will not be possible to do an analysis of the performance since it is not functional. The current implementation sends sensor data over to the OPC Server, but is not able to send it forward to the OPC Mainserver, to then be sent to the Digital Twin. As of now, this is a non-functional part of the Digital Twin and was not a successful integration. The new implementation would have most likely performed at a slower rate than the old implementation, given that it has increased layers of communication and applications to go through. This is done on purpose, not to slow down the application, but for it to be more realistic in how it is done in the tunnels in the real world. Sensor data does not magically appear where you want them for convenient use, but needs to be generated and go through a realistic endeavor. One of the biggest points of a Digital Twin, is for it to replicate the real world as best as possible, so it is possible to make decisions based on the information generated using such concepts. I think the approach I have discussed and tried to implement is a better solution for realism, but at this point I can not tell how this would effect the Digital Twin when it comes to time performance. In Table 6.3 below is the benchmarks done by Nohut in his thesis [5] for the previous implementation. Nohut tested the time it took to pull the data from the OPC Server and until it was discovered as a change in the database by the Digital Framework. He got the following results with the old implementation Table 6.3.

6.1 Experimental Setup

Table 6.3: Testing done by Nohut on old implementation, Reaction Time [5]

Number of Entries	Fastest	Slowest	Mean	Median
500	3123ms	5266ms	4081.28ms	4035ms

6.1.4 Broker

Broker time performance. From the old to the new implementation when it comes to Broker communication, is the switch from MQTT to Kafka. This was done to improve performance, but also for the Digital Twin to have more scalability for the future. Right now, that is not much needed, since the MQTT worked fine for the implementation that was implemented. This change was a change made for the future of the Digital Twin. Right now the Digital Twin only handles three different types of sensors, light, sign, and cabinet. Given that this is a project that will be worked a lot on and developed even further, it is important to think about how it will scale in the future. If the Digital Twin is able to almost fully mirror a real world tunnel, it will be a lot more sensors than these three mentioned here. It might be hundred of different types of sensors alone, and with this in mind it is really important to think about solutions that can scale to fit expected future needs.

To test the benchmark performance to test the new implementation, the best way would be to time the time between new data is stored in the database until the Virtual Twin receives a notification for it to update itself with the new information from the database. However, the new implementation here also was not successfully implemented to the Digital Twin. The current implementation is not currently working and it will not be possible to do an analysis of the performance since it is not functional. The current implementation sends notification to the Kafka Broker, which then sends the notification to the Virtual Twin with a pub/sub system. However, the Virtual Twin is not able to recognize this data, and will in turn not update the objects in the Virtual Twin with the new data. The loaded objects in the Virtual Twin will only have their default values when created and never be updated. It is hard to tell how the performance would line up to each other if they both were able to be tested for a benchmark test. The hypothesis is that they would be very similar in performance at this point,

6.2 Research Questions

given the small number of sensors, but Kafka Broker would be better with future development. As of now, there is no way to say based on the performance if this change would increase the notification time. Nohut did make a performance test in his thesis for the old implementation [5]. He calculated the notification time by running the project for an hour, the Digital Twin would then produce timestamps right after the database updated, and when the Virtual Twin produced a new timestamp after it received notification to update [5]. In Table 6.4 below are the results from Nohut's testing performance for the old implementation [5].

Table 6.4: Testing done by Nohut on old implementation, Notification Time [5]

Number of Entries	Fastest	Slowest	Mean	Median
500	7ms	145ms	59.035ms	54.0ms

6.2 Research Questions

In chapter 1 I introduced some questions I wanted answered by doing this thesis and I will now see if I am closer to give an answer for them now. The questions was as followed:

- Are there improvements to be made to the architecture of the Digital Twin?
- What features are there to be added to the Digital Twin that could improve functionality?
- Are the InfluxDB a suitable database to be used for this project?
- It is important to get people to use this application, how can changes be done to the implementation to ease the barrier to entry?

"Are there improvements to be made to the architecture of the Digital Twin?"

I did thoroughly look into the components of the Digital Twin to find where improvements could be made. In the big picture, of which frameworks to

6.2 Research Questions

be used, the only thing I would want to change is the pub/sub system, change the MQTT protocol to Kafka. In Chapter 4 I have explained why I think this is a good change to do, but unfortunately since the Kafka implementation is not working, I can not test it properly and see how it would work. I did implement some smaller changes that does not affect the big picture of the architecture, like making more realistic sensors and how they are set up.

"What features are there to be added to the Digital Twin that could improve functionality?"

