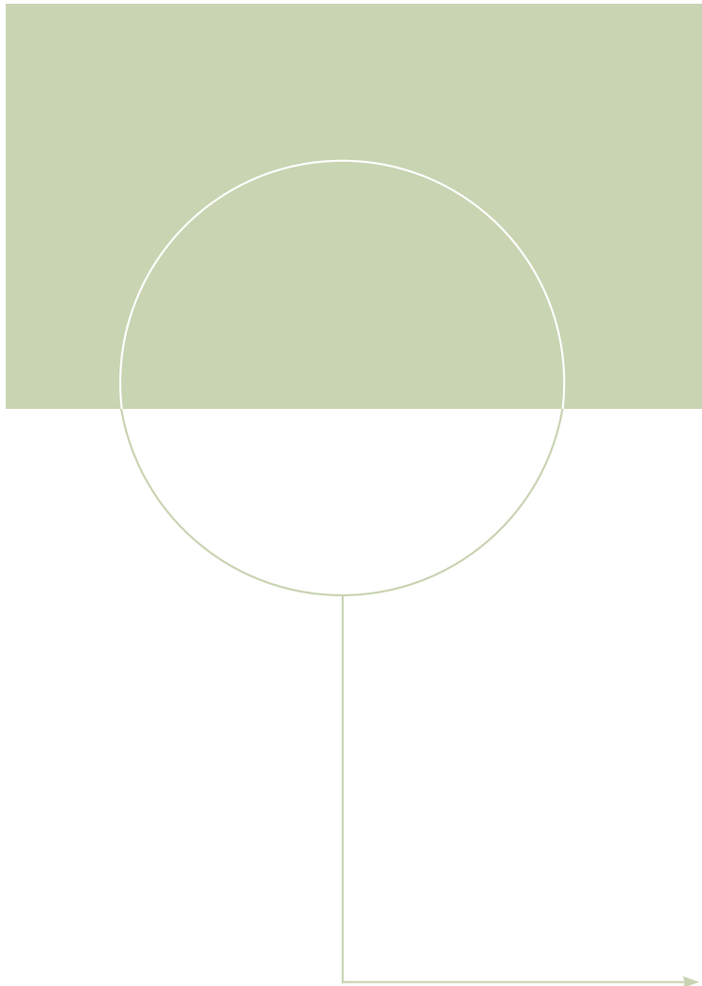

Master's thesis

2020

Espen Marstein Sandtveit

NTNU
Norwegian University of
Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Espen Marstein Sandtveit

Digital Twin Deployment at the Department of Mechanical and Industrial Engineering - NTNU

July 2020

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of
Science and Technology

Digital Twin Deployment at the Department of Mechanical and Industrial Engineering - NTNU

Espen Marstein Sandtveit

Mechanical Engineering and ICT
Submission date: July 2020
Supervisor: Bjørn Haugen
Co-supervisor: Terje Rølvåg

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

Preface

This Master thesis is written on behalf of the Department of Mechanical and Industrial Engineering (MTP) as part of the study program mechanical engineering and ICT. The project was completed during the spring semester of 2020 as a continuation of the specialization project from the prior semester. This thesis is aimed at deploying a digital twin instance and in the process evaluate and continue development of the current digital twin solutions at MTP.

The project is supervised by Bjørn Haugen and the co-supervisor is Terje Rølvåg. The supervisors have provided assistance and guidance throughout the project. FEDEM technologies and SAP have assisted by providing resources on the physical asset used in this thesis. A special thanks to Runar Heggelien from SAP for taking the time to answer questions during the project.

Abstract

Digital twins is a fast-growing field and publications on the topic are increasing. Businesses are dedicating an increasing amount of resources to the implementation of digital twins. Especially fields such as structural integrity monitoring and predictive maintenance stand to benefit greatly from the digital twin technology. Several companies like Siemens, Ansys, and IBM are working on digital twins, but these solutions are proprietary and expensive. Despite this commercial interest, open-source digital twin solutions are few and not well suited to the applications at the department of mechanical engineering (MTP).

There is an ongoing project at MTP to develop a digital twin platform for academic work. This thesis builds on the work done in previous years and focuses on the backend of the digital twin platform. The work done in previous years has been analyzed and areas of improvement have been identified. The previous work is compared to the definition of a digital twin that is defined in this thesis. Features to add or improve are derived form this comparison. The solutions are then developed further before a digital twin instance is selected and deployed to the platforms.

The thesis reviews the deployment of a digital twin at MTP. Two platforms are reviewed, one utilizing the Azure digital twin API and one in-house solution. The thesis explains why development on the platform using Azure is not continued. Then Bidirectional communication is added to the in-house solution and the message format is changed before a new digital twin instance is deployed to the platform. The thesis concludes with a section on future work on the platform.

Sammendrag

Digitale tvillinger er et raskt voksende felt, og det er flere og flere publikasjoner om emnet. Bedrifter bruker en økende mengde ressurser på implementeringen av digitale tvillinger. Spesielt områder innen overvåking av strukturell integritet og forutsigbart vedlikehold kan dra stor nytte av digital tvilling teknologi. Selskaper som Simens, Ansys og IBM jobber med digitale tvillinger, men disse løsningene er proprietære og kostbare. Til tross for denne kommersielle interessen er open source digitale tvilling løsninger få og ikke godt egnet for applikasjonene ved avdeling for maskinteknikk og produksjon (MTP).

Det pågår et prosjekt på MTP for å utvikle en digital tvilling plattform for akademisk arbeid. Denne avhandlingen bygger på arbeidet som ble gjort i tidligere år og fokuserer på backend av den digitale tvilling plattformen. Arbeidet gjort i tidligere år er analysert og forbedringsområder er identifisert. Arbeidet sammenlignes med definisjonen av en digital tvilling som er definert i denne oppgaven. Fra denne sammenlikningen blir funksjoner som kan legges til eller forbedres identifisert. Løsningene blir deretter videreutviklet før de evalueres på nytt. En digital tvilling instans blir valgt og deretter testet på plattformene.

Oppgaven omhandler evalueringen av digital tvilling plattformene på MTP samt utviklingen av en ny digital tvilling instans. To plattformer blir gjennomgått. Av disse bruker den ene Azure IoT Hub API og den andre er en eget stående løsning uten eksterne aktører. Oppgaven begrunner hvorfor utviklingen på plattformen som bruker Azure IoT Hub API ikke videreføres. Deretter legges det til toveiskommunikasjon til den interne løsningen og meldingsformatet endres før en ny digital tvilling instans blir testes på plattformen. Oppgaven avsluttes med et avsnitt om fremtidig arbeid på plattformen.

Abbreviations

List of Abbreviations

API	Application Programming Interface
DAS	Data Acquisition System
DT	Digital Twin
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
JSON	JavaScript Object Notation
MTP	Department of Mechanical and Industrial Engineering
PLM	Product Lifecycle Management
UDP	User Datagram Protocol
TCP	Transmition Control Protocol
IoT	Internet of Things
CAE	Computer Aided Engineering
JSON	Java Script Object Notation

Glossery

FEDEM: A computer program for multibody simulation of mechanical systems. In this thesis FEDEM is used to mesh finite element models and export the executable files(FMUs) for simulations.

Tvilling Digital: The API created by Simen Norderud Jensen in 2019.

Bluerig: Or "Testrig" as it is also called is a jack placed on a trolley at MTP faculty. The asset is depicted in appendix D.

Tingen: Or "The Thing" as it is also called is an inverse pendulum used as the physical asset in this thesis.

HBM data acquisition board: The specific data acquisition board used on the "Bluerig".

Catman software: Used for data acquisition on the "Bluerig". This sotfware has some data visualization and allows data to be sent to an IP address.

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

The Department of Mechanical and industrial engineering (MTP) has a goal to develop a digital twin(DT) platform that will serve as a tool for research in the field of real-time structural integrity monitoring and predictive maintenance.

In previous years there have been work done on the development of a DT platform at MTP. This thesis has used the work done in last years specialization project and the thesis written by Simen Norderud Jensen on the "Tvilling digital" platform[1] to evaluate and develop a complete DT platform. A DT instance was chosen and deployed to the platform to evaluate the work done. In an effort to standardize the DT setup, a data acquisition board was chosen and an API for communication with the platform was developed.

1.1 Background and Motivation

In recent years, the introduction of affordable 4G and soon 5G has accelerated the development in the field of Internet of Things (IoT)[2]. There is an increasing amount of publications in the field of IoT and DTs. Companies are now looking to add value to their IoT data by implementing DTs. This can enable new ways of looking at their data and handling their physical assets in the field. A DT can bring new information and knowledge about an asset and thereby increasing the value of the data. Enabling DTs opens up for several benefits for industry by giving more information to the decision basis and allowing remote control of their assets in the field.

There are several types of DTs, but this thesis will only discuss DTs that have a physical counterpart. The main area of interest in this thesis is real-time structural integrity monitoring. The data created by real-time structural integrity monitoring

can be used for purposes like estimation of Remaining Useful Lifetime(RUL). It can also be used to evaluate the performance of the assets in the field against the uses the asset was intended for. The insight given by DTs can reduce the need for on-site presence of personnel in dangerous or remote locations. It can also be used to optimize downtime due to planned maintenance.

Today there are several companies working on DT solutions that answer industry needs, but these are costly and not well suited for the academic work at MTP. DT is a term that refers to a software representation of a physical asset. In this thesis, a DT will refer to a virtual representation of physical assets that can be represented with a finite element model (FE model). The FE model is provided by SAP and the FEDEM software is responsible for the calculations performed based on the sensor data. This project also tries to generalize the setup of these DTs and make this into a process that can be performed in a cost effective and quick manner. In order to achieve this, two platforms that use different technologies was investigated.

During the specialization project of fall 2019, a proof of concept was developed. This platform utilized the API and infrastructure provided by Azure in order to simplify development and increase the versatility of the DT platform compared to the existing solution. From previous years another solution has been developed at MTP. This solution was built in-house and was running on a virtual machine at NTNU. This thesis evaluates the platforms and continue developing their features before a new DT instance is deployed.

1.2 Problem Description

The main goal of this thesis is to deploy a DT instance and to develop the current platforms in accordance with the initial evaluation. To achieve this the thesis have been divided into sub-goals that are described below.

1. Research the existing solutions and evaluate their initial state against the definition of a DT.

2. Identify a physical asset to be implemented as a DT instance.

3. Develop the DT platforms in accordance with the initial evaluation.

4. Deploy the DT instance and evaluate.

1.3 Outline

This section will explain the overall structure of the thesis, and the purpose of each section.

- **Introduction:** Introduces the topic of this thesis and motivates for its relevance. Gives some background information and formulates goals for the project.
- **Theory:** This section will explain relevant concepts and literature that makes the foundation of the project.
- **Technology Research:** This section explains the technology choices based on the concepts from the section on theory. It also covers some in depth features of technologies used.
- **Implementation:** This section contains the development process. It describes the technology that has been used and how they have been applied. It describes the deployment of the physical asset and the considerations that were made in the process.
- **Results:** Presents the results generated in the thesis. Performance data for the platform and the DT deployment is visualized.
- **Discussion:** Discusses results and how the current implementation answers the problem statement in Section 1.2. The platform is also compared to the theory of a DT platform from Chapter 2.
- **Conclusion:** Gives a critical view of the project. Goals that where achieved and not achieved and how this thesis contributes to MTPs goal of developing a DT platform.

Chapter 2

Theory

2.1 Definition of a Digital Twin

The definition of a DT varies according to which field and context its being used in. The first attempts of a DT is perhaps the "pairing" technology developed by NASA for the space program during the 1970's[3]. This paring technology consisted in gathering information from astronauts and instrumentation and then manually update the physical replica on earth. By doing this the ground crew was able to assist the astronauts both in testing and in identifying errors in a way that would be impossible without the pairing technology. In particular the Apollo 13 benefited greatly from pairing technology and it is said to have been crucial to Apollo's reentry to earth[3]. Despite this it is widely acknowledged that the concept of a DT as the term is used today was originally described by Dr. Michael Grieves at the university of Michigan in 2002. In a lecture held at the formation of a PLM center, Dr. Grieves describes the components of a DT system and how they interact. At the time the concept wasn't called DT, but the lecture clearly illustrates the concept of a DT as they are known today. Figure 2.1 shows Dr. Grieves scheme from the 2002 lecture[3][4][5][6].

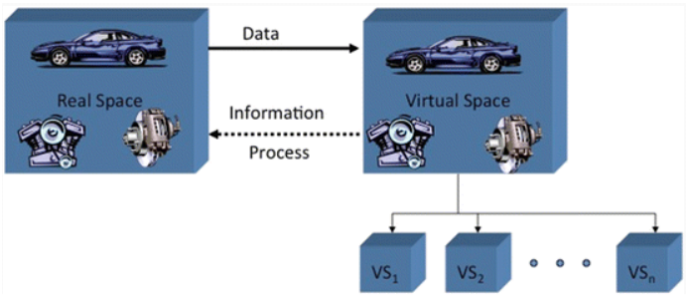

Figure 2.1: Dr. Grieves original schema of the DT concept.

It wasn't until 2010 that the concept conceived by Dr. Grieves was called "digital twin" by John Vickers at NASA. The DT concept consisted of three parts. The physical asset, the virtual asset and the communication between the two. Later in 2017 Dr. Grieves and Vickers wrote an article together where they defined the term and specified different types of DT's[7]. The following paragraph is the definition of a DT that Dr. Grieves and Vickers proposed.

"Digital Twin (DT)—the Digital Twin is a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level. At its optimum, any information that could be obtained from inspecting a physical manufactured product can be obtained from its Digital Twin."

This definition clearly states the goal for an optimum DT. What the definition does not directly include is a clear statement for the connection between the real and the virtual representation.

In 2018 a research group from Vienna in Austria gathered resources on the naming conventions for a DT[8]. Because the term DT is used slightly differently in the disciplines that discussed them, they proposed to classify the DTs according to their level of integration. They formulated three categories:

- **Digital model**: Manual data transfer from physical object to digital object and from digital object to physical object.

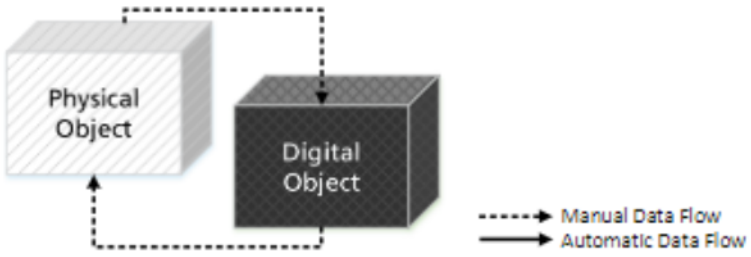

Figure 2.2: Information flow in a digital model

- **Digital shadow**: Direct data transfer from physical object to digital object and manual data transfer from the digital object to the physical object.

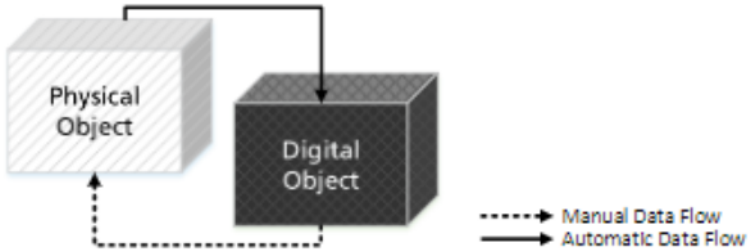

Figure 2.3: Information flow in a digital shadow

- **Digital twin**: Direct data transfer both from the physical object to the digital object and from the digital object to the physical object.

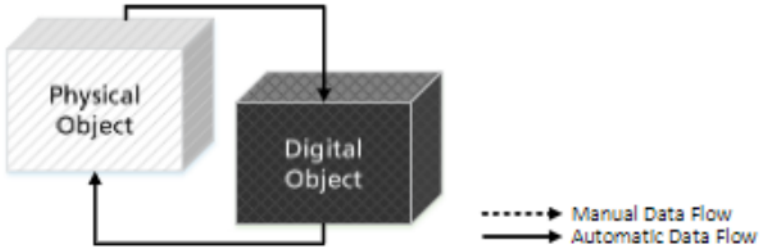

Figure 2.4: Information flow in a DT

The paper on naming conventions by Kritzinger[8] states that for a physical object to have a DT, the information flow has to be automatized both from the physical asset to virtual and from the virtual asset to the physical[8].

The two reports both propose classifications and goals for DTs. While Kritzinger from the research group in Vienna proposes a classification system, Dr. Grieves and John Vickers proposes to grade DTs from optimum DTs to less optimum DTs. Depending on how much of the information from the physical asset can be obtained from the DT. These two definitions are both widely used. By fulfilling one definition one does not necessarily fulfill the other. Therefore, this paper evaluates a DT with respect to both definitions. A "full DT" refers to a DT instance where the physical asset is fully described by its digital counterpart, and the communication between them is automated.

2.2 Applications of Digital Twin Technology

DTs have many applications, and as the field matures new uses will be found. Today DTs have industrial applications in several areas. Some examples are:

- **Maintenance**:
  DTs are being used to optimize maintenance of equipment such as power generation and jet engines[9].
- **Enterprise Architecture**:
  A field that have gotten some attention lately. By making entire blue prints of an organization and having metrics on performance means that insight to how businesses operates at any time can be obtained[10].
- **Asset Monitoring**:
  Used to create digital replicas of physical objects. This means that key personnel like operators know the status or configuration of the physical object without having to manually investigate the asset[3].