I have gone over several implementations that I think are good features to add to the Digital Twin, namely GUI, login and monitoring systems. These are systems that are present in almost every single Digital Twin I researched and should be in all of them. Also adding analytics is an important part of Digital Twins, which I have not tried to implement in this project. Either real time streaming analytics or batch loads.

"Are the InfluxDB a suitable database to be used for this project?"

As I thoroughly went through in Chapter 4, I think InfluxDB is a great choice of database for this project. InfluxDB is a database made for projects like this, time-series sensor data. It is free to use, which is important in this research project, and it is open source which is also important to let us know what is inside the program.

From the benchmark tests, it was miles better than the other databases that was tested, however there were not that many databases in the test, it does indicate at least that it is a lot better than some similar time-series databases, so its performance should at least be decent.

From the popularity overview InfluxDB was by far the most popular database from the metrics used in that research. This indicates that it is a good database since so many people are using it. And when many people use the same software, it is easier to find solutions for them and people that know how they work. Also by looking at InfluxDB in Digital Twins table in Chapter 2, Table 2.2, I can see that InfluxDB are also widely used in the creation of Digital Twins. Considering all this, I think InfluxDB is a great fit for this project.

6.3 Visualization

"It is important to get people to use this application, how can changes be done to the implementation to ease the barrier to entry?"

For people to try new things it is important that it is smooth running, visually pleasing and easy to use. I have been focusing in this project to implement something that makes the Digital Twin easier to use. And I think by adding the GUI to the implementation it will make users more appealing to try it out when they do not have to open a bunch of command windows and typing a lot of parameters just to get the program running. I think this is a good feature and will benefit this project with increased user approval.

6.3 Visualization

The Digital Twin is not only used for gathering data and processing them and putting them in to a database to be analyzed. The Digital Twin are also used to make the person that uses them feel like this is a replica of the real thing. The user should feel like this is close to the real world, and that everything that happens in the Digital Twin, can also happen in a real tunnel. The Digital Twin of the tunnels are made to look very similar to Norwegian Tunnels, and it is important that they are as realistic as possible, so the people that performs real life scenarios on them will have it feel realistic. For this reason, it is important for the Digital Twin to look good visually.

It is also important for the aesthetics to look good when it comes to sparking an interest for new people to try out the project. As one of the goals is to start generating a user base, either companies or other researchers, the barrier to entry is lower when having a visually more pleasing product to showcase.

The visualization of the Digital Twin have no way of being performance tested, and can only be tested by people viewing the application. I have not made any noticeable changes to how the Virtual Twin are being visualized, other than with GUI and Monitoring device. The Virtual Twin are however affected by Kafka Broker and OPC Emulator not working, so it will affect how the Virtual Twin is viewed to some degree. Below I will show some

6.3 Visualization

visualizations of the running application.

The first thing that meets the user when running the application is the GUI log in screen. Here I will create an account and log in with the created account.

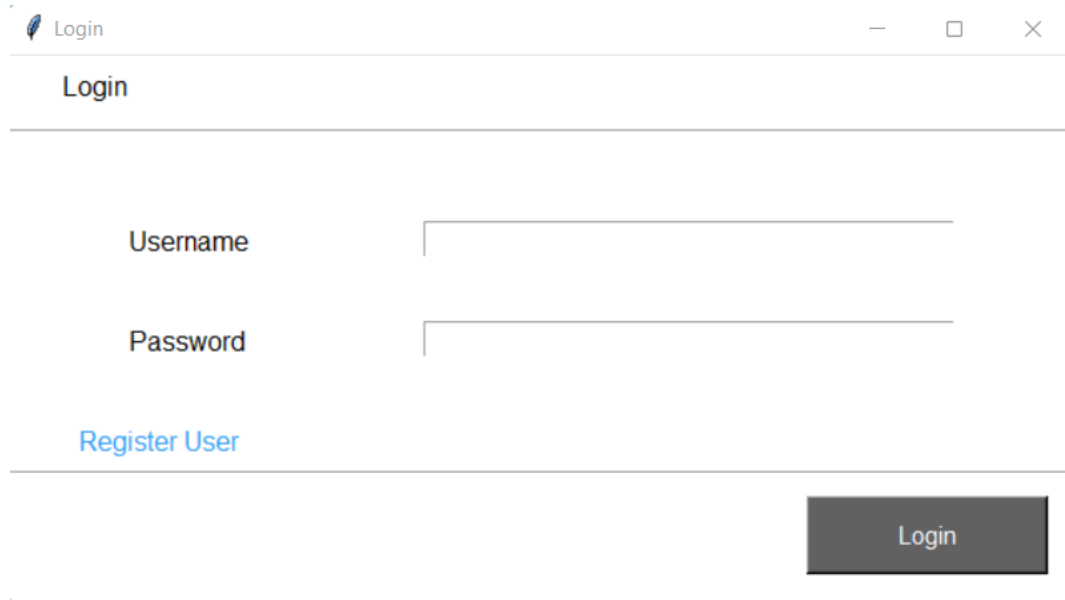


Figure 6.1: Screenshot of the GUI login screen

When the user has logged in, the user is taken to the main screen of the log in GUI. After filling out the wanted parameters, the user can now run the application. The correct order of running, is to first start the OPC Emulator, then the broker, and last the Digital Twin Framework.

6.3 Visualization

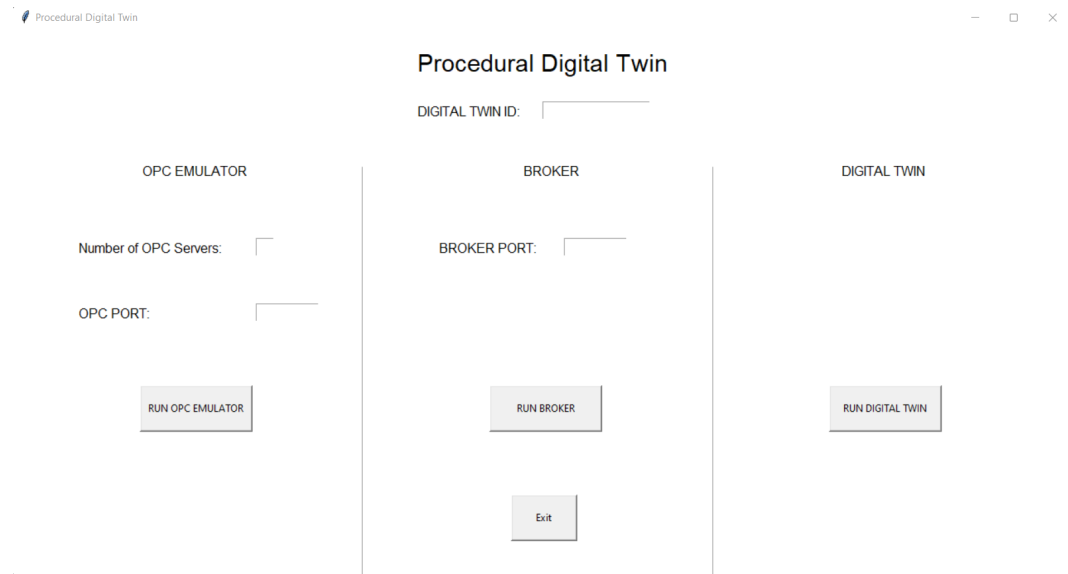


Figure 6.2: Screenshot of the main GUI screen

The Digital Twin will now start up, and a unity screen will open and the user is placed inside the desired Norwegian road tunnel. Figure 6.3, below visualizes how the tunnel looks from a distance when zooming out of the tunnel.