2.3 Cloud Computing

Amazon states: "Cloud computing is the on-demand delivery of computing power, database, storage, applications, and other IT-resources"[11]. A cloud can be everything from your own laptop being remotely accessible to the large services provided by companies like Google, Amazon and Microsoft. Cloud providers provide services that is accessible through the internet. These services offers access to computing power storage and more. Cloud services provide many advantages instead of buying and maintaining hardware for every use. By centralizing the management of hardware and computational power, resources can be managed more efficiently. The initial cost of hardware is also eliminated for the user. Big cloud providers also ensure that safety is up to date. Clouds also provide the possibility for dynamic scaling depending on the service. The different services provided typically fall into one of four categories, IaaS(Infrastructure as a Service), PaaS(Platform as a Service), serverless or SaaS(Software as a Service)[12]. These four offer different levels of control or independence to the user by letting almost all configurations be optional for the user, or providing the user with development environments and even ready to use software.

IaaS

The model with the highest level of control given to the user is the IaaS. This provides the user with a high-level API and the user must define things like man-

agement of resources, location, operation system and backup of data. This service is typically renting servers, virtual machines or storage[12].

PaaS

This model offers a development environment to application developers. Normally this include a predefined operating system, database, web server and programming-language execution. Developers buy the software platform instead of leasing the underlying hardware[12].

Serverless Computing

Serverless computing overlaps with the PaaS model but this is a cloud-computing execution model where the cloud provider runs the server and dynamically manages the allocation of computational resources. The applications are event driven. This means it only uses resources when a function is triggered. Pricing is based on the amount of resources consumed by a user, rather than on pre-purchased fixed capacity[13].

SaaS

SaaS is a method for delivering applications over the internet. Examples of SaaS are software like Outlook or Gmail. With SaaS the cloud provider handles underlying infrastructure, maintenance and updates[12].

2.4 Data Acquisition System

A Data Acquisition System (DAS) are systems designed to measure and track physical systems. Their purpose is to convert measurements of the physical system into data the computer can read, store and manipulate. A DAS consists of three parts: sensors, an analog to digital converter(AD converter) and a connection between the AD converter and the sensors. Figure 2.5 shows how sensors gather data from the physical system and then transmits this to the AD converter. Sensor readings are subject to interference form the environment and some noise will be present in the reading. Using filters besides the AD conversion are in many cases beneficial.

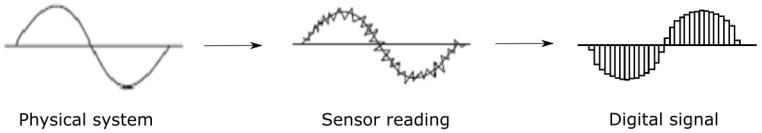

Figure 2.5: Illustrates how data is collected and processed in a DAS.

To make an effective DAS the sensors are placed in key locations on the physical system. The sensors are then connected to the Data Acquisition Board(DAB). This connection is normally some low power transmission over short distances, either through Bluetooth or wiring. The DAB is often responsible for several sensors and continuously receives sensor data. The DAB transmits the data to either a computer at the location or directly to the cloud.

DAB is a micro processor or computer. The aim of the DAB is to reliably perform it's task while being as cost effective as possible. In many cases it is hard to know what is required of the DAB going into a project. Therefore, some DABs are specially designed with development in mind.

At NTNU Arduino is the most popular of this development boards, while Raspberry Pi with the possibility of using different programming languages is a good choice for more software intensive development. Boards like Arduino and Raspberry Pi became very popular because they are relatively cheap and because they offer both data acquisition and micro-controller capabilities. They also have fairly high level programming languages which makes them easier to start developing with[14].

2.5 Messaging System

The following sections discusses the theoretical foundation for a high-level messaging system. Message systems are used to manage large amounts of events or messages. A message system is responsible for transferring data from one application or process to another so they don't get bogged down or cluttered by the data transfer. Messages are placed in queues, thereby decoupling processes and applications. There are two types of messaging patterns. The first is "point-to-point" and the other is "publish-subscribe" pattern. The "publish-subscribe" or "pub-sub" is the most used[15]. This pattern allows multiple data sources to publish data to the queue and multiple sinks to subscribe to the queue.

2.5.1 Messaging Architecture

In the field of IoT it is normal to divide the system into two parts. These are often referred to as the edge solution and the cloud solution[16][17]. Edge makes up the "on-site" hardware and software while cloud is the software that resides in the cloud. The hardware utilized by the cloud are responsible for storage, processing and allowing access to users. This hardware can be utilized in a number of ways through different services models, see Section 2.3.

Figure 2.6 shows a generic architecture for a streaming platform that collects data from different devices. In this case a device can be a single sensor, or several sensors and actuators connected to a DAB. In any of these cases the data is then sent to the gateway, as shown in figure 2.6. This gateway is any computer or DAB that is connected to Wi-Fi and is responsible for moving the data from the edge solution and to the cloud[17].

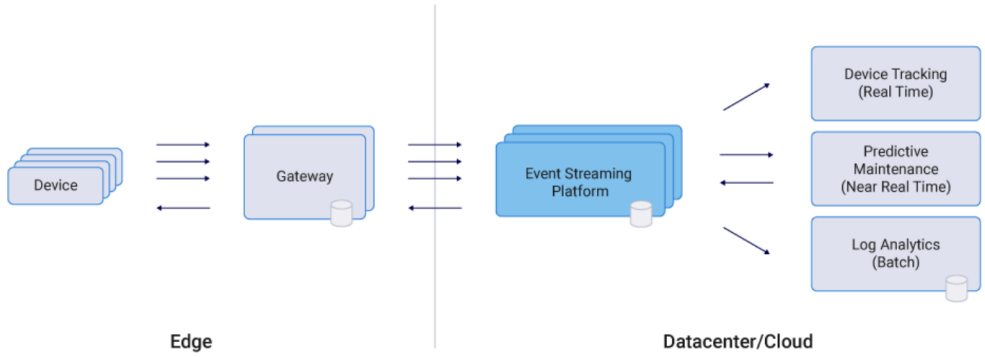

Figure 2.6: Illustrates how information flow though a generic IoT architecture. The vertical line indicates the separation between the on-site system and the cloud. The figure is taken from Confluent blog post [17]

2.5.2 Message Brokers

A message brokers is “A program that translates a message to a formal messaging protocol of the sender, to the formal messaging protocol of the receiver”. This means that message brokers act as middlemen between applications and processes[18]. Message brokers are useful when a system is processing large amounts of messages or data. The implementation of a message broker will simplify the handling of data and also helps to decouple processes allowing for temporary storage which allows processes to consume data asynchronously[19].

In a DT system it can be expected that there will be both large amounts of messages and differences in processing time. A DT instances may produce as much

as 200 messages per second. Data that have to be inferred from the DT model have a longer processing time than raw data. The implementation of a message broker allows these processes to work independently and enables asynchronous communication in the system. In this way messages are ready for the simulation process to consume as it finishes its current calculations.

2.6 Networking and Data Transfer

A key aspect of DTs is remote monitoring. This is achieved by sending sensor data from the physical asset to the cloud platform. The internet is ideal for this task. Sending data over the internet requires an IP address to ensure that packages reach their destination. Each computer connected to the internet is assigned an IP address. Personal computers that are connected to a router have a local IP address[20][21]. These are only used in that sub-net. Routers on the other hand have a public IP address. A public IP address is recognized across the internet[20][21]. Computers that communicate behind the router use the local IP addresses. These are assigned to every device that is connected to the router but can only be used inside that network. Communication over different networks uses the public IP address that is globally unique to ensure it reaches its destination[20][21].

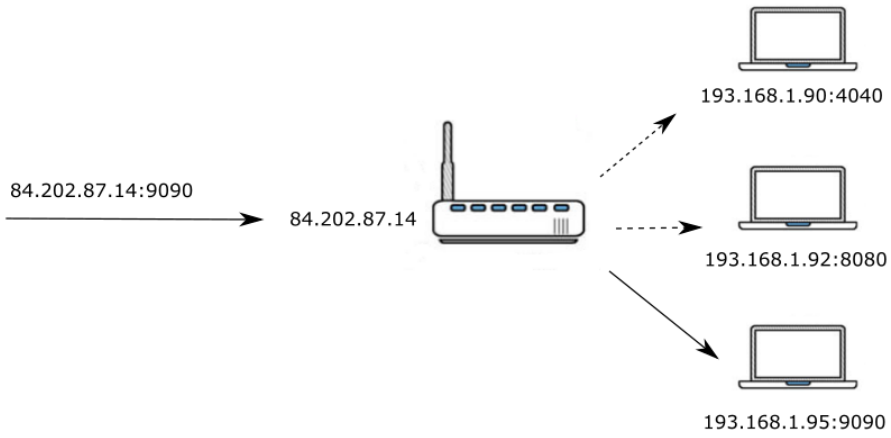

Figure 2.7: Illustrates how a request from outside the network is mapped to a computer inside the network.

Figure 2.7 illustrates an example network. In this network machines behind the router are represented as laptops. These three computers can address each other by referring to the local IP address and a specific port on the receiving computer[20][21]. The arrows illustrates how a message from outside the local network reaches its destination. The message is addressed to the public IP address of the router. The router is then responsible for directing the message to the correct local IP address inside its own network.

2.7 FMI and FMU

The Functional Mock-up Interface (FMI) is a standard that defines a container and an interface to exchange dynamic models. FMI standard uses a combination of XML files, binaries and C code that are zipped into a single file. The FMI standard is supported by over 100 tools and is maintained as a Modelica Assosiation Project [22].

2.7.1 Model Exchange

An instance of the FMI is called an FMU or functional Mock-up unit. These are containers or models that follow the standards of the FMI. By following this standard the models can be exchanged between any of the FMI compatible programs no matter the original format [23].

2.7.2 Co-Simulation

Some FMUs also support co-simulation. This means that the FMU contains a solver for the specific model and that the model can be solved in any compatible program as long as the platform supports the binaries[23].

2.7.3 FEDEM FMU Export

The FMUs that are exported by FEDEM utilizes the model exchange in order to run the models independently from the FEDEM software. Co-simulation is used to export the solver from FEDEM. The FMU builder from FEDEM currently exports binaries that are supported in a 64-bit windows platform. In addition, the FMU builder adds a link to a database. This checks a license and insures that FMUs that are built by FEDEM only can run in networks that are licensed.

2.8 System Delay

System delay in this thesis refers to the time from the measurement is taken to the moment the data is received in the browser. All measurements are made on a single computer; thereby ensuring that the clocks are synchronized and eliminating the delay caused by internet transfer. The transfer time over the internet is dependant on the geographical locations and routing. Because of the arbitrary locations of any DT instance this delay will vary according to the geographical

location and the traffic on the routers used. The development of this platform will not effect the internet transfer time and it was therefor decided to leave this out of the delay calculations. An additional delay will occur from the distance between the user and the physical asset. The routing between the to will also play a role and may vary depending on traffic to the specific routers.

2.9 Real-Time

The goal of this thesis is to perform real-time finite element calculations on DT's. This section discusses the term real-time and finite element simulations in the context of a DT platform.

"A real-time computer system may be defined as one which controls an environment by receiving data, processing them and returning the results sufficiently quickly to affect the functioning of the environment at that time" [24].

The quote above is in this thesis interpreted to mean that the definition of real-time for a DT instance is dependant on the physical asset. In processes that are slow, updates every minute gives the process a high resolution, like the waterline changing with the tide. Other processes are faster, like the oscillation of a spring. In this case updates are required to be much more frequent than every minute. The system needs to be able to measure the spring at any point in its oscillation and respond to that position. For a generic DT instance the sample rate should be small enough so that information lost between measurements is considered to be trivial and not effect the overall picture of the asset's operational conditions. The DT platform should then also be able to respond fast enough to affect the system at these times.

2.9.1 Real-Time Finite Element Simulations

In the field of mechanical engineering the Finite Element Method(FEM) has imposed it self as the most powerful and versatile tool when it comes to structural analysis[25]. It enables highly effective modeling and simulation of structures characterized by complex geometries that have high numbers of boundary and initial conditions. Typically these simulations are preformed off-line[25], and contain three essential parts, the first being preprocessing. This includes building the model and placing boundary and initial conditions. The second is the solver, which provides the solution to the problem with all required quantities. Finally there is the post-processor that offers tools for visualization to analyze the obtained solution [25]. In the case of a DT platform it is the last two phases that are of interest. The design of the model and the initial conditions are determined by engineers in

advance.

FEM simulations in the sense it has been discussed so far refers to a structure divided into a finite number of elements. This means that for any general geometry in 3 dimensions the model will consists of thousands of elements. This makes the simulation process very computationally demanding and time consuming. Preforming these calculations in real-time is not feasible for a general geometry. Therefor the FMU that is exported from FEDEM does not contain the full model with all nodes. Instead the user have to select nodes for input and output. Only the necessary equations for the selected nodes is kept, and the rest of the equation system is removed. This reduces the number of equations needed to be solved for every input. In theory the simulation of a DT should describe the model down to the micro atomic level, see Section 2.1. Practically this is difficult, and therefore it is left to the engineers to choose key values, or a resolution that describe the system sufficiently.

2.10 Async & Multiprocessing

In order to make a web API for a DT platform, both async and multiprocessing are techniques that are central to the implementation.

2.10.1 Async

Async is a technique that is used in web development to make the web page more responsive. Async declares that the function will execute asynchronously via the event loop, and it uses an implicit promise to return the result. This means that while the execution is waiting for the result from a request to return, other tasks can be executed. A typical scenario in a server can be a request for data in a database. The server will then send a request to the database and wait for the response, before returning it to the user. Async allows the server to handle other tasks while waiting for the database to respond to the initial request. This is not the same as doing two things at the same time. Async only works when the execution is waiting, meaning not doing anything. This is why it is so well suited for web APIs. There are often many database calls leaving the server waiting for a response[26][27].

2.10.2 Multiprocessing

By using multiprocessing, processes can be executed in parallel. When a new DT is added to the platform, resources need to be programmatically made available

to the DT instanse so that the data processing for that instance can be preformed in real-time. In the instance of a DT platform, multiprocessing is used to allocate resources to processes. This allows the platform to do FEM calculations while still being able to handle requests from the web page, or to send data directly to the user while the calculations are being executed in the background[28].

Chapter 3

Technology Research

3.1 Data Acquisition Board

From Section 2.4 it follows that the Data Acquisition Board (DAB) needed was a board made for development. Factors like high flexibility and customization for academic projects is paramount for the selection of the DAB. The main users for this system will be students at NTNU. Reducing cost and complexity compared to the HBM DAB from previous years will be critical for student involvement in the future. The DAB should be familiar to the user in terms of setup and programming language. The two biggest vendors are either the Arduino or the Raspberry Pi. The Arduino is used in classes at NTNU. This means that the Arduino is familiar and available to the students at NTNU. The Raspberry Pi is very popular world wide and comes with an operating system. This means that users can choose the programming language and that the platform is well documented. Another factor was that both the Azure IoT hub API and the Kafka library was available for Raspberry Pi, while the C code used in the Arduino would make the implementation of these two a lot more complicated. For the last couple of years NTNU has made Python the programming language in the introduction to programming classes. This combined with the versatility of the Raspberry Pi has made it the DAB of choice in this thesis.

3.2 Digital Twin Instance

In order to test the platforms as realistically as possible it was decided to deploy a DT with a physical asset to the platforms. At the start of the project multiple assets was considered for this purpose. Among the assets considered was a small scale wind turbine, the elevator at the MTP office and a cantilever provided by

SAP. The wind turbine was modeled in NX and 3D printed. The complexity of this system and the lack of material data, contributed to this option being dropped. The elevator at MTP was suggested. The platform was to be represented as a spring with a weight. This would have given the deployment a real world use case, but on examination of the elevator, no power source was found in the elevator cabin. In addition, a system of this size would be harder to trouble shoot during development. This meant that the elevator option was also dropped. At SAP they have already worked with real time structural integrity monitoring and have used "Tingen" as a demo physical asset. "Tingen", which translates to "the thing" in English, was named this because it is supposed to represent a generic physical asset in a structural integrity monitoring case[29]. This device is small enough to sit on a table top, and the fact that SAP was familiar with the setup meant that documentation and resources where available. This made "Tingen" from SAP an ideal choice.

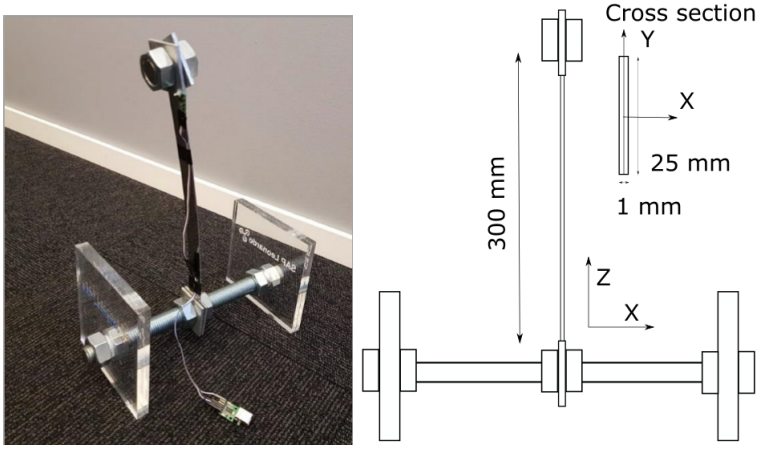

Figure 3.1: Illustrates how the physical asset is constructed.

3.2.1 Physical Properties

"Tingen" is a cantilever standing vertically, as illustrated in Figure 3.1. At the top of the cantilever there is a weight. The system works like an inverted pendulum where the weight oscillates. The way the system oscillates due to the weight on top of the cantilever imposes requirements on the deployment in regards to sample rate and filtering of sensor data. The device also have large and rapid deflections that the FMU have to calculate in real time. This makes "Tingen" an interesting DT case.

Table 3.1: Physical properties of the cantilever in figure 3.1

	Physical Properties
Length	300 mm
I	2.1 mm^4
E-module	210000$\frac{N}{mm^2}$

In Table 3.1, "I" is the area moment of inertia for the cantilever cross section over the Y axis, see Figure 3.1. "E-module" is the elasticity modulus of the material in the cantilever.