6.3 Visualization

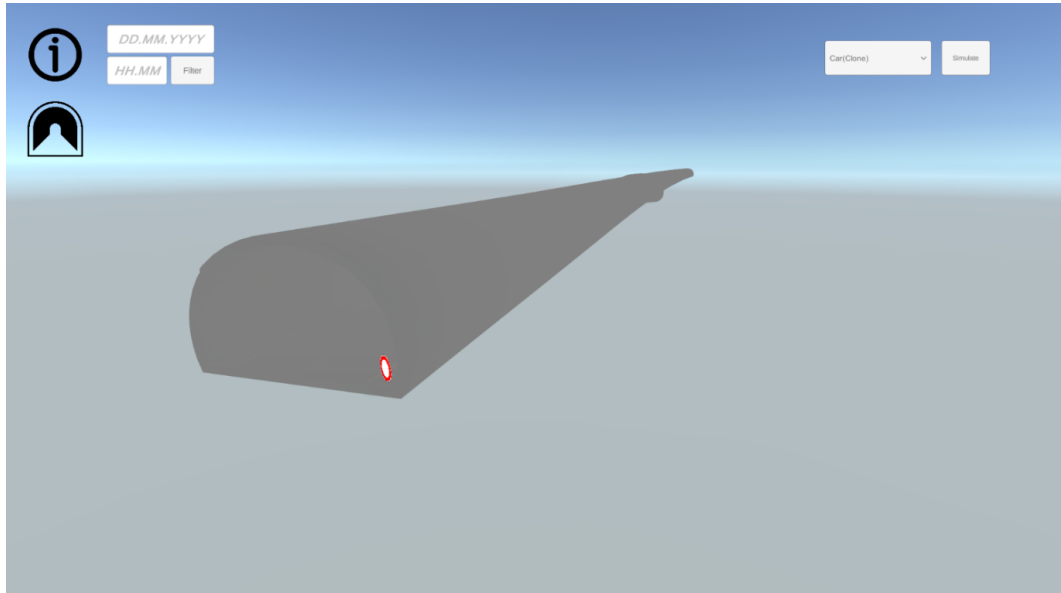


Figure 6.3: View of the zoomed out Virtual Twin in Unity

Immediately when starting up the Digital Twin and the Unity screen, the user is placed a little bit into the entrance of the tunnel. In this scenario I have used "Kleppetunnelen" in Rogaland as the tunnel I am replicating with my Digital Twin. Figure 6.4 below shows the immediate view of starting the Digital Twin in "Kleppetunnelen".

6.3 Visualization



Figure 6.4: Immediate view of "Kleppetunnelen" as of running the Digital Twin

In the current view I can not really see any of the objects that has been placed in the tunnel, only the procedurally generated tunnel model of this specific tunnel.

6.3 Visualization

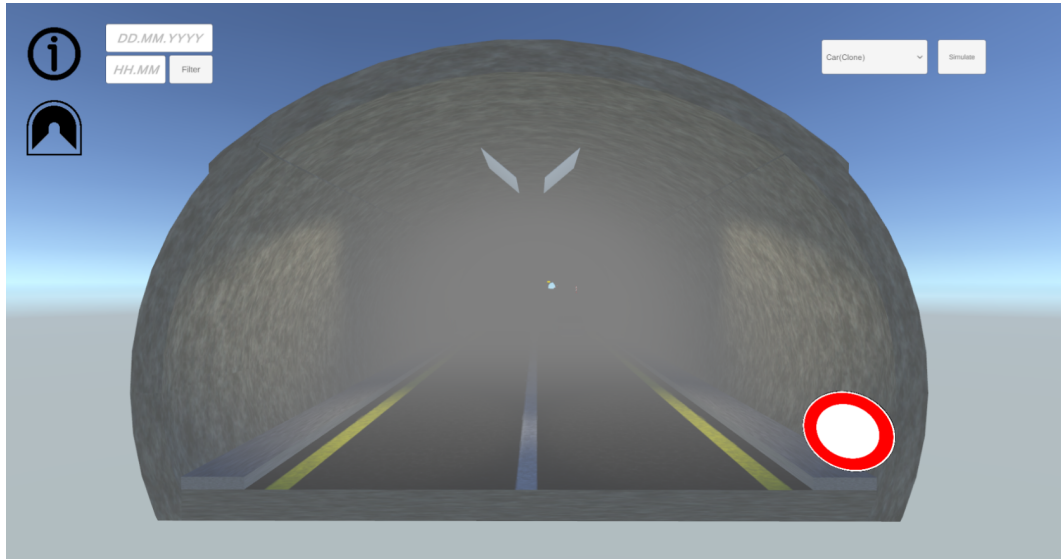


Figure 6.5: View of the entrance of "Kleppetunnelen"

By moving a little bit back, I can now see the entrance of the tunnel in the Figure 6.5. Here I can also see the first objects that has generated sensor data from OPC Emulator and set to the Digital Twin and be generated as objects. However, here I can see that since my Kafka Broker implementation is not working, the objects in the tunnel currently has no values, and the sign are therefore shown as a blank sign with no value inside it.

6.3 Visualization



Figure 6.6: View of the entrance of "Kleppetunnelen", taken from Nohut's thesis with the old implementation [5]

Figure 6.6 is taken from Nohut's thesis [5] and shows the same entrance as Figure 6.5, but in the previous implementation. I can see that his sign does have a value assigned to it, so it shows the sign for 60 km/h. I can also see that the lights have a different value, and therefore his tunnel are darker.

6.3 Visualization

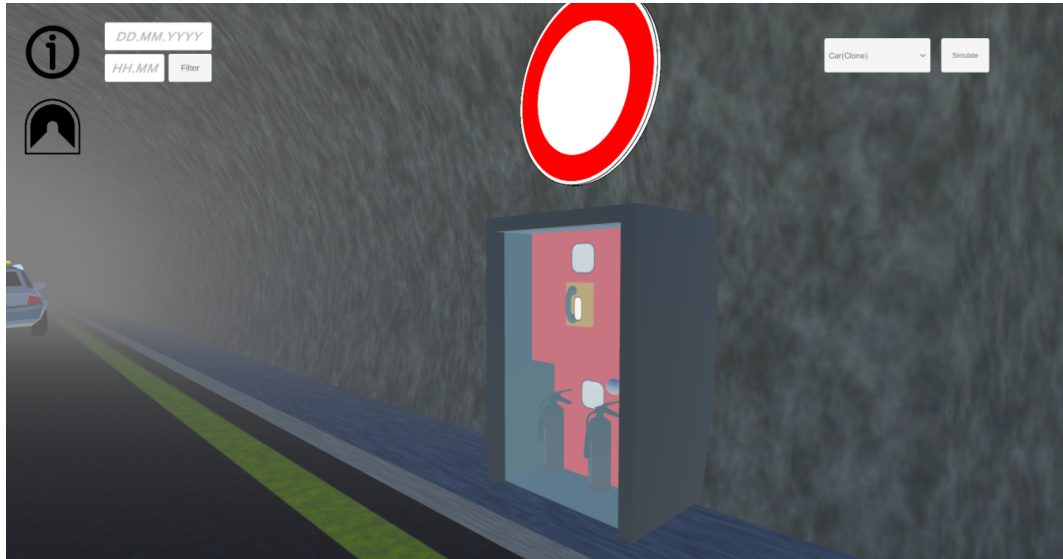


Figure 6.7: View of "Kleppetunnelen" midway in, with some objects

Figure 6.7 is taken further in to the tunnel and about midway. Here I can observe more of the objects that have been placed inside the Virtual Twin, here I can see a new sign and a emergency cabinet.

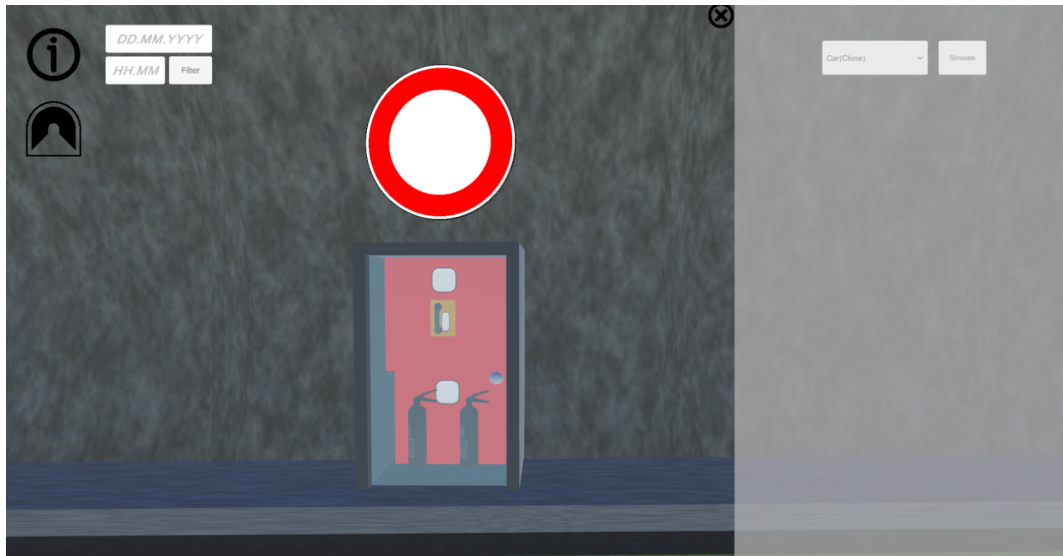


Figure 6.8: View of selecting objects inside the Virtual Twin

6.3 Visualization

Figure 6.8 shows what happens when I select one of the objects. A panel shows up on the right and is supposed to show the values of the objects. Since the current version has no working Broker, no results are shown when selecting the object.



Figure 6.9: View of the car fire simulation

Figure 6.9 shows what happens when I run one of the simulations that are made for the Virtual Twin. This simulation shows what happens when a car gets set on fire. All traffic from the cars behind will stop, and no more cars will drive through.