3.3 Digital Twin Platform

Two systems was considered as solutions for a DT platform. The two platforms use different technologies to achieve DT monitoring. The following paragraphs describes the platforms and formulate features that the system should implement in order to become a full DT platform. The two platforms that will be discussed is the "Tvilling digital" system developed by Simen Norderud Jensen and the PoC of an Azure based system developed as part of the specialization project of fall 2019, see Appendix B.

3.3.1 Azure Digital Twin PoC

The Azure platform was conceived as a response to the first weeks spent configuring the "Tvilling digital" system, the HBM DAB and the Catman software. There was a desire to build a system that was easier to configure and to develop further. Azure have in recent years launched an IoT API called "Azure IoT Hub" that utilizes their infrastructure in order to build scalable IoT streaming platforms.

In the fall of 2019 a PoC was built to explore the functionality and features provided by Azure. The PoC built was a streaming platform using the Azure IoT hub API to read data from an edge solution. The data was then streamed through a NodeJS server and visualized in a web browser.

Evaluation of the Azure Digital Twin PoC

The API provided by Azure made the configuration of new devices easier and less code was needed. This is because Azure holds and maintains the code for transmitting messages. This API is being adapted to new languages and platforms

presently. This makes the choice in DAB more versatile for the user. The format on the messages is string. This means that messages can be built using JSON format. The advantage with JSON is that it is a much used format that is easy to read and supported in most programming languages and platforms.

The system was, however, missing the parallel processing capability. This means that FEM simulations and filters was not yet integrated in the system. Therefore, the platform has to be developed to implement FEM simulations in order to evaluate it fully.

3.3.2 "Tvilling digital" System

"Tvilling digital" is the name given by Jensen to the platform developed in 2019[1]. It appears from the thesis written by Simen Norderud Jensen on "Tvilling digital" that there are in fact two systems in the solution. The entire system is called "Tvilling digital" and a sub system is called the "Blueprint" system. The "Blueprint" system is used to create different processes in parallel with the main execution. A class called "P" implements a set of methods like an interface. Depending on input, different processes are started by the "Blueprint" system. This makes adding new filters or other operations standardized and easier to implement. The "Tvilling digital" platform uses "datasources" to add new DT instances. This receives UDP messages and places them into Kafka topics. The system uses aiohttp to build the web API and runs on a virtual machine at NTNU. The platform is well documented with HTML resouces and a PDF file.

Evaluation of the "Tvilling Digital" System

The system is designed alongside the "Bluerig". The picture in Appendix D describes the asset. "Bluerig" is the name given to a physical asset that have been worked on in previous years. The "Bluerig" is a jack used to lift an arm and apply torsion to a connected staff. The HBM DAB with Catman as the software is used to collect data on the "Bluerig". The "Tvilling digital" system was developed along side the HBM DAB and the Catman software. The Catman software exports data in a messaging format using byte arrays. This makes the manipulation of data complicated and hard to read for the user. "Tvilling digital" have continued to use this format throughout the system. This have complicated further development.

The system offers a lot of functionality, and it works well with the current physical asset. In this thesis a new asset is to be introduced. Therefore some changes can be made to make the setup of new DT instances easier. The system is currently lacking bidirectional communication capability. By implementing this the platform can operate as a full DT platform. The Catman software has a GUI that allows data

to be sent to an IP address. In a general DT case this catman software will not be used. This means that sending messages from device to platform have to be coded. This requires both some understanding of IP addresses and how the APIs that the platform is built on works. By implementing an edge solution API that handles the configurations of the platform and standardizes the communication between platform and device. The user friendliness of the system is thus improved.

3.4 Kafka

Kafka is a messaging system, see Section 2.5, meant as a way of decoupling processes. Kafka clients are implemented in the processes to handle the messages that are sent between them. These clients are normally implemented on the same machine or in the same network. In an IoT setting, one client is needed to transfer messages from the physical asset. For Kafka to be used in this context, it has to be configured to communicate with clients outside its local network. In the "server.properties" file one can add a listeners attribute that tells the client how to connect to the Kafka cluster. The key is that the only parameter required in the client is the bootstrap server. This tells the client where to go and get the metadata about brokers in the Kafka cluster. It is the host that is passed back in the initial connection that will be the one the client connects to for reading and writing data. Below is a snippet of code that is added to the "server.properties" file in order to allow clients outside the local network to connect [30].

Code listing 3.1: Configurations added to server.properties file

```
listeners=INTERNAL://0.0.0.0:9092,EXTERNAL://0.0.0.0:19092
listener.security.protocol.map=INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT
advertised.listeners=EXTERNAL://[YOUR IP ADDRESS]:PORT,INTERNAL://localhost:9092
inter.broker.listener.name=INTERNAL
```

- **listeners:** The first line is a comma-separated list of listeners which tell Kafka the host/IP and port to bind to. In Code listing 3.1, the pattern "0.0.0.0" is used. this means that Kafka is listening to all interfaces, meaning it can be reached on all the machine addresses.
- **listener.security.protocol.map:** The second line "listener.security.protocol.map" configures the security protocol to be used for each listener as a key value pair.
- **advertised.listeners:** The "advertised.listeners" attribute contains the metadata passed back to the client in the initial request. This attribute have to be configured with the address for the server the client is going to read and write to. In an IoT setting where the client is in a different location, the "advertised.listeners" attribute must be configured with the address of the broker that the client can reach for reading and writing.

- **inter.broker.listener.name:** The "inter.broker.listener.name" attribute is used to specify the listeners to communicate between brokers. In this implementation, the system is only one broker on one machine. Therefore, this attribute is not relevant the way the system is running now.

3.4.1 Kafka Consumer Groups

A Kafka consumer group is a group of related consumers that preform a task. In the case of a DT platform the task is sending messages to a process or application. A consumer group is recognized by it's "group.id" attribute. All members of a consumer group have the same group id. Kafka then divides the partitions in that topic evenly, if possible amongst the consumers in that group. If there are more consumers then there are partitions, the extra consumers remain idle until another consumer dies before they start to consume messages.

3.5 A note about Python and FEDEM binaries

Most computers to day run a 64 bit operating system. The binaries exported by FEDEM are also 64 bit. However the standard Python installation is a 32 bit program running on a 64 bit operating system. This means that an error may occur when running the FEDEM FMU. The error will say that the 64 bit binaries can't be run on the current platform. In order to run the FMU, the user have to install a 64 bit Python interpreter. This problem has not been tested on other platforms than the current 64 bit windows 10 machine used for development.

Chapter 4

Implementation

This section describes the approach to deploy a DT instance and to develop the platforms based on the theory and the research done in the previous chapters. This section will address the process done in order to answer the initial problems described in Section 1.2.

4.1 "Tingen" from SAP

"Tingen" from SAP was chosen as the physical asset for the DT instance. In order to deploy it, the asset had to be modeled in FEDEM and a FMU had to be exported. The asset had to be instrumented with sensors and a DAB in order to stream data from the asset through the internet to the platform. The physical properties can be found in table 3.1.

4.1.1 "Tingen" FMU Generation

The cantilever was already a device that SAP had used in other structural integrity monitoring cases. Therefore a finite element model was available. The illustration on the left in Figure 4.1 shows the model when meshed using shell elements in FEDEM. The base of the model is fixed allowing no movement or rotation. The point where the force acts on the model is assigned at the top of the cantilever as shown in the figure. The sensor is placed near the point of attack on the model. In this way the measurement is taken as close to where the input force acts as possible. The illustration shows how the top nodes are connected to the point of attack.

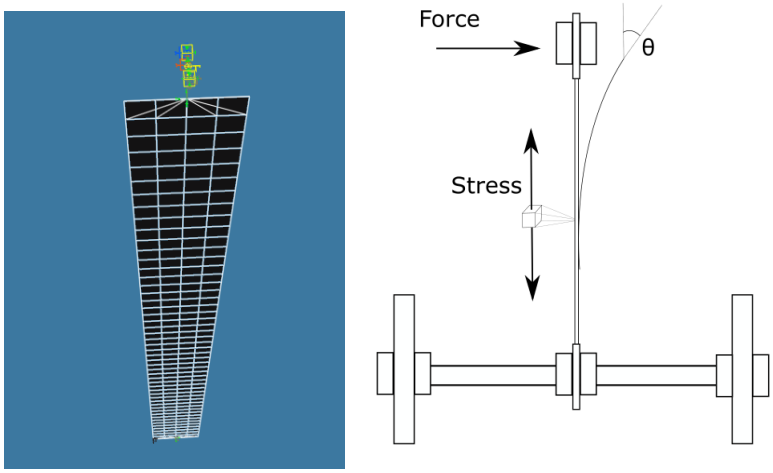

Figure 4.1: Picture on the left shows the element model of the cantilever in FE-DEM. The drawing on the right illustrates where the FMU input and outputs are on the physical asset.

The FMU generated uses one input and calculates two outputs. The input data is force applied to the top of the cantilever as shown in the drawing to the right in Figure 4.1. The force data is calculated form the angle output of the sensor, as is explained more in depth in Section 4.1.3. The outputs are stress on the cantilever beam and the angle at the top of the cantilever, also marked in the drawing to the right i Figure 4.1. This angle output from the FMU is compared to the sensor value in order to control the correctness of the FMU calculations.

Table 4.1: Model properties

	FE model Tingen
Mesh	2d_mapped_mesh
Mesh Collector	ThinShell (1mm)
Property	PSHELL Steel
Material	Steel-Rolled

4.1.2 Communication between Data Acquisition Board and Sensors

The asset was equipped with a Pololu MiniUMI-9 v5, see Figure 4.2. This sensor measures both angle and angle acceleration. The sensor have five main connections and in the current configuration the SCL, SDA, GND and VDD are used to communicate with the DAB through wires. The wires are connected to GPIO pin 1, 3, 5 and 6 on the Raspberry Pi with the pins corresponding to ground, power supply and data transfer. See Figure 4.3 for information on the pins.

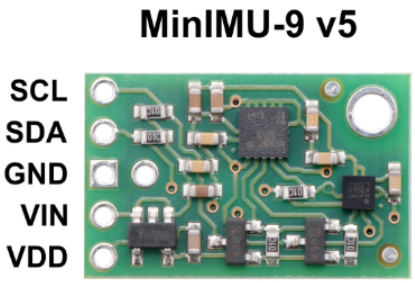

Figure 4.2: Shows a MiniIMU-9 v5.

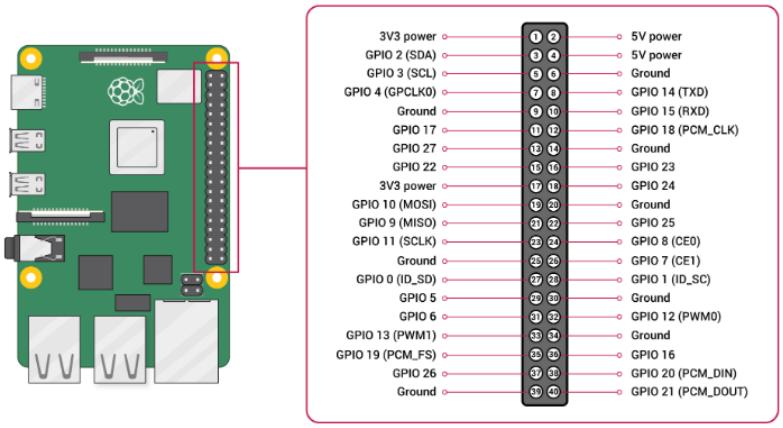

Figure 4.3: Shows a map of the pins on a Raspberry Pi with their corresponding applications.

Table 4.2: Describing the data communication from MinIMU-9 v5

PIN	Description
SCL	Level-shifted I^2C clock line: HIGH is VIN, LOW is 0 V
SDA	Level-shifted I^2C data line: HIGH is VIN, LOW is 0 V

In the standard configuration, SCL and SDA transfers data with a voltage that matches the voltage on the VIN pin on the HIGH and 0 V on the LOW. The current implementation has opted for an alternative configuration with 3.3V power supply connected to the VDD pin, and the VIN is left disconnected. This is also a configuration that is possible and described in the data sheet[31]. Table 4.2 is taken form the data sheet on the pololu MiniIMU-9 v5 and describes how data is transferred using the SCL and SDA connections with the I^2C interface.

4.1.3 Data Processing

The sensores are read using I^2C. I^2C is a standard for data transfer using the SCL and SDA connections. I^2C was invented by Phillips and is now used by most major IC(Integrated circuits) manufactures[32]. The data that is read using I^2C is translated into radians, degrees and force using scripts that was developed with the help of Runar Heggelien Refsnaes at SAP. The scripts are available in Appendix C.

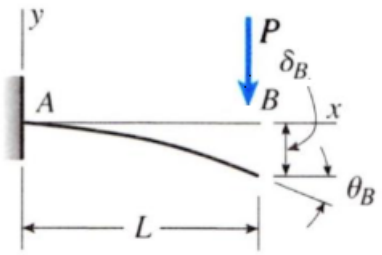

Figure 4.4: Shows the connection between force and deflection of a cantilever

The sensor doesn't directly measure force. Therefore this has to be calculated from the angle of the tip of the cantilever. By applying the inverse method, see equation 4.1 to calculate the force necessary to achieve the deflection that corresponds to the measured angle the input force is found. Figure 4.4 illustrates the correlation between deflection of a cantilever and the force applied.

$$p = \theta_B \frac{2EI}{L^2} \tag{4.1}$$

Equation 4.1: Is the inverse method used to calculate the force on the cantilever.

The angle that is sent form the Raspberry Pi to the platform is not read directly from the sensor. In order to smooth the momentary angle reading, expression 4.2 is applied. The equation sums the contribution of the change in angle and the angle acceleration at that moment. Adjusting K changes the fraction of the contribution from angle acceleration and angle to the momentary reading. An increasing K smoothes the reading by weighting the acceleration less.

$$\theta_{(i+1)} = K(\theta_i + d\theta) + K_1\ddot{\theta} \quad \text{where} \quad K = 0.85, \quad K_1 = 1 - K \tag{4.2}$$

Equation 4.2: Is used to smooth the momentary angle reading.

The code for reading data from the sensor only returns angle in positive values. The zero radians point is at 90 degrees relative to the horizontal line. I.e. the script

returns zero when the cantilever is pointing straight up and positive radians for any deflection to either side.

4.2 Azure API and Infrastructure

In order to evaluate the Azure platform that was started during the specialization project of fall 2019, parallel processing capability has to be implemented. Specifically the platform was missing the ability to do FEM simulations which is a key part of a DT platform. The system description for last years project can be found in Appendix B.

4.2.1 Implementing Parallel Processing

The FMUs generated by FEDEM must be executed in Python with a 64 bit windows operating system. Because running or using a FMU is a CPU demanding task and the platform had to be able to run multiple instances. It was necessary to create a new process dedicated to the FMU calculations.

The IoT hub API from Azure provides an easy to use messaging format that enables the DT instance to communicate with the server, but Azure IoT hub API doesn't support Python as a server language at this time. Therefore the telemetry from the devices had to be received in a Node server. This means that a bridge between the Node server and the Python process must be built. In Node a much used API is the child_process API. This allows the main Node process to programmatically create a new process where the the FEM calculations can be made while the sensor streaming still runs. The child_process API is equivalent with running a script from a terminal window. Therefore, a Python process could be started from the node process. A method in the API called "pipe()" is used to communicate between the processes created. This would be responsible for transmitting new telemetry and changing simulation configurations during execution.

4.3 "Tvilling digital" System

From the framework discussed in Section 3.3.2 it was clear that some changes could be made to improve the system. In order to make the platform a more complete DT platform. There was a desire to standardize the DT setup and make the system easier to develop. To achieve this the message format would be changed. The current solution was also missing the bidirectional communication capability. This would make the system a DT platform as supposed to a digital shadow

platform.

4.3.1 Bidirectional Communication

In implementing bidirectional communication, an edge solution API had to be developed. The platform was all ready using Kafka, therefore it was decided to use a Kafka client in the API, see Section 3.4. The Kafka client is responsible for establishing the connection with the server. The API implements methods for sending and receiving messages to and from the platform using this Kafka client. The Kafka client uses request-response type communication. In order establish a connection with the clients in the edge solution, the Kafka server had to be configured. The "advertised.listeners" attribute was changed to the IP address of the development computer, and the bootstrap server provided as a parameter in the client was changed to the same IP address. Thereby the connection was established. The client was now able to produce and consume messages from the Kafka server.

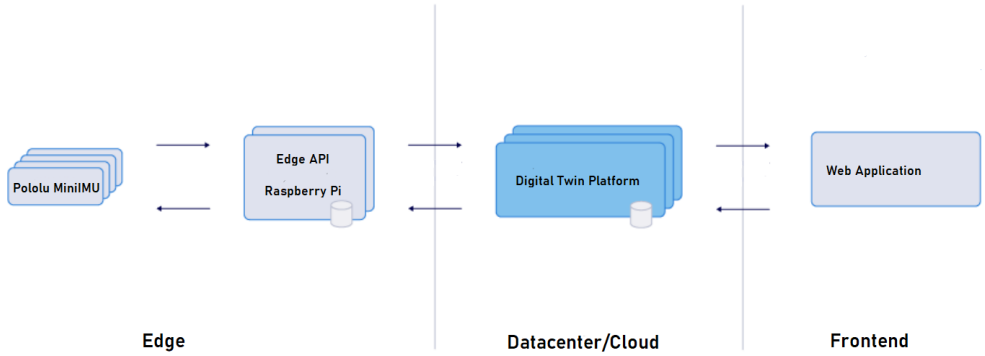

Figure 4.5: Schema of how information flows through the architecture. The vertical line indicates the separation between the edge system, the cloud and the frontend.