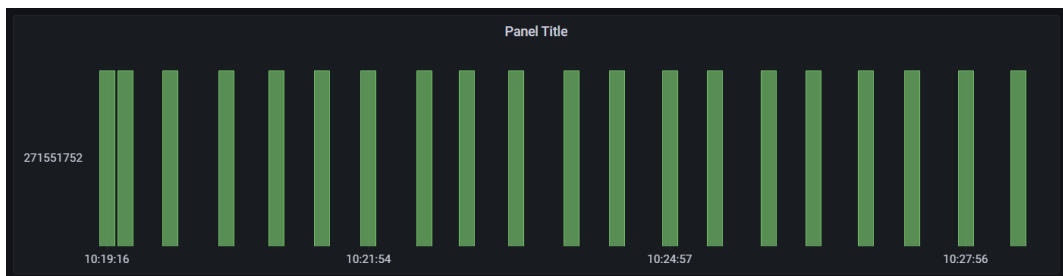


Figure 6.10: View from Grafana with bar view for a specific sensor

6.3 Visualization

Even though the Virtual Twin can not read the data from the database, the Grafana can. Using Grafana I can keep track of the sensor data I want at any time. And even if the Kafka Broker had worked, it would be a lot more tedious to go up to every sensor I wanted to know the value of, when I could just use Grafana to check its value at any point without having to move up to it. See Figure 6.10.

Chapter 7

Conclusion

There are two previous theses that have been written at University of Stavanger that builds upon each other to further develop the procedural generation of Digital Twins in Norwegian road tunnels. The first one aimed to generate tunnels in Unity based on data from NVDB, while the next one focused on generating sensor data that could be linked to objects inside these previously generated tunnels. This thesis aimed to further build upon the work that was put in to generate sensor data and link them to objects inside the tunnels.

This thesis studied the state of the art of Digital Twins to understand where improvements could be made, and new features could be added. With this thesis I studied several of the architectural components that was used in the previously made implementation and argued for weather they could be changed to a more fitting solution. I put a great deal of effort in finding ways to think about the architectural structure of the model and find improvements where it could be made.

I argued for the use of the current database, weather it was a good solution for this specific project, and I argued for weather the communication used between the Digital and Virtual Twin was the most optimal choice. In which I ended up with keeping the database and change the communication protocol used.

7.1 Future Work

I worked on five new implementations in this project, realistic sensor emulator, communication protocol, GUI, login system and monitoring system. Some of them failed and I was not able to implement them to test if these implementation would progress the work of the Digital Twin, and some of them worked. It is important to know that this Digital Twin is a very complex piece of software, and making big changes to its architecture are going to be very tedious and time consuming given that it is easy to disturb other parts of the application by doing so. Literature for Digital Twins were heavily studied to get an idea of how they generally work and how previously made Digital Twins in other industries could help thinking how we can further develop ours.

There are still a lot more work to be done in this project for it to be a state of the art Digital Twin, but I believe we are on our way there.

7.1 Future Work

7.1.1 Complete failed implementations

The first thing that should be done is to implement the changes suggested in this thesis that was not successfully implemented. Knowing how a Kafka broker will perform compared to the MQTT protocol and if the change are important to make. It is important to get the big architectural pieces in place before moving on with the application, since it will be increasingly hard to change later with an application that is just getting bigger and bigger. The same goes for implementing the failed OPC Emulator generator. Having sensor that is more realistic to the real world is important for Digital Twins to be as real as possible.

7.1.2 Platform

The application has only been tested using a Windows OS. The application is generically created and should in theory also be running on other OS but it has never been tested to do that. Future work should include testing if it runs well on other OS, and in case not, implement functionality for it to

7.1 Future Work

work on them. It is important to get users to use this application, and by giving more people opportunity to do that could increase the chances for it to be used.

7.1.3 Quality of textures

The Digital Twin is a research project and not a commercially available product. Therefore it is important to be cost effective. The textures used for the Digital Twin are all taken free from the internet and are not professionally made to look as good as possible or to represent all the available sign types that we have in Norway. Further development could be made to improve the visual quality of the application.

7.1.4 More sensors

The current application only allows for four different sensors to be generated and displayed in the Digital Twin. Further work needs to increase the amount of sensors for the tunnel to be more realistic, generate more information and create more analytics based on the many sensors to be generated. Right now the application can only generate sensor data and create objects for lights, signs, and two types of cabinets. It is not enough to make the Digital Twin a great application with thousands of data streams.

7.1.5 Cloud

A lot of the new Digital Twins that are being created are more and more cloud based. As seen in the research work in Chapter 2, there are many of them that are connected to the cloud in some ways. Whether it is the application itself running on the cloud, or the sensor generators, the cloud can save the user from needing to have an excellent computer that needs to run everything locally. Also by using the cloud, the applications are available everywhere, without needing to download all of the software and frameworks on each computer that needs to run it. Microsoft Azure and Amazon Sagemaker are two of the biggest rivals in today's market when it

7.1 Future Work

comes to creating cloud solutions that work for the general purpose.

7.1.6 Security

In the implementation of user account to be able to log in to the Digital Twin, I have used a very basic solution, and the database is also locally stored on the computer. This is not a very good solution at all, but it was used to create a concept for future development. A good future solution would be to create a log in system that is hosted on the servers at University of Stavanger, and connected to a database on these servers. The user would then just send API calls to these servers to log in and be able to use the application. Since the application already needs to be connected to the internet, this would not create more hassle for the user, and it would create a much more secure solution for the log in system. The database would also need to be changed to a better performance based database, since SQLite is not very scalable.

Bibliography

- [1] F. Tao, H. Zhang, A. Liu and A. Y. C. Nee, "Digital Twin in Industry: State-of-the-Art," in IEEE Transactions on Industrial Informatics, vol. 15, no. 4, pp. 2405-2415, April 2019, doi: 10.1109/TII.2018.2873186.
- [2] T. Ruohomäki, E. Airaksinen, P. Huuska, O. Kesäniemi, M. Martikka and J. Suomisto, "Smart City Platform Enabling Digital Twin," 2018 International Conference on Intelligent Systems (IS), 2018, pp. 155-161, doi: 10.1109/IS.2018.8710517.
- [3] R. Tomar, J. Piesk, H. Sprengel, E. Isleyen, S. Duzgun, J. Rostami, "Digital twin of tunnel construction for safety and efficiency", Tunnels and Underground Cities: Engineering and Innovation meet Archaeology, Architecture and Art, 2019, doi: 10.1201/9780429424441.
- [4] H. Elayan, M. Aloqaily and M. Guizani, "Digital Twin for Intelligent Context-Aware IoT Healthcare Systems," in IEEE Internet of Things Journal, vol. 8, no. 23, pp. 16749-16757, 1 Dec.1, 2021, doi: 10.1109/JIOT.2021.3051158.
- [5] Nohut, Berke Kagan, "Digital Tunnel Twin Using Procedurally Made 3D Models", 2021, University of Stavanger.
- [6] "WHAT IS DIGITAL TWIN TECHNOLOGY AND HOW DOES IT WORK?", <https://www.twi-global.com/technical-knowledge/faqs/what-is-digital-twin>. (Accessed March 13, 2022).
- [7] "What is a digital twin?" , <https://www.ibm.com/topics/what-is-a-digital-twin>. (Accessed March 13, 2022).
- [8] "InfluxDB", <https://en.wikipedia.org/wiki/InfluxDB>. (Accessed March 13, 2022).

BIBLIOGRAPHY

- [9] Skedler Team, "Everything You Need to Know about Grafana", <https://www.skedler.com/blog/everything-you-need-to-know-about-grafana/>. (Accessed March 13, 2022)
- [10] Steindl, G.; Stagl, M.; Kasper, L.; Kastner, W.; Hofmann, R. Generic Digital Twin Architecture for Industrial Energy Systems. *Appl. Sci.* 2020, 10, 8903. <https://doi.org/10.3390/app10248903>
- [11] K. M. Alam and A. El Saddik, "C2PS: A Digital Twin Architecture Reference Model for the Cloud-Based Cyber-Physical Systems," in *IEEE Access*, vol. 5, pp. 2050-2062, 2017, doi: 10.1109/ACCESS.2017.2657006.
- [12] Anro Redelinghuys, Anton Basson, Karel Kruger, "A Six-Layer Digital Twin Architecture for a Manufacturing Cell", 2019, *Service Orientation in Holonic and Multi-Agent Manufacturing. SOHOMA 2018. Studies in Computational Intelligence*, vol 803. Springer, doi: 10.1007/978-3-030-03003
- [13] V. Kamath, J. Morgan and M. I. Ali, "Industrial IoT and Digital Twins for a Smart Factory : An open source toolkit for application design and benchmarking," 2020 *Global Internet of Things Summit (GIoTS)*, 2020, pp. 1-6, doi: 10.1109/GIOTS49054.2020.9119497.
- [14] H. Zhou, C. Yang and Y. Sun, "A Collaborative Optimization Strategy for Energy Reduction in Ironmaking Digital Twin," in *IEEE Access*, vol. 8, pp. 177570-177579, 2020, doi: 10.1109/ACCESS.2020.3027544.
- [15] Vering, Christian and Borges, Sebastian and Coakley, Daniel and Krützfeldt, Hannah and Mehrfeld, Philipp and Mueller, Dirk. (2021), "Digital Twin Design with On-Line Calibration for HVAC Systems in Buildings".
- [16] Violeta Damjanovic-Behrendt and Wernher Behrendt (2019) An open source approach to the design and implementation of Digital Twins for Smart Manufacturing, *International Journal of Computer Integrated Manufacturing*, 32:4-5, 366-384, DOI: 10.1080/0951192X.2019.1599436
- [17] R. Rossini et al., "AI environment for predictive maintenance in a manufacturing scenario," 2021 26th *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* , 2021, pp. 1-8, doi: 10.1109/ETFA45728.2021.9613359.