Figure 4.5 illustrates the information flow in the system. The physical system is measured with the Pololu MiniIMU. The measurements are read and processed with the Raspberry Pi. Then the "Edge API" places the data in Kafka. The DT platform then consumes the messages placed in Kafka and either performs FMU calculations or sends the data directly to the frontend where it is visualized.

4.3.2 Kafka Configuration

Kafka Consumer Groups

When implementing bidirectional communication it was decided to reserve a partition in every topic that is used for communication from the platform to the DT. To ensure that the consumers will divide the partitions amongst them in a desired fashion, every consumer is given an unique group ID within every topic. When consumers are not in the same consumer group, the partitions are not divided amongst them. Instead, every consumer consumes every message from every partition in the topic. This property is used to ensure that every message is read by the desired consumers. [33]

AIOKafkaConsumer

During testing it was discovered that from the creation of a new topic, either when starting a new DT instance or a new process, the time AIOKafkaConsumer would take to start consuming from this topic could be several minutes, even as much as five minutes. This is because the AIOKafkaConsumer has an attribute "$metadata\_max\_age\_ms$" that by default uses 300000 *ms*. This attribute dictates how often the AIOKafkaConsumer updates the meta data that amongst other information holds which topics the consumer is subscribed to. By changing this to 3000 *ms* the creation of new processes and addition of new DT instances is more responsive.

Chapter 5

Results

This chapter presents the results generated during this project. Several graphs are used to show how the platform is performing and how the DT instance is displayed in the platform. The graphs are pictures captured in the frontend of the platform. The values can be hard to read due to the size of the labels. Therefore some of the relevant figures have an equivalent graph in Appendix F, where the picture is enlarged so that values are easier to read. In the figure text of the relevant graphs, a hyper link to the enlarged picture is provided.

5.1 Azure API and Infrastructure.

In Section 3.3.1, during the initial evaluation of the system it was discovered that the platform was missing the parallel processing capability. From the implementation in Section 4.2, parallel processing was implemented using the child_process API. This allowed the Azure platform to perform FEM simulations on the device data. The pipe method, a part of the child_process API was intended to allow the node process to send telemetry to the process continuously and allow configurations of the FMU process while running. It was discovered that the pipe method is unable to establish a connection between processes using different languages. This meant that there was no communication between the node process and the Python process.

5.2 "Tvilling digital" System

A web API for a DT platform has been developed. In this platform, bidirectional communication has been implemented. An API for communication between the

platform and Raspberry Pi has been made to standardize the communication. The internal and external messaging format has been changed to JSON format, and a DT instance has been deployed to the platform.

5.2.1 Bidirectional Communication

A Kafka client is used to hold the connection between the device and the Kafka server in the "edge" API. The API implements a class called "connect" that initializes and holds the Kafka connection. Sending and receiving messages with this client is done through the methods "send" and "receive" in the API. Code listing 5.1 shows a simple use case for the API. First, a connection to Kafka is established. Then a message is built. The "add_to_message()" can be used to update existing values or insert new attributes. "send()" publishes the message to Kafka and "receive()" checks for incoming messages. If the message is "pause" the "pause()" function is triggered.

Code listing 5.1: Script showing how to send a single message using the edge solution API

```
from digitalTwinPlt import Connect


dtMessaging = Connect(device_id='test_device', topic='0000')

dtMessaging.add_to_message(name='Temperature', value=20)
dtMessaging.add_to_message(name='Humidity', value=78)
dtMessaging.add_to_message(name='Input_F_in', value=8)
dtMessaging.send()
instruction = dtMessaging.receive()
    for topic, partition_msg in instruction.items():
        for m in partition_msg:
            if m.value == b'pause':
                pause()
```

The maximum message rate of the "edge" API was found to be approximately 500 messages per second. The addition of bidirectional communication lowers the message rate to just under 400 messages per second. This has been tested using a message counter on a simulated device.

5.2.2 Latency

The delay in milliseconds for the last 300 data points is represented in Figure 5.1. The delay is normally below 20 ms with some peak values. These are due to the processor doing other tasks and then pausing the data processing in that period. From time stamp 10:12:30 to 10:12:34, the mean of the delay is clearly larger

than the delay before and after. This rise in the mean value was induced to the system by opening a new program during this time interval.

The delay is calculated as described in Section 2.8. Without simulations the system has been tested for the API max message rate of about 500 samples per second and the system converges to a delay of 100 ms to 250 ms per data point.

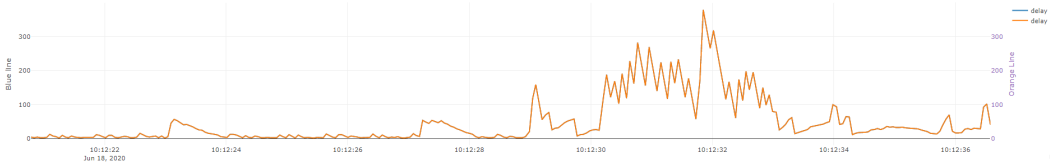

Figure 5.1: Shows the delay for the last 300 data points. The enlarged picture is provided in Appendix F.1

Figure 5.2 shows the delay in milliseconds for the last 300 data points. The data have been calculated by the FMU made for the cantilever that is part of "Tingen". The system was able to handle about 150-160 messages per second but the delay start to diverge for a sample rate of 175-185 messages per second. The delay is normally below 100 ms for sample rates of 100 samples per second. The addition of simulations have added a delay of about 50 ms to 70 ms.

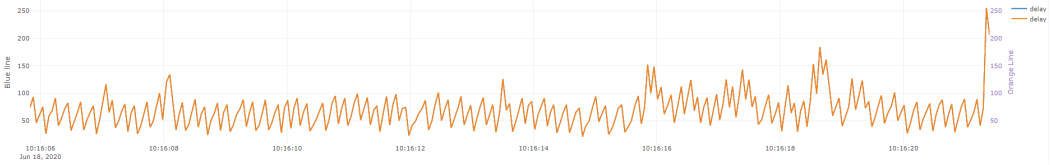

Figure 5.2: Shows the delay for the last 300 simulated data points. The enlarged picture is provided in Appendix F.2

5.3 Deployment of the Digital Twin Instance

The inverted pendulum called "Tingen" has been implemented as the physical asset in this thesis. A finite element model was derived from the physical asset by SAP and then later used in this thesis. A FMU was constructed using force as input, and the angle of the tip of the cantilever and stress of the cantilever beam as outputs. A Raspberry Pi was implemented as a DAB using the "edge" API for communication with the platform.

Figure 5.3 show the DT Instance "Tingen" as represented in the platform. The graph illustrates how the inverted pendulum oscillates. The radians on the tip of

the cantilever is represented on the y-axis. The blue line represents the sensor values and the orange line represents the FMU calculated angle. The maximum radians value is 0.8 which is just under 46 degrees. The figure shows a sample rate of about 100 samples per second. The K value is set to 0.90. This smooths the momentary read-out from the sensor.

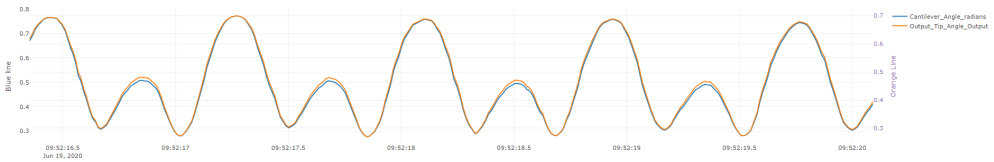

Figure 5.3: Illustrates measured and calculated values for the physical response of "Tingen" with 100 samples per second and the K value at 0.9. The enlarged picture is provided in Appendix F.3

In contrast, the Figure 5.4 with the same sample rate and close to the same oscillations are shown when the K value is at 0.1. The graph clearly contains more noisefrom the sensor.

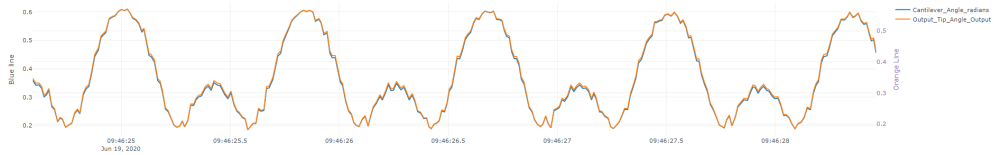

Figure 5.4: Illustrates measured and calculated values for the physical response of "Tingen" with 100 samples per second and the K value at 0.1. The enlarged picture is provided in Appendix F.4

From the y-axis in Figure 5.4 and 5.3 the values from the FMU is smaller than the sensor values. The sensor values reads almost 0.8 radians, while the FMU values are at most 0.7 radians. This makes out a difference of about 5 degrees for the largest of the angle displacements. The difference between calculated and measured angle is smaller for smaller angle displacements.

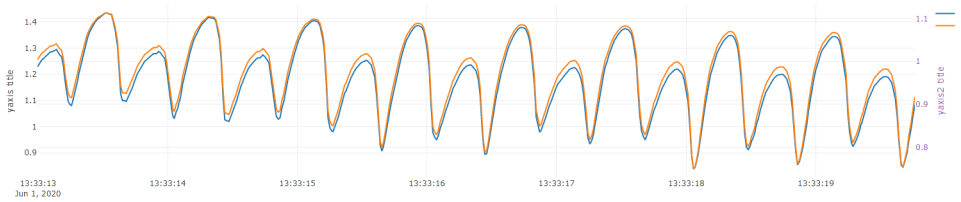

Figure 5.5: Illustrates measured and calculated values for the physical response of "Tingen" with 20 samples per second. The enlarged picture is provided in Appendix F.5

Figure 5.5 illustrates the system with a sample rate of 20 samples per second and with a large angle displacement applied to the cantilever. From Figure 5.5 the peak values of the cantilever are steadily decreasing with about the same interval for each period. The lowest values of each period corresponds to the zero radians mark. These values varies and doesn't evaluate to zero radians in every oscillation.

5.3.1 Documentation

A video has been produced showing how to start the DT platform, and also some key functionality. The video is available on youtube by following this link: https://youtu.be/JtvOd3jCTvU. A document with instructions on the prerequisites required and the installation has also been made, see Appendix A. Code belonging to the DT platform have been documented using Docstring. Sphinx has been used to generate a HTML resource in the "docs" folder in the project. A PDF document describing project has also been generated using Sphinx, the PDF is available in Appendix E.

Chapter 6

Discussion and Future Work

This section answers the subgoals from Section 1.2. The subgoals are devided in to four sections in this chapter. The section on the Azure DT platform and the "Tvilling digital" platform answers subgoal 1 and 3. The section on DT deployment answers subgoals 2 and 4. The last section suggests future work on the platform.

6.1 Azure API and Infrastructure.

During the evaluation of the initial state of the platform, it was found that the platform was lacking the parallel processing capability. This is an important part of a DT platform and it was decided to implement this feature.

The PoC was finished and the utilization of parallel processing was added. The platform was then tested on a simulated device. It was found that the process created, using the child_process API was unable to establish a pipe connection between the Python and Node processes. This meant that there was no communication between the processes after the child process was started. The sensor data then had to be batched, and a process had to be started for each batch before returning the results. During development it was discovered that the unpacking and initialization of the FMU takes several seconds. This meant that for every batch calculated an additional delay corresponding to the initialization time would be added. It would be possible to aggregate a lot of data and then send this to the child process. This would decrease the delay per message compared to sending one data point at a time. However this would introduce a delay in addition to the initialization time. This delay would be dependant on the batch size but the execution would be several seconds behind real-time. This made the child_process API in this particular implementation of the system impractical. The batching of data combined with the time it takes to unpack and initialize a FMU meant that

the delay was unacceptable for the applications of this system.

A possible way of replacing the pipe method in the child_process API could be to build two servers. The two servers would then be connected using websockets or a similar technology. This technology is familiar and is capable of handling the large amounts of data sent through the system. Because a Python server had already been constructed in previous years, and this platform could hold both the web API and the FEM simulation. This was an argument for continuing on the existing solution. In addition to this, there is also a monetary cost associated with sending messages through Azure. This means that, for a DT instance like the cantilever where the sample rate needs to be between 50 and 100 samples per second to give the system a resolution that accurately represents the physical behavior, it would either be costly when scaling to multiple DTs or when sending data over time. A remedy for this could be more data processing in the edge system in order to reduce the amount of messages from edge to cloud and in this way making the system a viable solution.

The IoT hub API is meant to simplify the messaging between edge to cloud by handling IP configurations and security. The API also provides bidirectional communication capability which was an element needed to let the system become a more complete DT platform. The API was dropped in this project because it became apparent that two servers had to be built. This would increase complexity of the overall solution and the implementation of bidirectional communication seemed to be feasible without the Azure API. The Raspberry Pi was ideally suited for this task. The main feature that was needed from the Azure API was now something that could be implemented without using it. This lead to the decision to not to continue the development of this system.

6.2 "Tvilling Digital" System

During the evaluation of the initial state of the platform in section 3.3.2. It was found that the platform was lacking bidirectional communication capability. Changing the messaging format would also help to improve the platform by simplifying further development. In an effort to standardize the DT setup, an API for communication between the edge system and the platform is developed. The platform has been tested on two FMUs produced by FEDEM and using a combination of simulated data and the deployed DT instance. As much as three DTs have been tested at once to verify the solution. The platform has been tested for one night or approximately eight hours of continues running. During the test no issues with the platform was discovered.

A problem with the FEDEM FMUs has been that they gradually use more and more memory[1]. Testing the FMU of both "Tingen" and the "Bluerig" proved that this

still is a problem which could fill up the memory of the computer and potentially crash the system.

6.2.1 Bidirectional Communication

In the implementation of bidirectional communication it was decided to use a Kafka client in the "edge" API. This has the benefit of having a very tight integration and low architectural complexity. The implementation of bidirectional communication opens the platform up to not only monitor the physical assets, but controlling them based on decisions made from the web application. The introduction of bidirectional communication moves the platform from a digital shadow platform to a more complete DT platform, see section 2.1.

The Kafka client is relatively complex and resource intensive compared to client libraries optimized for IoT[34]. This means that the DAB needed must have the resources to run both the Kafka client and the other tasks that are needed in the edge system. The client have only been tested on a Raspberry Pi and works well in this instance.

In this thesis, Raspberry Pi is used as both DAB and gateway. A possible improvement of the gateway could be to use Kafka connect and add a MQTT proxy in place of the Gateway, see figure 6.1. In this way the "edge" API could implement the MQTT protocol that is optimized for IoT. This would allow other more light weight devices to implement the edge API and communicate directly with the MQTT proxy that puts the messages into Kafka. Kafka connect offers a ready built MQTT connector but only for linux.

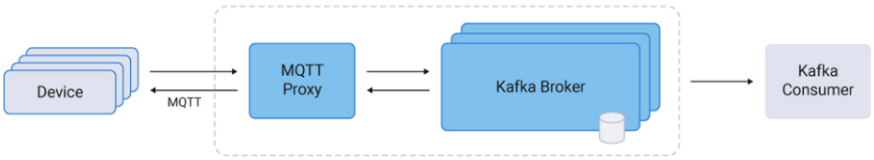

Figure 6.1: This scheme is taken from the Confluent blog on Kafka in IoT. It illustrates how devices can send MQTT messages directly to the cloud as supposed to implementing a gateway.

6.2.2 "Edge Solution" API

In order to help standardize the setup of DTs, an API for the DAB has been developed. The API is written in Python and is developed and tested on a Raspberry Pi. The API also formats messages to ensure that the platform can process them.

The API is developed so that students that are deploying a DT can focus on the instance specific code. The students will be responsible for writing logic for sensor reading and building functions that operate motors or actuators on the asset. The API will then provide a simple way of interacting with these functions. This enables the asset's sensor data to be remotely visualized through the "send()" method, or remotely activating motors or actuators on the asset using the "receive()" method.

6.2.3 Latency

Figure 5.1 and 5.2 is a representation of how the system performs in general. The performance is dependent on factors like the specific computer and other background processes running on this computer. The delay shown in figure 5.1 and 5.2 may vary depending on these factors. There are some peak values in figure 5.1. These are probably due to the processor performing other tasks and pausing the data processing at that time interval.

Comparing the latency found in Section 5.2.2 to the definition of a DT in Section 2.1 specifically that, all information that can be derived from the inspection of a physical asset should be available in the DT. The current latency of the system allows a sample rate of more then a hundred samples per second. With this resolution of the inverted pendulum oscillation, the information that is not represented in the DT is only the data that occurs between the samples. At 100 samples per second this loss is minimal. The delay with FMU calculations in figure 5.2 was measured at bellow 100 *ms* for the majority of the data points. This is acceptable for most DT instances.

For many applications of the platform, the requirements on latency may not demand such a low delay. An argument can be made for increasing the delay by implementing buffers around the processes in the solution and thereby reducing the CPU cost per message. This would allow the system to run more efficiently, meaning less CPU usage per message, but would increase the latency to some extent.

The delay introduced into the system in figure 5.1 coincided with a program being opened. This means that the processor is probably forced to pause either the FMU process or the streaming process while this task is being handled.

6.3 Digital Twin Deployment

In Section 3.2 "Tingen" from SAP was chosen as the DT instance. The physical properties of the system makes it an interesting DT case. Figures 5.5, 5.4 and 5.3 show "Tingen" oscillating without any input to the system. From Section 4.1.3

on data processing, the radians values are only returned in positive values. This gives the graph the distinctive look with rounded tops and sharp minimums. The cantilever in "Tingen" is slightly deformed and leans to one side. This means that the inverted pendulum oscillates around a point of equilibrium that is not zero radians. This is why every other maximum in the graph is larger than the one between.

A physical behavior of the inverted pendulum is that the arm is moving at it's fastest close to the zero radians mark. Figure 5.5 shows the inverted pendulum with a sample rate of 20 per second with a large angle displacement applied to the system. The lowest point between each peak should always evaluate to zero, this is it not the case. A possible explanation is that the pendulum passes the zero radian point so fast that the sample rate is unable to capture this value. The sample is then taken at some point around the zero radians mark, giving the graph in figure 5.5 a minimum that varies.

In the situation in figure 5.5 the platform is still representing some of the operation conditions of "Tingen", but a significant amount of information is now lost between the sampling points. The sampling is done at steady intervals and will never sample the cantilever at the exact point where it passes zero radians. Therefore, it is important to have a sample rate that is fast enough to represent the system no matter the timing of the samples.

The way the platform represents the DT allows for all the needed functionality to call the platform a full DT platform. However, the FMU that is used in this paper only provides two outputs for the asset. This is less than the definition of a DT defines as optimum, see Section 2.1. Even though this is the case, before a FMU is created the user has to choose the outputs, by selecting key values that represents points of max stress or torsion. These values can represent the points of interest in the structure, without calculating all nodes in the FE model. Therefore the DT will not be optimal but can still add value to the sensor data. The same is true for the inputs of the model. The amount of sensors placed and the chosen inputs restrict the overall picture of how the asset is preforming in reality. If the inverted pendulum in this thesis is subject to rotation around the cantilever arm, the torsion that arises will not be picked up by the sensors. Therefore, the DTs as they are used today only represents a picture of the asset as defined in advance. Choosing attributes with knowledge and making good assumptions, the DTs can still represent a good picture of the operational conditions of the asset.

Sources of Error

The material properties used are the same as SAP has used for the model earlier and have not been verified extensively. In figure 5.3 there is a deviation between

the calculated angle from the FMU and the angle measured by the sensor. The calculated angle from the FMU is being compared to same the sensor angle used as input. This means that any error that is present in figure 5.3 have to originate in the angle calculations. To calculate the angle from the FMU, the force corresponding to the sensor angle is found using the inverse method. This force is then used as input in the FMU before the angle output from the FMU is compared to the original angle. Both in the FMU calculations and in the inverse method, material data is used. These properties will have some variations with temperature, fatigue and imperfections that will cause errors in the calculations.

6.3.1 Prototyping

Web Application

To test the Azure based API and the "Tvilling digital" API a web Application has been developed for each. This made it easier to test features and to make sure that the features worked. These web applications are only meant for personal testing and is fairly cluttered and hard to use. Therefore a video of how to do basic operations on the web application for the "Tvilling digital" API has been made. The video can be found on youtube with the following link: https://youtu.be/JtvOd3jCTvU. This is to help further development and hopefully make the process of getting started with the platform easier.

Simulated Devices

During the development of the system, two simulated devices have been used extensively. They are documented and part of the project under the folder "digitalTwinPltAPI". These simulated devices helps to show how the API is is used. They also help the further development of the system by adding a data source that is fast to deploy and easy to customize.

6.4 Future Work

- **Data Base:** The addition of a data base would be needed in both authentication of users and to implement some historical data features. It would also be an important part in the development of security in the platform.

- **Security:** This has not been a focus in this paper but is important to the platform before it is to be deployed. Currently the platform is vulnerable

due to the fact that any one can produce messages to Kafka and consume messages from Kafka given a topic name.

- **Blueprint system:** The platform still uses the blueprint system. The only blueprint currently in use is the FMU blueprint. The addition of new blueprints like filtering, event triggers and aggregation of data would add functionality to the platform.

- **FEDEM FMU:** Both the FMUs exported from FEDEM have worked well in this thesis. In the future, FMUs from a different FMU builder should be tested to verify that the FMU blueprint can generalize. There are also some potential improvements that could be made to the FMUs. The FMUs currently have a 64 bit binary, a second 32 bit binary could be implemented, thereby avoiding the set up of a 64 bit python environment. The FMUs also have a interval of values that are accepted and if data out side this interval is used the FMU crashes. When inspecting the FMU there is no way of seeing what values are in this interval. Adding this information to the model description would help in testing and development.

Chapter 7

Conclusion

The goal of this thesis was to deploy a DT instance and to evaluate and develop the current platforms. "Tingen" from SAP has been deployed as a DT. The platforms have been developed further in accordance with the definition of a DT.

The Azure PoC was finished, but it was decided that the development of this platform would not be continued. This was because the solution would not reduce the complexity of the overall system and because of monetary constraints. The PoC was only tested with the child_process API and batching. No other solutions was tested before it was decided to stop the development and start working on the "tvilling digital" system. In a different case where the requirements on real-time is different, the API may be a better option.

Using the code and framework from "Tvilling digital" a DT platform has been created. This platform implements bidirectional communication in order to become a full DT platform. In addition, the platform uses JSON format on both internal and external messages in order to increase the user friendliness and simplify further development of the system.

"Tingen" was used as a DT instance. The platform is able to represent the physical asset and characterize the physical behaviors of the system. The FMU and the scripts used to transfer data and calculate values for the cantilever gives a good picture of the operating conditions of the system. The angle calculated and the angle measured have some deviations which is to be expected.

An edge solution has also been implemented. In an effort to standardize the DT setup, an API for communication with the platform has been created. The API has only been tested on "Tingen". However, the methods implemented are general and an arbitrary DT instance should be able to utilize methods like "send()" and "receive()" in a DT deployment.

Bibliography

[1] Simen Jensen. 'Building an extensible prototype for a cloud based digital twin platform'. In: *NTNU Master Thesis* (June 2019).

[2] Fredrik Dahlqvist Mark Patel Alexander Rajko Jonathan Shulman. *Growing opportunities in the Internet of Things*. https://www.mckinsey.com/industries/private-equity-and-principal-investors/our-insights/growing-opportunities-in-the-internet-of-things. 2019.

[3] Bernard Marr. *7 Amazing Examples Of Digital Twin Technology In Practice*. 2019.

[4] Samuel Greengard. 'Digital Twins Grow Up'. In: *ACM news* (June 2019), pp. 1016–1022.

[5] Pascoa J. Trancossi M. Cannistraro M. 'Can constructal law and exergy analysis produce a robust design method that couples with industry 4.0 paradigms? The case of a container house'. In: *International Information and Engineering technology Association, IIETA* 5 (June 2018), pp. 303–312.

[6] Jean Thilmany. 'Identical Twins'. In: *The American Society of Mechanical Engineers, ASME* (Sept. 2017).

[7] Michael Grieves and John Vickers. 'Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems'. In: Aug. 2017, pp. 85–113. ISBN: 978-3-319-38754-3. DOI: 10.1007/978-3-319-38756-7_4.

[8] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes and Wilfried Sihn. 'Digital Twin in manufacturing: A categorical literature review and classification'. In: *IFAC-PapersOnLine* 51 (Jan. 2018), pp. 1016–1022. DOI: 10.1016/j.ifacol.2018.08.474.

[9] *Gartner's Top 10 Strategic Technology Trends for 2017*. https://www.ge.com/research/offering/digital-twin-creation.

[10] Adrian Grigoriu. *The Integrated Architecture of the Digital Twin of the Organization, Enterprise Architecture Tool and Enterprise Monitoring Systems*. https://it.toolbox.com/blogs/adriangrigoriu/the-integrated-architecture-of-the-digital-twin-of-the-organization-enterprise-

architecture-tool-and-enterprise-monitoring-systems-010920. 2020.

[11] Amazon. *What is cloud computing?* https://aws.amazon.com/what-is-cloud-computing/. 2019.

[12] Qahtan Shallal and Mohammad Bokhari. 'CLOUD COMPUTING SERVICE MODELS: A COMPARATIVE STUDY'. In: *IEEE Network* (Mar. 2016), pp. 16–18.

[13] *What is serverless computing?* https://www.cloudflare.com/learning/serverless/what-is-serverless/.

[14] *About us - Raspberry Pi*. https://www.raspberrypi.org/about/.

[15] Pethuru Raj. *Advances In Computers - A Deep Dive into NoSQL Databases: The Use Cases and Applications*. 2018. Chap. 5.4.2 Apache Kafka.

[16] Paul Stokes. '4 Stages of IoT architecture explained in simple words'. In: *Medium.com* (Dec. 2018).

[17] https://www.confluent.io/blog/iot-with-kafka-connect-mqtt-and-rest-proxy/.

[18] *IB (Integration Broker)*. https://www.gartner.com/en/information-technology/glossary/ib-integration-broker.

[19] *Use cases*. https://kafka.apache.org/uses.

[20] Nick Vogt. *What's The Difference Between External And Local IP Addresses?* https://www.h3xed.com/web-and-internet/whats-the-difference-between-external-and-local-ip-addresses. 2014.

[21] Stephanie Crawford. *What is an IP address?* https://computer.howstuffworks.com/internet/basics/what-is-an-ip-address.html. 2001.

[22] Modelica Association. *Modelica Association Projects*. https://modelica.org/projects.

[23] Fraunhofer Institute for Integrated Circuits IIS Jens Bastian. 'Master for Co-Simulation Using FMI'. In: 014 (63 2011), pp. 115–120.

[24] James Martin. *Programming Real-time Computer-systems*. IBM system Research Institute, 1965.

[25] Dragan Marinkovic and Manfred Zehn. 'Survey of Finite Element Method-Based Real-Time Simulations'. In: *Applied Sciences* 9 (July 2019), p. 2775. DOI: 10.3390/app9142775.

[26] Mohit Gupta. *What the hell is "ASYNC" ??* https://medium.com/magentacodes/what-the-hell-is-async-65aea57146ef. 2017.

[27] Adam Robinson. *Asynchronous vs synchronous execution, what does it really mean? [closed]*. https://stackoverflow.com/questions/748175/asynchronous-vs-synchronous-execution-what-does-it-really-mean. 2009.

[28] Urban Institute. *Using Multiprocessing to Make Python Code Faster*. https://medium.com/@urban_institute/using-multiprocessing-to-make-python-code-faster-23ea5ef996ba. 2018.

[29] Bjorn Haugen. *quote from Bjorn Haugen*. 2020.

[30] Robin Moffatt. *Kafka Listeners - Explained*. https://www.confluent.io/blog/kafka-listeners-explained. 2019.

[31] *MinIMU-9 v5 Gyro*. https://www.pololu.com/product/2738.

[32] *I2C info - I2C Bus, Interface and protocol*. https://i2c.info/.

[33] *KAFKA ARCHITECTURE: CONSUMERS*. http://cloudurable.com/blog/kafka-architecture-consumers/index.html. 2017.

[34] Dominik Obermaier and Ian Skerrett. *HiveMQ and Apache Kafka - Streaming IoT Data and MQTT Messages*. https://www.hivemq.com/blog/streaming-iot-data-and-mqtt-messages-to-apache-kafka/. 2019.

Appendix A

System Setup

This section explains how to setup the system on a personal computer to either test the system or to continue developing. The steps are kept short and on reasoning or theoretical background will be given. The system is reliant on a FEDEM lisenced network, therefore this set-up should be done either on the NTNU network or with a VPN. If you are using a VPN make sure to connect the VPN before you start kafka. If the set-up is preformed on the NTNU network the Raspberry Pi mush be configured to connect to eduroam. This proedure is explained in the following link: "https://autottblog.wordpress.com/raspberry-pi-arduino/connecting-raspberry-pi-to-eduroam/"

A.1 Prerequisites

- **python 3.7:** This have to be able to execute 64 bit binaries.
- **Kafka:** Installation can be done form this "link:https://kafka.apache.org/quickstart" The kaka version used in the development is 2.3.
- **Code for DT platform:** The code is available in the zipped file called "pythonDTsolution" that is inside the zipped file delivered with the PDF of the thesis. The code is also available on github following this link: https://github.com/espenmarstein/pythonDTsolution. The FMUs are only available in the zipped file, and have to be acquired from SAP separately if you use github.

A.2 Configurations

- **Kafka:** Open the kafka folder from your installation. navigate inside the the config folder and open the server.prperties file. Add the following lines under the section called "Socket Server Settings"

Code listing A.1: Configurations added to server.properties file

```
listeners=INTERNAL://0.0.0.0:9092,EXTERNAL://0.0.0.0:19092
listener.security.protocol.map=INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT
advertised.listeners=EXTERNAL://[YOUR IP ADDRESS]:PORT,INTERNAL://localhost:9092
inter.broker.listener.name=INTERNAL
```

  Substitute "[YOUR IP ADDRESS]:PORT" with your IP address this can be found using the "ipconfig" command in the terminal.
- **digitalTwinPlt.py** The "Kafka_server" property have to be configured on the initial run. This must be the same IP address and port as configured in code listing A.1

A.3 Initial Run

Install the requirements in requirements.txt file using "pip install -r requirements.txt" when you are in the project folder. After this follow the instructions in the video provided with the report to complete the platform set-up. The video is also available with this link: https://youtu.be/JtvOd3jCTvU

Appendix B

Specialization Project Report

Exploring Azure as cloud provider for digital twin monitoring

Espen Marstein Sandtveit

Fall 2019

Summary

This project looks at the definition of a digital twin and formulates requirements for a digital twin platform. Then the current Blueprint system is investigated and evaluated before a new architecture is proposed in Azure. The project has been written under the supervision of Bjørn Haugen and Terje Rølvåg.

A prototype has also been developed to test the technology choices. The prototype is a streaming platform that receives and visualises data from a simulated Raspberry Pi circuit. A quick start have been written to help deploy the system, see appendix A.

Contents

Chapter 1

Introduction

The Department of Mechanical and industrial engineering (MTP) has a goal to develop a digital twin platform that will serve as a tool for research in the field of real time structural integrity monitoring and predictive maintenance. The goal of this project is to explore Azure as a possible cloud provider to host the software for digital twin monitoring.

1.1 Background and motivation

In recent years the introduction of affordable 4G, and soon 5G internet has accelerated the development in of the field of internet of things (IoT). This has reduced the cost of sensors, and some industries are now looking to make use of their IoT data by implementing digital twins. By implementing digital twins businesses are giving more value to the data collected. Digital twins enable several benefits to businesses. Regarding digital twins with physical assets there are two mayor areas. The first is structural monitoring. This can be used to reduce the need for onsite presence of personnel in dangerous or remote places. The other one is predictive maintenance. By providing more data to models that predict remaining useful lifetime (RUL) their effectiveness can be greatly improved.
Today there are several companies working on digital twin solutions that answer industry needs but these are costly and not well suited for the academic work at MTP. Digital twin is broad term that is loosely defined but it refers to a software representation of a real process or a physical asset. In this thesis a digital twin refers to a physical asset that can be represented with finite element model (FE model). The FE model will be provided by FEDEM and the FEDEM software is responsible for the simulations preformed based on the sensor data. This project also tries to generalize the setup of these digital twins and make this into a process that can be performed cheaply and quickly.

1.2 Problem formulation

This report explores the concept of a digital twin and formulates requirements for a digital twin platform. Then the report looks at the Blue Print system and explores the possibilities of making a cloud platform in Azure. The hope is then to introduce new functionality and reducing complexity while reusing parts of the Blue Print system. The project work can be split into four major goals:

- Formulate functional requirements for a digital twin.
- Explore the Blue print system.
- Explore Microsoft Azure and formulate a architecture
- Make a prototype of the new architecture.

1.3 Requirements

This section describes the different components of the digital twin system that is required to give the system it's desired functionality. The requirements listed make up a minimal viable product and not all the requirements are in the scope of this project. They are listed because these are the sub goals the project have worked to achieve.

- **Physical asset**:
  - Finite element model.
  - Measure physical conditions.
- **Sensors**:
  - Capture relevant data.
  - Stream in real time.
  - Quality for sensor is adequate.
- **Data acquisition board**:
  - Signal processing.
  - Bi-directional communication.
  - Internet connectivity.
  - Compatible with Azure.
- **Azure Cloud**:
  - Event consumer
  - Data lake, storage.

- Python backend for FEM simulations.
- Server hosting Web portal.

- **Web Portal**:
  - User interface.
  - Different acquisition boards.
  - Different sensors belonging to a acquisition board.
  - Visualization of real time streaming sensor data.
  - Visualization of FE simulations.

Chapter 3

Technology research

In order to begin the process of developing a platform for digital twins it was important to research the different APIs and infrastructures for digital twins provided by Azure. There is already work on developing a digital twin platform at NTNU and this paper aims at reusing as much as possible form this solution while adding benefits form Azure. The documentation provided by azure and an extensive research into the Blueprint system forms the foundation for the literature study in this chapter. The goal is to combine the Blueprint system with a cloud platform in azure to answer the requirements listed in section 1.3.

3.1 Azure

In the field of digital twin azure offers a few different types of services. These services differ in how much functionality they offer. All Instances below are compared to the list of requirements in the implementation chapter 4.2.

3.1.1 Event consumers

- Azure Event Hub: Is a big data streaming platform and event ingestion service. Azure is able to receive and process millions of events per second. This makes it ideal for real time streaming. It is a PaaS platform but integrates Azure functions which means that a serverless architecture is possible. This helps speed up development and makes the architecture highly scalable. Azure event hub is supported in many programming languages for flexibility in development. Figure 3.1 below shows the event hub architecture. Events are created on the left and then moves to the right, before ending up at either some visualization or calculation. (microsoft 2018a)

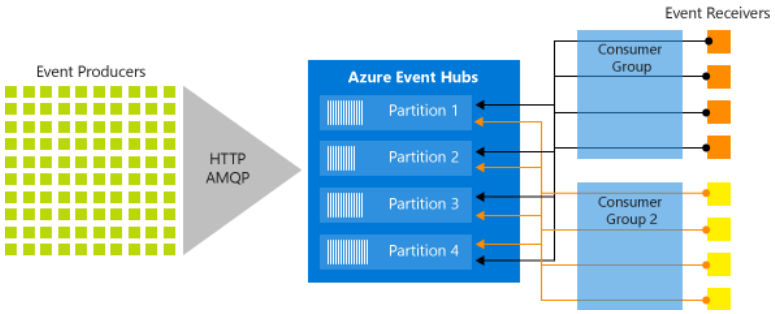

Figure 3.1: Shows the scheme for the event hub architecture

event producers is any entity that is sending data to the event hub. I the case of Digital twins this is acquisition boards that have collected sensor data. A single event can be data from several sensors either at a sinlge time or several recordings over an interval. **Partition** only reads a subset or partition of the incoming data. **Consumer Group** subscribes to any number of partitions. A consumer group enable applications to each have a separate view of the event stream. This is practical when data streamed is either personal or only for a small subset of consumers. This way the wind turbines from Equinor is not visable for anyone else that is using the digital twin platform. **Event receivers** any entity that reads the event data from the event hub. Any consumer group may have may event receivers as is shown in figure 3.1.