BIBLIOGRAPHY

- [18] Cunha, B., Hernández, E., Rebelo, R., Sousa, C., Ferreira, F. (2021). An IIoT Solution for SME's. In: Gonçalves, J.A., Braz-César, M., Coelho, J.P. (eds) CONTROLO 2020. CONTROLO 2020. Lecture Notes in Electrical Engineering, vol 695. Springer, Cham. https://doi.org/10.1007/978-3-030-58653-9_30
- [19] A. Borghesi et al., "IoTwins: Design and Implementation of a Platform for the Management of Digital Twins in Industrial Scenarios," 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), 2021, pp. 625-633, doi: 10.1109/CCGrid51090.2021.00075.
- [20] Fei Tao, Meng Zhang, A.Y.C. Nee, Digital Twin Driven Smart Manufacturing, 2019, doi: 10.1016/B978-0-12-817630-6.00001-1
- [21] Jay Lee, Behrad Bagheri, Hung-An Kao, A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems, Manufacturing Letters, Volume 3, 2015, doi: 10.1016/j.mfglet.2014.12.001.
- [22] Michael W. Grieves, PLM—beyond lean manufacturing, 2003, Society of Manufacturing Engineers.
- [23] David Gelernter, Mirror Worlds, 1993, Oxford University Press, ISBN-10: 019507906X
- [24] Piascik, R., et al., Technology Area 12: Materials, Structures, Mechanical Systems, and Manufacturing Road Map. 2010, NASA Office of Chief Technologist.
- [25] "Time Series Database", <https://hazelcast.com/glossary/time-series-database/>. (Accessed June 10, 2022)
- [26] "DB-Engines ranking of Time-Series DBMS", <https://db-engines.com/en/ranking/time+series+dbms> (Accessed June 10, 2022)
- [27] "Method of calculating the scores of the DB-Engines Ranking", https://db-engines.com/en/ranking_definition (Accessed June 10, 2022)
- [28] Chris Churilo, "InfluxDB vs. Graphite for Time Series Data and Metrics Benchmark", 2019, <https://www.influxdata.com/blog/influxdb-outperforms-graphite-in-time-series-data-metrics-benchmark> (Accessed June 10, 2022)

BIBLIOGRAPHY

- [29] Chris Churilo, "InfluxDB vs. Splunk for Time Series Data, Metrics and Management", 2019, <https://www.influxdata.com/blog/influxdb-vs-splunk-for-time-series-data-metrics-management> (Accessed June 10, 2022)
- [30] Chris Churilo, "InfluxDB vs. Elasticsearch for Time Series Data and Metrics Benchmark", 2018, <https://www.influxdata.com/blog/influxdb-markedly-elasticsearch-in-time-series-data-metrics-benchmark> (Accessed June 10, 2022)
- [31] Chris Churilo, "InfluxDB Tops Cassandra in Time Series Data and Metrics Benchmark", 2018, <https://www.influxdata.com/blog/influxdb-vs-cassandra-time-series> (Accessed June 10, 2022)
- [32] Kai Waehner, "Apache Kafka and MQTT (Part 1 of 5) – Overview and Comparison", 2021, <https://www.kai-waehner.de/blog/2021/03/15/apache-kafka-mqtt-sparkplug-iot-blog-series-part-1-of-5-overview-comparison> (Accessed June 10, 2022)
- [33] "MQTT and Kafka", 2019, <https://www.emqx.com/en/blog/mqtt-and-kafka> (Accessed June 10, 2022)
- [34] Jay Clifford, "MQTT vs Kafka: An IoT Advocate's Perspective (Part 1 - The Basics)" 2022, <https://www.influxdata.com/blog/mqtt-vs-kafka-iot-advocates-perspective-part-1> (Accessed June 10, 2022)
- [35] Brian McClain, "Understanding the Differences Between RabbitMQ vs Kafka", 2020, <https://tanzu.vmware.com/developer/blog/understanding-the-differences-between-rabbitmq-vs-kafka> (Accessed June 10, 2022)