- Azure IoT Hub: Is is another type of the Event hub, but it offers some more capabilities. The IoT hub comes with a extended API that includes bidirectional communication. This enables the digital twin to receive data and commands form the cloud just like the cloud receives data from the device. IoT hub also supports several messaging formats like device-to-cloud telemetry, file upload form device and request-reply methods that enables control of device form the cloud. Like with the event hub the IoT hub scales to millions of devices, and also provides a secure communication channel between device and cloud. Azure IoT hub also provides a device provisioning service. This enables the automatic provisioning of ids to the devices (microsoft 2018b). This is used when new devices is registered to avoid doing the registration manually.

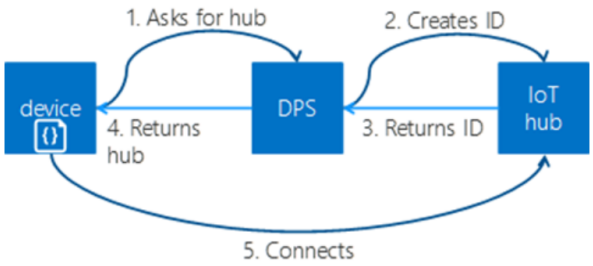

Figure 3.2: Shows how the Provisioning service assigns ID's.

- Azure Digital twin: "Azure Digital Twins is an Azure IoT service that creates comprehensive models of the physical environment.". This the first line in Azures documentation of the azure digital twin. The digital twin service from Azure provides the capability of a "spatial intelligence graph" this graph models the relationship and interactions between devices. This is the service that offers the most functionality and is built using IoT hub to connect to devices witch again is a type of event hub.

3.2 Blue Print System

This is the solution developed by students at NTNU in previous years. The Tvilling digital or Blue print system is built to be a start point for a general purpose digital twin system. The current system is built to be general and is designed so adding filters, solvers or other FMUs(othr digital twin) is possible. Even though this is the case the system is specifically designed for the HBM data acquisition system and catman as the software this are used on the blue rigg that is stationed at MTP faculty. Both the Catman software and the HBM data acquisition board are systems that are relatively expensive. The Catman software is developed to allow data streams to specific IP addresses but it is worth here to mention that the task of configuring Catman with the blue print system was a fairly complicated procedure. The set up was not stable if the system used wifi and an Ethernet cable had to be used in order to have a stable stream from the data acquisition board. This system is open sourced and as much as possible will be reused for this system. However, there are parts of the system that can be replaced to add more functionality and simplify development.

3.3 Raspberry Pi & Arduino

From the research stage it follows that the data acquisition board needed was a board made for development. The main users for this system will be students at NTNU therefore reducing cost and complexity in deployment is critical. The data acquisition board should be familiar to

the user in terms of setup and programming language. The two biggest vendors are either the Arduino or the Raspberry Pi. In terms of capabilities both have boards that deliver all needed functionality and more.

The Arduino is a used a lot in classes at NTNU and in the mechatronics lab several boards are available. The board is required to be able to connect to wifi and none of the Arduino's available that the Mechatronics lab have this built in. This means that a new board will have to by purchased no matter if it's a arduino or a Raspberry Pi. Following Azure's github repositories it's clear that the APIs for Arduino is not yet ready while the Raspberry Pi who supports several programming languages and thereby several API's are ready to use for the Raspberry Pi. On closer examination it was also discovered that the Arduino and the other APIs are very similar, for instance the functions have the same names like "start", "stop" and so on (https://github.com/Azure/azure-iot-arduino/graphs/contributors n.d.) (https://github.com/Azure/iot/grapl n.d.). A prototype made with a Raspberry Pi will therefore be highly transferable to the Arduino when this API is done. Also, the Raspberry Pi is well suited for this task.

Chapter 4

System overview & Implementation

4.1 System overview

This section takes the previously discussed technologies and requirements and explains how each part of desired functionality is answered. The system is designed to be able to fulfill the definition of a digital twin as defined by both Kritziger 2.4 and Dr. Grieves 2.1. Figure 4.1 show the information flow in the architecture of the system as it is currently implemented. The Raspberry Pi is placed near the physical asset. This asset is then instrumented with sensors that transmit data to the Raspberry Pi. A WiFi connection between the Raspberry Pi and the cloud is used to transmit data directly to the cloud. Here illustrated by the cloud surrounding all the components in the cloud, see figure 4.1. The data received is then processed and the visualisation is accessible through a web browser.

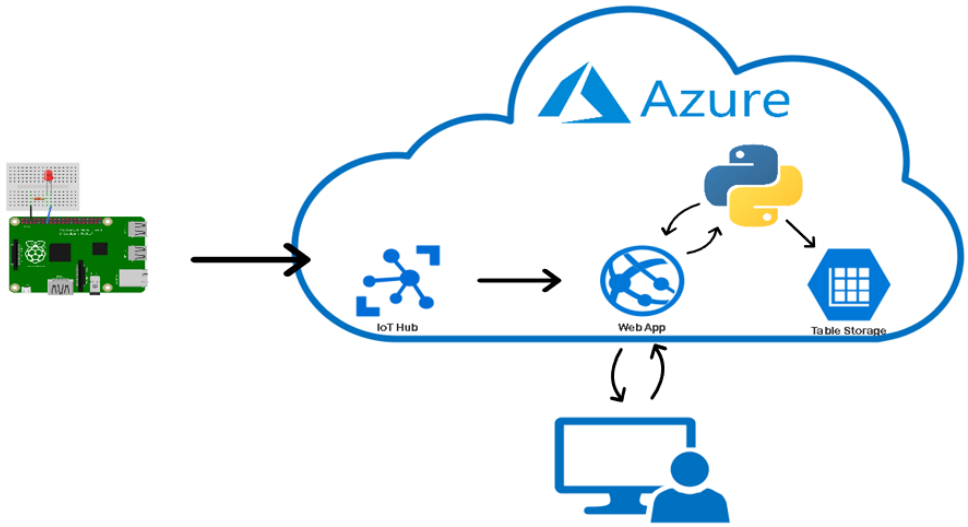

Figure 4.1: Show the scheme for the digital twin architecture

4.2 Implementation

4.2.1 Data Acquisition System

The sensors and the Raspberry Pi makes up the data acquisition system in the digital twin system. The Raspberry Pi was chosen because it is a versatile development tool that is believed to be accessible to many students and familiar in setup, there by make the system more user friendly. The fact that the API was ready was also a important factor in the choice of DAB. This system have for the benefit of development and time constrain been substituted for a online simulator and a Python script in this iteration of the system. In real world applications the Raspberry Pi will receive data from the sensors and through the internet send data directly to the cloud using the API provided by azure. By using a Raspberry Pi the goal is to reduce cost and complexity of deploying a digital twin for the end user. Reducing the deployment complexity and cost will make the system more accessible for students at NTNU.

To help explore the azure API and also to help generalize the system three different devices have been used to send telemetry to the IoT hub. The devises used are a mobile phone with and android application, a Python script called "blueRiggSimulation.py" running on a laptop, and also a online simulated Raspberry Pi with a script called "raspberryPi.js" have been used to send simulated telemetry to the cloud. The Raspberry Pi is the device that the system will continue using but the Python script have also been very help full in testing and development. The API provided by Azure formats all events in a JSON format. This means that the cloud architecture is device independent. All events handled by the IoT hub are just JSON objects and devices are treated the same way.

4.2.2 Cloud Architecture

The rest of the digital twin system is contained within the Azure cloud. In the cloud a server is constantly running two scripts server.js and Chart-device-data.js. server.js is responsible for the logic of the entire system and Chart-device-data.js handles the visualization and user interactions. The information flow of the cloud architecture starts with an event arriving at the IoT hub. This is then routed to the consumer group. The server.js is listening to the consumer group and is triggered when a event arrives there. Server.js then sends data to FEM.py. this is where the Blue Print system will be implemented, for now only delay is calculated. Then the data is sent back to server.js. The message is then broadcast through websocket to Chart-device-data.js for visualization.

Event Consumer: The IoT hub is here implemented as the Event consumer. The IoT hub provides the necessary functionality through the IoT hub API. Azure digital twin was also con-

sidered but this only offered more solutions that is not yet part of the system and there by complicating the development. The digital twin service uses the azure iot to keep track of the physical assets and extending the system with digital twin service can therefore be done if needed in the future. The Azure IoT hub that is implemented is configured to send the telemetry to a consumer group that server.js is listening to.

server.js: This is the main logic for the backend and is constantly running. The server.js calls other scripts as needed during execution. This script is listening for events arriving in the consumer group and as an event is picked up by server.js the event is parsed in to a message. This is done using the event-hub-reader.js. Then it is sent to the Python script FEM.py. In the Python script the delay is calculated and the result is sent back to server.js. A websocket connection is then used to then broadcast sensor data to Chart-device-data.js for visualization.

FEM.py is responsible for the calculations done in the system. It was important in the prototype to have the backend be able to spawn python processes. Spawning a Python process means that calculations are separate processes with a Python environment. This is so that the Blueprint system can be implemented at a later stage.

Chart-device-data.js is as previously stated the logic for the frontend. This class keeps the last 50 data points in memory and plots them. This also handles the user interactions witch for now is only switching DAB. Figure 4.2 shows how the temperature is plotted with delay. Chart-device-data.js with the CSS and HTML files are taken from a tutorial on azure IoT hub and reused of this purpose with only minor changes.

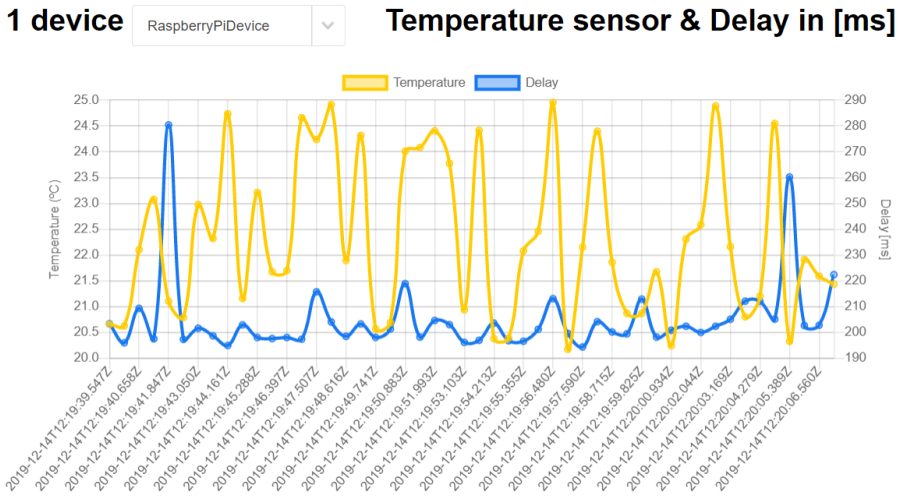

Figure 4.2: Shows the user interface.

Table Storage is used for the long term data storage. This was the cheaps form of storae while stil being table based. This was implemented but then later removed because the frontend had no way of handling the data. This meant that sensor data was just augmenting in the cloud with out any way of using it or removing it with out scripting or removing from within the azure portal manually.

Chapter 5

Results

The goal of this thesis was to investigate the concept of a digital twin and explore Azure cloud services as a potential cloud provider for for digital twin platform. The project looked at the current solution the Blueprint system hosted here at NTNU and investigated the possibility of adding functionality while simplifying the current solution and further development by using Azure.

5.1 digital twin

The concept of a digital twin in this thesis is formulated the definition from both Dr. grieves and form Kritzinger. In the definition of a digital twin provided by Dr. Grieves, he and John Vickers states that "the Digital Twin is a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level.". In order to address this all digital twins of the system will be represented by a FE model. In this way FEM calculations can be done, and the physical asset described in great detail. Kritzinger proposes a digital twin classification where the connection between the digital twin and the physical asset must be automatic both from the physical to the virtual asset and the virtual to the physical asset in order to classify system as a digital twin. By using Azure bi-directional communication it is possible with automated communication in both directions to be implemented.

5.2 The Blue Print System

A system for digital twin monitoring was already developed by Simen Jensen at MTP. This system was designed to be extendable with filters and solvers but is specific for the data acquisition board and software on the Blue rigg at MTP. The system manages to stream data and by FMUs

exported form FEDEM it also manages to do real time FEM simulations. By combining the current Blueprint system with the platform in azure a great deal of code that is both specific to the Blue rigg and that handles the byte streams will then be inside the Azure API. This cuts down the complexity of the system greatly and makes the system more usable for students at NTNU. After a lot of research into the Blueprint system there are identified parts that are possible to reuse for the new platform. These parts will require some rewriting but the main functionality will be the same.

5.3 prototype system

In order to fully understand and to see if in fact a true digital twin could be developed using Azure a proof of concept have been developed. The digital twin system is currently a streaming platform that receive data from a data acquisition board and displays this in a web portal. In order to investigate if the different requirements formulated in 1.3 could be met the system provides some basic functionality that proved the the different technologies are present and work together. See chapter 4

There was also a concern about the delay caused by routing data through Azure. In figure 5.1 the delay for 50 events are plotted. The delay is calculated by running a python script on the local computer and using the computers internal clock to give the event a time stamp that corresponds to the time the event is sent. Then the message is processed in azure before the message is received back at the same computer and the time differential is calculated before it is plotted. Figure 5.1 shows how the delay is plotted the y axis on teh right show teh delay in milliseconds and the time stamp is plotted along the x axis. The max value is almost 205 ms and the lowest is almost 165 ms. This delay is fairly typical but variations do accrue. In early November the same test was carried out and this time the mean value was closer to 130 ms. In addition to the fact that the mean varies, some values occur that are fare bigger then the mean. This have been observed to be almost 800ms and could of course be even greater. The test does not have statistical background for the numbers presented but serves more like example values.

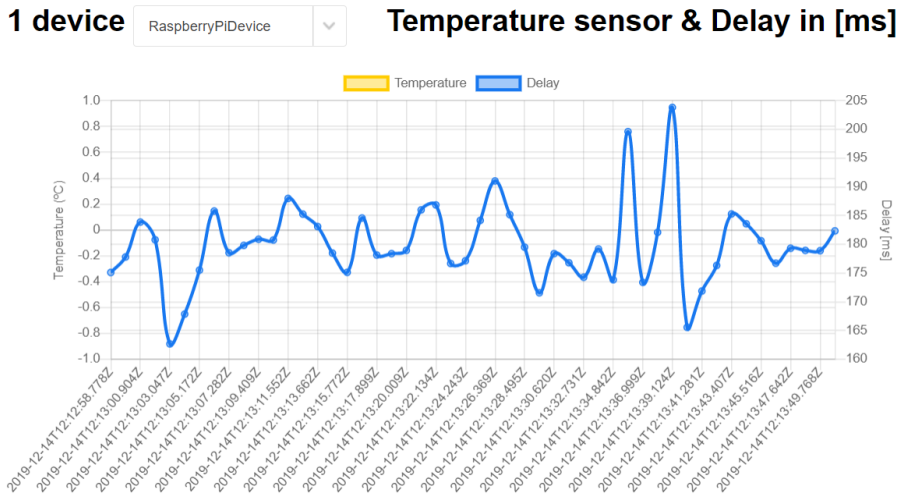

Figure 5.1: Show the scheme for the digital twin architecture

Although some delay is inflicted on the system by using azure. This values are well with in acceptable. In order to implement some type of signal processing sensor data have to be aggregated for a couple of seconds to few seconds before the data is sent. In this case the delay caused by routing through Azure will be relatively small compared to the seconds of delay from the signal processing. The recorded latency for the Tvilling Digital system without any processing was between 100 and 200 ms(Norderud Jensen 2019). This is about the same delay that is observed in the Azure system.

Chapter 6

Discussion

6.1 Requirements

Requirements have been formulated and based on the papers written by Kritzinger and Dr. Grieves. The requirements are for a minimal viable product and the proof of concept have not been able to incorporate all desired features. It was necessary to redefine the requirements from the earlier papers written on development of a digital twin system. Because the new system incorporates both a self-defined data acquisition system and a cloud architecture with storage. The project thesis and the master thesis by Simen Jensen(Norderud Jensen 2019) have been used as starting point for the requirements and then changes and additions have been as needed.

6.2 Digital Twin Definition

It appears that there is no one definition for a digital twin. There are many different definitions proposed from different papers and these are variation of the definition proposed by Dr. Grieves (Kritzinger et al. 2018) giving the understanding that a digital twin is a digital counter part to a physical system. Instead of a new definition Kritzinger proposed a classification system of the integration level between the physical and virtual. By combining these two the digital twin for this specific system is defined.

6.3 The Blueprint system

The Blueprint system works as a good starting point for further development and parts of the solution is applicable to the new system. There are however problems with the systems technical debt. The better part of 2 months was spent trying to get the system running. This was to a large extent caused by the HBM acquisition board and the Catman software. These are the parts

of the system that are specifically developed for the blue rigg system and will not be extended in the new version. Out of the parts that are transferable it is unclear exactly how much rewriting is required.

6.4 Prototype

The prototype developed in Azure serves as a proof of concept for the technology research done. The prototype shows how technologies that are needed work and that they are available in a Azure environment. Some work still remains on the prototype, it was a goal to run simulations during this semester but this have not been achieved. This is in part to technical debt in the Blueprint system but a change in thesis half way in to the semester have also effected the time spent on development.

Initially there were concerns regarding the delay caused by using azure both from geographical location and azures allocation of resources. The delay was measured at random times during the semester and was found to vary from time to time and to have some messages that would have a lot longer delay than the mean delay value. Even though this is the case by introducing signal processing this delay would by relatively small in comparison. The delay is also comparable to the delay in Tvilling Digital that was hosted here at NTNU.
Bye developing a system in Azure the digital twin platform is gaining functionality in the form of bi-directional communication. By introducing the Raspberry Pi the system is greatly generalized and more user friendly. The system is also easier to scale and maintain. The main target group of this system is perhaps the students at NTNU and cost and fast deployment is important factors if this system is to be used by the students.

6.5 Challenges

The study is affected by the time constraint the first half of the semester was spent working on the Blueprint system before it was decided that a new platform would by developed from scratch. This greatly reduced the time spent on development and the current prototype. This has resulted in no integration of the Blueprint system into the Azure platform and therefore no FEM simulations is currently possible. The time spent on trying to further develop the Blueprint system have become important information as the systems might be merged but to much time was allocated to this.

6.6 Future Work

As this report illustrates there is still work to be done for the system to be implemented in Azure. The data visualization is specific for the current solution and only allows the user to view one sensor with the delay. The user can change between different DAB but not between the different sensors form the DAB. There should also be implemented some 3D visualisation of the physical asset this can be achieved by implementing Ceetron for instance. FEM simulations also needs to be integrated into the system. The python backend is ready for integration with the Blueprint system by integrating the Blueprint system a significant amount of time can be saved in the development. There is however some uncertainty as to the scope of the integration.

Simulated devices have been used for development up to this point and it is recommended to continue using these during development. The flexibility and easy access makes them ideal for generalising the solution. This being said a physical asset and a Raspberry Pi should be deployed in the future to gain real world experience.

In order to reduce the amount of events sent to the IoT hub some signal processing should be implemented in the Raspberry Pi. The script in the Raspberry Pi should be made specifically for each physical asset in this way different types of signal processing and number of sensors can be deployed for each digital twin. This leaves the user with some coding to be done before deployment but this is deemed a good compromise. Because the Raspberry Pi will be customized for every asset a message format should therefore be developed so the backend of the azure system can process a event form a general digital twin.

Chapter 7

Conclution

Definitions for a digital twin have been researched and a meaning of the term have been derived in the context of this project. Requirements that fulfill the derived meaning of digital twin have been formulated and a prototype have been developed. By utilizing Azure as a cloud provider, the digital twin platform gains several advantages. The Azure API will help to generalize the platform by introducing the Raspberry Pi as an easy to use and easy to modify data acquisition board. Raspberry Pi can also be used as a micro controller and thereby allowing meaningful bi-directional communication. Azure cloud platform is flexible and the possibility of running the Blueprint system in azure may speed up further development. The introduction of Raspberry Pi and the Azure API also helps to reduce technical debt in the Tvilling Digital system. Specifically by removing the part of the old system that is responsible for communication between the software Catman and the digital twin platform.

Bibliography

Greengard, Samuel (June 2019). "Digital Twins Grow Up". In: *ACM news*, pp. 1016–1022.

Grieves, Michael and John Vickers (Aug. 2017). "Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems". In: pp. 85–113. ISBN: 978-3-319-38754-3. DOI: 10.1007/978-3-319-38756-7_4.

https://github.com/Azure/azure-iot-arduino/graphs/contributors (n.d.). *Azure IoT Currently Does Not Support Arduino*. https://github.com/Azure/azure-iot-arduino.

https://github.com/Azure/iot/graphs/contributors (n.d.). *Awesome Azure IoT*. https://github.com/Azure/iot.

Kritzinger, Werner et al. (Jan. 2018). "Digital Twin in manufacturing: A categorical literature review and classification". In: *IFAC-PapersOnLine* 51, pp. 1016–1022. DOI: 10.1016/j.ifacol.2018.08.474.

Marr, Bernard (2019). *7 Amazing Examples Of Digital Twin Technology In Practice*.

microsoft, Development team at (Dec. 2018a). *Azure Event Hubs — A big data streaming platform and event ingestion service*. https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-about.

— (Dec. 2018b). *What is Azure IoT Hub?* https://docs.microsoft.com/en-us/azure/iot-hub/about-iot-hub.

Norderud Jensen, simen (June 2019). "Building an extensiable prototype for a cloud based digital twin platform". In:

Thilmany, Jean (Sept. 2017). "Identical Twins". In: *The American Society of Mechanical Engineers, ASME*.

Trancossi M. Cannistraro M., Pascoa J. (June 2018). "Can constructal law and exergy analysis produce a robust design method that couples with industry 4.0 paradigms? The case of a container house". In: *International Information and Engineering technology Association, IIETA* 5, pp. 303–312.

What is cloud computing? (N.d.). https://aws.amazon.com/what-is-cloud-computing/. Accessed: 2019.

Appendix A

Quickstart

A.1 Prerequisites

Python 3.7
Git
NodeJS

Nodejs can be installed from this link:
https://nodejs.org/en/

A.2 Importing project

extract all files from from the zip file to a file location of your choice.

A.3 Packages

A.3.1 Pyhton

open a new terminal window and copy following comand:
pip install azure-iot-device

A.3.2 NodeJS

If You dont add Nodejs to your "path" in environment variables open the NodeJS command promt. Copy this commands in the window:

- npm install @azure/event-hubs
- npm install azure-storage
- npm install eslint
- npm install express
- npm install ws

A.4 Project configurations

in the project open server.js navigate to line 32. Here you replace "YOUR INTERPRETERS NAME" with the name for your interpreter.

```
25  wss.broadcast = (data) => {
26    wss.clients.forEach((client) => {
27      if (client.readyState === WebSocket.OPEN) {
28        try {
29          //console.log(`Broadcasting data ${data}`);
30          var spawn = require('child_process').spawn;
31          //C:\Users\espen\Documents\fag-NTNU\prosjekt\digitalTwinAzure\digitalTwin\pythonScripts\FEM.py
32          var child = spawn('YOUR INTERPRETERS NAME', ['./pythonScripts/FEM.py', data]);
33
34          child.stdout.on('data', function (data) {
35            var parsedData = JSON.parse(data);
36            console.log("DELAY [ms]: " + JSON.stringify(parsedData["IotData"]["humidity"]));
37            client.send(JSON.stringify(parsedData));
38          });
```

The project are now ready to run. In your node promp navigate to the project folder and copy this command:
Node server.js

```
C:\Users\espen> cd C:\Users\espen\Documents\fag-NTNU\prosjekt\digitalTwinAzure\digitalTwin

C:\Users\espen\Documents\fag-NTNU\prosjekt\digitalTwinAzure\digitalTwin> Node server.js
Listening on 3000.
Successfully created the EventHub Client from IoT Hub connection string.
The partition ids are:  [ '0', '1' ]
```

The web at is now running. Open a web browser of your choice in the address field write: localhost:3000

Appendix C

Raspberry Pi Code

This appendix contains the code that is used to read sensor data from MinIMU-9 v5 and send the data to the platform.

C.1 IMUv5m4.py

Code listing C.1: IMUv5m4.py: The main for running the sensor reading.

```
# socket configuration
from math import degrees

UDP_IP = "1.2.3.4"
UDP_PORT = 27015

# number of connected sensors
N = 1

# ------------------------------------------------------------------------------
# GENERAL definitions
# ------------------------------------------------------------------------------

# I2C bus and register
from smbus import SMBus

BUS = SMBus(1)
LSM = 0x6b  # 3-axis gyroscope and 3-axis accelerometer
LIS = 0x1e  # 3-axis magnetometer

# socket configuration and check for connection
import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
socket_check = 1
while (socket_check > 0):
    socket_check = sock.connect_ex((UDP_IP, UDP_PORT))
```

```
    time.sleep(1)

# ------------------------------------------------------------------------------
# functions
# ------------------------------------------------------------------------------

# from numpy import pi
pi = 3.14


def socket_send(INCdata):
    datastring = ''
    for i in range(N):
        for k in range(2):
            datastring = datastring + '%.8f' % (pi / 180 * INCdata[i][k]) + ', '
    sock.sendto(datastring, (UDP_IP, UDP_PORT))


# ------------------------------------------------------------------------------
# sensor connection
# ------------------------------------------------------------------------------

# import functions
from func_sensor_register import SENregister
from func_sensor_check import SENcheck
from func_sensor_config import SENconfig

# definitions
# Sout = [[False, False], [True, False], [False, True], [True, True]]
Sind = 0

# set register definitions
(LSMconf_gyro, LSMconf_acc, LISconf) = SENregister(BUS, LSM, LIS)

# go through all sensors
print('')

# check if sensor is available
Sind = SENcheck(BUS, LSM, LIS)

# perform sensor configuration
if (Sind == 2):
    SENconfig(BUS, LSM, LIS)
    print('IMU online')
else:
    print('IMU offline')

# ------------------------------------------------------------------------------
# sensor measurements
# ------------------------------------------------------------------------------

# import functions
from func_sensor_read import XYZ_data_1
from func_sensor_read import XYZ_data_n

# definitions
# from numpy import zeros
INCini = [[0., 0., 0., 0.]]  # zeros([N,4])
INCdata = [[0., 0., 0., 0.]]  # zeros([N,4])
SENdir = ['X', 'Y']
```

```
# initial measurement step

# read and store sensor data
INCdata = XYZ_data_1(BUS, LSMconf_gyro, LSMconf_acc)
INCini = INCdata

# kafka connection
import digitalTwinPlt

dthub = digitalTwinPlt.Connect(device_id='tingenSAP',
                               topic='demo',
                               receiving_partition=1)

start_time = tid = time.time()


# measurement step 2-n
def pause(is_paused=True):
    while is_paused:
        inst = dthub.receive()
        for _topic, _partition_msg in inst.items():
            for _m in _partition_msg:
                if _m.value == b'start':
                    is_paused = False


try:
    while True:
        # read and update sensor
        radians, force = XYZ_data_n(BUS, LSMconf_gyro, LSMconf_acc, INCdata)

        if time.time() - tid > 0.01:
            print("rotation:␣" + str(round(radians, 4)) + "␣|||||␣" +
                  "degrees:␣" + str(round(degrees(radians), 4)) + "␣|||||␣" +
                  "force:␣" + str(round(force, 4)) + "␣|||||␣" +
                  "time␣diff:␣" + str(time.time() - tid))
            tid = time.time()

            dthub.add_to_message(name='Cantilever_Angle_radians', value=radians)
            dthub.add_to_message(name='Input_F_in', value=force)
            dthub.send()
            instruction = dthub.receive()
            for topic, partition_msg in instruction.items():
                for m in partition_msg:
                    if m.value == b'pause':
                        pause()

            print("successfully␣sent␣to:␣" + str(dthub.topic))

        start_time = time.time()

        time.sleep(0)
# send data over sockect connection
#               socket_send(INCdata)

except:
    sock.close()
    exit()
```

C.2 funk_sensor_config.py

Code listing C.2: funk_sensor_config.py

```
# ------------------------------------------------------------------------------
# FUNCTION, sensor configuration
#
# danielz, 11/2015
# ------------------------------------------------------------------------------

# function definition


def SENconfig(BUS, LSM, LIS):
        # LSM6DS33 datasheet, page 46-54

        # 104 Hz (high performance), 2g accelerometer full-scale selection,
        # 50Hz anti-aliasing filter bandwidth selection
        BUS.write_byte_data(LSM, 0x10, 0b01000011)

        # 104 Hz (high performance),
        # 245 dps (degrees per second), full scale disabled
        BUS.write_byte_data(LSM, 0x11, 0b01000000)

        # high pass filter gyroscope enabled
        BUS.write_byte_data(LSM, 0x16, 0b01000000)

        # low pass filter accelerometer enabled
        BUS.write_byte_data(LSM, 0x19, 0b00111100)
```

C.3 funk_sensor_check.py

Code listing C.3: funk_sensor_check.py

```

# ------------------------------------------------------------------------------
# FUNCTION, sensor check
#
# danielz, 11/2015
# ------------------------------------------------------------------------------




# whoami identification
LSM_WHOAMI = 0b01101001
LIS_WHOAMI = 0b00111101


# function definition
def SENcheck(BUS, LSM, LIS):

        # definitions
        Sind = 0
```

```
        # LSM, 3-axis gyroscope and 3-axis accelerometer
        try:
                if (BUS.read_byte_data(LSM, 0x0f) == LSM_WHOAMI):
                        Sind = Sind + 1
        except IOError as e:
                        Sind = Sind

        # LIS, 3-axis magnetometer
        try:
                if (BUS.read_byte_data(LIS, 0x0f) == LIS_WHOAMI):
                        Sind = Sind + 1
        except IOError as e:
                        Sind = Sind

        # return value
        return (Sind)
```

C.4 funk_sensor_read.py

Code listing C.4: funk_sensor_read.py

```
# ------------------------------------------------------------------------------
# FUNCTION, sensor read
#
# ------------------------------------------------------------------------------


# ------------------------------------------------------------------------------
# definitions
# ------------------------------------------------------------------------------

pi = 3.14
raddeg = pi / 180

# filter parameters
K = 0.90
K1 = 1 - K
time_diff = 0.005

# Cantilever dimentions

CL = 0.300  # [m]
CH = 0.025  # [m]
CB = 0.001  # [m]
CE = 210000000000  # (N/m  ) Cantilever Elastisity module
CA = CH * CB
CIx = CB * pow(CH, 3) / 12
CIy = CH * pow(CB, 3) / 12

beam = raddeg * 2 * CE * CIy / pow(CL, 2)

# degrees of freedom
DOF = 2

# import modules
from math import sqrt
```

```
from math import atan2
from math import degrees


# ------------------------------------------------------------------------------
# functions
# ------------------------------------------------------------------------------

def twos_comp_combine(msb, lsb):
    twos_comp = 256 * msb + lsb
    if (twos_comp >= 32768):
        return (twos_comp - 65536)
    else:
        return (twos_comp)


def XYZ_sensor_data(BUS, LSMconf_gyro, LSMconf_acc):
    # Gyroscope
    LSMdata_gyro = [0] * DOF
    for i in range(DOF):
        LSMdata_gyro[i] = twos_comp_combine(
            BUS.read_byte_data(LSMconf_gyro[0], LSMconf_gyro[2 * i + 2]),
            BUS.read_byte_data(
                LSMconf_gyro[0], LSMconf_gyro[2 * i + 1])) \
                          * LSMconf_gyro[7]
    # Accelerometer
    LSMdata_acc = [0] * 3
    for i in range(3):
        LSMdata_acc[i] = twos_comp_combine(
            BUS.read_byte_data(LSMconf_acc[0], LSMconf_acc[2 * i + 2]),
            BUS.read_byte_data(
                LSMconf_acc[0], LSMconf_acc[2 * i + 1])) \
                         * LSMconf_acc[7]

    # return

    return (LSMdata_gyro, LSMdata_acc)


def XYZ_data_1(BUS, LSMconf_gyro, LSMconf_acc):
    # sensor reading
    (LSMdata_gyro, LSMdata_acc) = XYZ_sensor_data(BUS,
                                                  LSMconf_gyro,
                                                  LSMconf_acc)
    # initial values
    INC = [0] * 2 * DOF
    for i in range(DOF):
        INC[i] = get_rotation(LSMdata_acc, i)
        INC[i + DOF] = LSMdata_gyro[i]
    # print(INC)

    # return
    return (INC)


def XYZ_data_n(BUS, LSMconf_gyro, LSMconf_acc, INC):
    # sensor reading
    (LSMdata_gyro, LSMdata_acc) = XYZ_sensor_data(BUS,
                                                  LSMconf_gyro,
                                                  LSMconf_acc)
```

```
    # complementary filter
    for i in range(DOF):
        LSMdata_gyro[i] -= INC[i + DOF]
        gyro_delta = (LSMdata_gyro[i] * time_diff)
        rotation = get_rotation(LSMdata_acc, i)
        INC[i] = K * (INC[i] + gyro_delta) + (K1 * (rotation))

    force = (90 - INC[1]) * beam
    # return
    return ((90 - INC[1]) * raddeg), force


def dist(a, b):
    return (sqrt(pow(a, 2) + pow(b, 2)))


def get_rotation(LSMdata_acc, i):
    if (i == 0):
        radians = atan2(LSMdata_acc[1], dist(LSMdata_acc[0], LSMdata_acc[2]))
        return (-degrees(radians))
    elif (i == 1):
        radians = atan2(LSMdata_acc[0], dist(LSMdata_acc[1], LSMdata_acc[2]))
        return (degrees(radians))
    elif (i == 2):
        radians = atan2(LSMdata_acc[2], dist(LSMdata_acc[0], LSMdata_acc[1]))
        return (degrees(radians))
```

C.5 funk_sensor_register.py

Code listing C.5: funk_sensor_register.py

```
# ------------------------------------------------------------------------------
# FUNCTION, sensor registers
#
# ------------------------------------------------------------------------------


# import modules
from numpy import pi


# function definition
def SENregister(BUS, LSM, LIS):
    # LSM6DS33 datasheet, page 37-38
    # ------------------------------------------------------------------------------

    # Gyroscope output register
    LSMconf_gyro = []
    LSMconf_gyro.append(LSM)

    # OUTX_L_G - Angular rate sensor pitch axis (X) angular rate output register (r)
    LSMconf_gyro.append(0x22)

    # OUTX_H_G - Angular rate sensor pitch axis (X) angular rate output register (r)
    LSMconf_gyro.append(0x23)
```

```
    # OUTY_L_G - Angular rate sensor roll axis (Y) angular rate output register (r)
    LSMconf_gyro.append(0x24)

    # OUTY_H_G - Angular rate sensor roll axis (Y) angular rate output register (r)
    LSMconf_gyro.append(0x25)

    # OUTZ_L_G - Angular rate sensor yaw axis (Z) angular rate output register (r)
    LSMconf_gyro.append(0x26)

    # OUTZ_H_G - Angular rate sensor yaw axis (Z) angular rate output register (r)
    LSMconf_gyro.append(0x27)

    # scaled angular rate sensitivity for 245 dps
    LSMconf_gyro.append(0.00875 * pi / 180)

    # Accelerometer output register
    LSMconf_acc = []
    LSMconf_acc.append(LSM)

    # OUTX_L_XL - Linear acceleration sensor X-axis output register (r)
    LSMconf_acc.append(0x28)

    # OUTX_H_XL - Linear acceleration sensor X-axis output register (r)
    LSMconf_acc.append(0x29)

    # OUTY_L_XL - Linear acceleration sensor Y-axis output register (r)
    LSMconf_acc.append(0x2A)

    # OUTY_H_XL - Linear acceleration sensor Y-axis output register (r)
    LSMconf_acc.append(0x2B)

    # OUTY_L_XL - Linear acceleration sensor Z-axis output register (r)
    LSMconf_acc.append(0x2C)

    # OUTY_H_XL - Linear acceleration sensor Z-axis output register (r)
    LSMconf_acc.append(0x2D)

    # scaled linear acceleration sensitivity for 2g full-scale
    LSMconf_acc.append(0.000061 * 9.81)

    # LIS3MDL datasheet, page 23
    # ------------------------------------------------------------------------
    # Magnetometer output registers
    LISconf = []
    LISconf.append(LIS)
    LISconf.append(0x28)  # OUT_X_L - x-axis data output
    LISconf.append(0x29)  # OUT_X_H - x-axis data output
    LISconf.append(0x2A)  # OUT_Y_L - x-axis data output
    LISconf.append(0x2B)  # OUT_Y_H - x-axis data output
    LISconf.append(0x2C)  # OUT_Z_L - x-axis data output
    LISconf.append(0x2D)  # OUT_Z_H - x-axis data output

    # return values
    # ------------------------------------------------------------------------
    return (LSMconf_gyro, LSMconf_acc, LISconf)
```

Appendix D

Bluerig

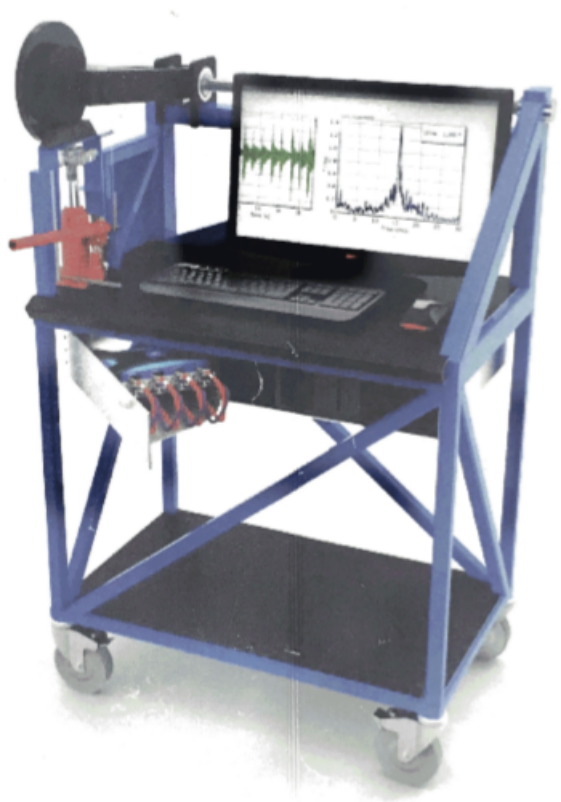

Figure D.1: Illustration of how the "Bluerig" or "Testrig" asset is constructed.

Appendix E

Digital Twin Platform Code Documentation

Digital Twin Platform Documentation

Release 1.0.0

Espen M. Sandtveit

July 05, 2020

Table of Contents

class main.**Settings** (*settings_module*)
Class for application settings from settings file

main.**main** (*args*)
Starts the application. Sets up settings and calls run app. :param args: Command line args if the script is run as script :return: Application

app.**check_for_existing_topics** (*app*)
Checks kafka for existing topics and adds them to the app. :return: nothing

app.**cleanup_background_tasks** (*app*)
A method to be called on shutdown, closes the WebSocket and Kafka connections

app.**init_app** (*settings*) → aiohttp.web_app.Application
Initializes and starts the server

app.**start_background_tasks** (*app*)
A method to be called on startup, initiates the Kafka consumer loop.

views.**bidirectional_communication** (*request: aiohttp.web_request.Request*)
Sends single command to topic that is defined in the request. :return: status OK if message is sent.

views.**index** (*request: aiohttp.web_request.Request*)
The API index
A standard HTTP request will return a sample page with a simple example of api use. A WebSocket request will initiate a websocket connection making it possible to retrieve measurement and simulation data.
Available endpoints are - /client for information about the clients websocket connections - /datasources/ for measurement data sources - /processors/ for running processors on the data - /blueprints/ for the blueprints used to create processors - /fmus/ for available FMUs (for the fmu blueprint) - /models/ for available models (for the fedem blueprint) - /topics/ for all available data sources (datasources and processors)

views.**models** (*request: aiohttp.web_request.Request*)
List available models for the fedem blueprint

views.**session_endpoint** (*request: aiohttp.web_request.Request*)
Only returns a session cookie
Generates and returns a session cookie.

views.**subscribe** (*request: aiohttp.web_request.Request*)
Subscribe to the given topic

views.**topics** (*request: aiohttp.web_request.Request*)
Lists the available data sources for plotting or processors
Append the id of a topic to get details about only that topic Append the id of a topic and /subscribe to subscribe to a topic Append the id of a topic and /unsubscribe to unsubscribe to a topic Append the id of a topic and /history to get historic data from a topic

views.**topics_create** (*request: aiohttp.web_request.Request*)
Creates new Kafka topic. :return:

views.**topics_delete** (*request: aiohttp.web_request.Request*)
Deletes topic. :return: Status

views.**topics_detail** (*request: aiohttp.web_request.Request*)
Show a single topic
Append /subscribe to subscribe to the topic Append /unsubscribe to unsubscribe to the topic Append /history to get historic data from a topic

views.**unsubscribe** (*request: aiohttp.web_request.Request*)
Unsubscribe to the given topic

class utils.**RouteTableDefDocs**
A custom RouteTableDef that also creates /docs pages with the docstring of the functions.

static get_docs_response (*handler*)
Creates a new function that returns the docs of the given function

route (*method: str*, *path: str*, ***kwargs*) → Callable[Union[typing.Type[aiohttp.abc.AbstractView], typing.Callable[[NoneType], typing.Awaitable[NoneType]]], Union[typing.Type[aiohttp.abc.AbstractView], typing.Callable[[NoneType], typing.Awaitable[NoneType]]]]
Adds the given function to routes, then attempts to add the docstring of the function to /docs

utils.**add_partition_for_simulation** (*app*, *topic*)
Adds one new partition to the topic

utils.**find_in_dir** (*filename*, *parent_directory=''*)
Checks if the given file is present in the given directory and returns the file if found. Raises a HTTPNotFound exception otherwise

utils.**getClient** (*request: aiohttp.web_request.Request*)
Returns the client object belonging to the owner of the request.

utils.**make_serializable** (*o*)
Makes the given object JSON serializable by turning it into a structure of dicts and strings.

utils.**try_get_all** (*post*, *key*, *parser=None*)
Attempt to get all values with the given key from the given post request. Attempts to parse the values using the parser if a parser is given. Raises a HTTPException if the key is not found or the parsing fails.

class client.**Client**
Handles connections to a clients websocket connections

close ( )
Will close all the clients websocket connections

dict_repr ( ) → dict
Returns a the number of connections the client has

receive (*topic*, *messages*)
Asynchronously transmit data to the clients websocket connections
Will add the data to the buffer and send it when the buffer becomes large enough

Parameters
- **topic** – the topic the data received from
- **bytes** – the data received as bytes

kafkaconsumer.**consume_kafka** (*app: aiohttp.web_app.Application*)
The function responsible for delivering data to the connected clients.

class processors.processor.**Processor** (*processor_id: str*, *blueprint_id: str*, *blueprint_path: str*, *init_params: dict*, *topic: str*, *source_topic: str*, *source_format: str*, *min_input_spacing: float*, *min_step_spacing: float*, *min_output_spacing: float*, *processor_root_dir: str*, *kafka_server: str*)
The main process endpoint for processor processes

retrieve_status ( )
Retrieves the status of the processor process
Can only be called after initialization. Should be run in a separate thread to prevent the connection from blocking the main thread :return: the processors status as a dict

set_inputs (*input_refs*, *measurement_refs*, *measurement_proportions*)
Sets the input values, must not be called before start

Parameters **output_refs** – the indices of the inputs that will be used

set_outputs (*output_refs*)
Sets the output values, must not be called before start

Parameters
- **input_refs** – the indices of the inputs that will be used
- **measurement_refs** – the indices of the input data values that will be used. Must be in the same order as input_ref.
- **measurement_proportions** – list of scales to be used on values before inputting them. Must be in the same order as input_ref.

start (*input_refs*, *measurement_refs*, *measurement_proportions*, *output_refs*, *start_params*)
Starts the process, must not be called before init_results

Parameters
- **input_refs** – the indices of the inputs that will be used
- **measurement_refs** – the indices of the input data values that will be used. Must be in the same order as input_ref.
- **measurement_proportions** – list of scales to be used on values before inputting them. Must be in the same order as input_ref.
- **output_refs** – the indices of the inputs that will be used
- **start_params** – the processors start parameters as a dict

Returns the processors status as a dict

stop ( )
Attempts to stop the process nicely, killing it otherwise

class processors.processor.**Variable** (*valueReference: int*, *name: str*)
A simple container class for variable attributes

processors.processor.**processor_process** (*connection: multiprocessing.connection.Connection*, *blueprint_path: str*, *init_params: dict*, *processor_dir: str*, *topic: str*, *source_topic: str*, *source_format: str*, *kafka_server: str*, *min_input_spacing: float*, *min_step_spacing: float*, *min_output_spacing: float*)
Runs the given blueprint as a processor
Is meant to be run in a separate process

Parameters
- **connection** – a connection object to communicate with the main process
- **blueprint_path** – the path to the blueprint folder
- **init_params** – the initialization parameters to the processor as a dictionary
- **processor_dir** – the directory the created process will run in
- **topic** – the topic the process will send results to
- **source_topic** – the topic the process will receive data from
- **source_format** – the byte format of the data the process will receive
- **kafka_server** – the address of the kafka bootstrap server the process will use
- **min_input_spacing** – the minimum time between each input to the processor
- **min_step_spacing** – the minimum time between each step function call on the processor
- **min_output_spacing** – the minimum time between each results retrieval from the processor

Returns

processors.views.**processor_create** (*request: aiohttp.web_request.Request*)
Create a new processor from post request.
Post params:
- id:* id of new processor instance max 20 chars, first char must be alphabetic or underscore, other chars must be alphabetic, digit or underscore
- blueprint:* id of blueprint to be used max 20 chars, first char must be alphabetic or underscore, other chars must be alphabetic, digit or underscore
- init_params: the processor specific initialization variables as a json string
- topic:* topic to use as input to processor
- min_output_interval: the shortest time allowed between each output from processor in seconds

processors.views.**processor_list** (*request: aiohttp.web_request.Request*)
List all created processors :return: List of processor names

processors.views.**processor_start** (*request: aiohttp.web_request.Request*)
Start a processor from post request.
Post params:

- id:* id of processor instance max 20 chars, first char must be alphabetic or underscore, other chars must be alphabetic, digit or underscore
- start_params: the processor specific start parameters as a json string
- input_ref: list of reference values to the inputs to be used
- output_ref: list of reference values to the outputs to be used
- measurement_ref: list of reference values to the measurement inputs to be used for the inputs. Must be in the same order as input_ref.
- measurement_proportion: list of scales to be used on measurement values before inputting them. Must be in the same order as input_ref.

processors.views.**processor_stop** (*request: aiohttp.web_request.Request*)
Stop the processor with the given id.

processors.views.**retrieve_processor_status** (*app*, *processor_instance*)
Retrieve the initialization results from a processor
Will put the results in app['topics'] and return them.

class digitalTwinPltAPI.digitalTwinPlt.**Connect** (*device_id*, *topic*)
Class that holds the connection to the Kafka server. This resides in the edge solution. Therefore configurations have too be done manually.

add_to_message (*name*, *value=None*)
Adds or updates attributes in the message.

receive ( )
Listens to the receiving topic for in coming messages

send (*data=None*)
sends the message to Kafka topic

digitalTwinPltAPI.blueRiggdeviceSim.**sensor_sampling** (*dtMessaging*)
Example method showing applications of bidirectional communication.

digitalTwinPltAPI.theThingDeviceSim.**pause** (*is_paused=True*)
Example method showing applications of bidirectional communication.

digitalTwinPltAPI.theThingDeviceSim.**sensor_data** (*dtMessaging*)
Loop running indefinitely generating simulated data. :param dtMessaging: This is the object holding the connection to Kafka

blueprints.views.**blueprint_detail** (*request: aiohttp.web_request.Request*)
Get detailed information for the blueprint with the given id

blueprints.views.**blueprint_list** (*request: aiohttp.web_request.Request*)
List all uploaded blueprints.
Append a blueprint id to get more information about a listed blueprint.

blueprints.views.**fmu_model_variables_and_valuereferences** (*request: aiohttp.web_request.Request*)
Get name and internal value reference of the fmu instance.

Returns Name and value reference of all variables in selected fmu

blueprints.views.**retrieve_method_info** (*class_body*, *method_name*, *params_ignore=1*) → Tuple[str, List]

Retrieves docs and parameters from the method

Parameters
- **class_body** – the body of the class the method belongs to
- **method_name** – the name of the method
- **params_ignore** – how many of the first params to ignore, defaults to 1 (only ignore self)

Returns a tuple containing both the docstring of the method and a list of parameters with name and default value

A blueprint for running FMUs.

class blueprints.fmu.__init__.**P** (*fmu='Cantilever.fmu'*)

The interface between the application and the FMU

start (*start_time*, *time_step_input_ref='-1'*)

Starts the FMU

Parameters
- **start_time** – not used in this blueprint
- **time_step_input_ref** – optional value for custom time_step input

blueprints.fmu.__init__.**prepare_outputs** (*output_refs*)

Create FMUPy compatible value references and outputs buffer from output_refs

Parameters **output_refs** – list of output indices

Returns tuple with outputs buffer and value reference list

- *Index*
- *Module Index*
- *Search Page*

R

S

T

U

V

Appendix F

Result Graph from Digital Twin Platform

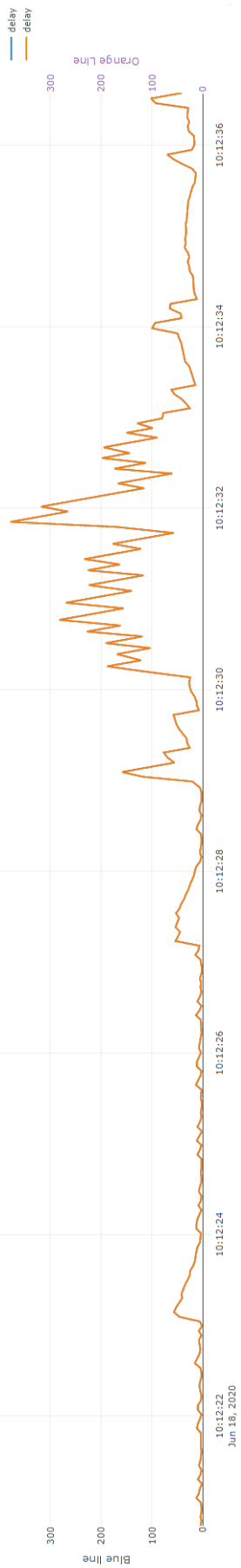

Figure F.1: Shows the delay for the last 300 data points.

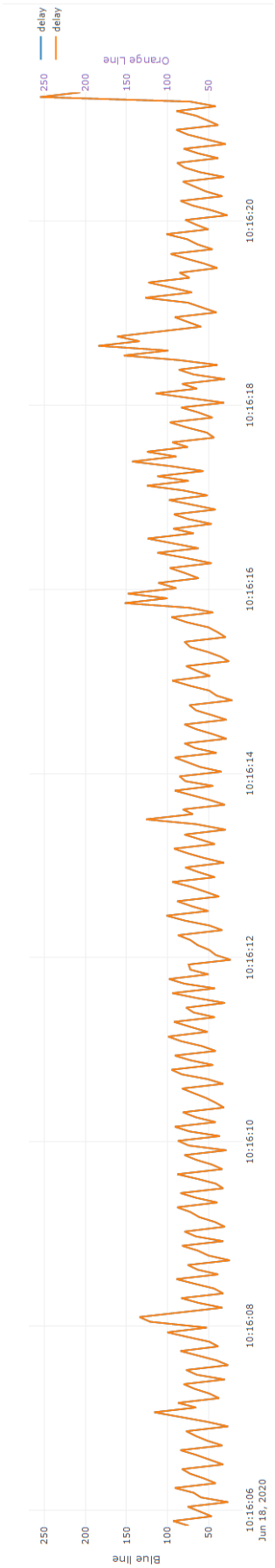

Figure F.2: Shows the delay for the last 300 simulated data points.

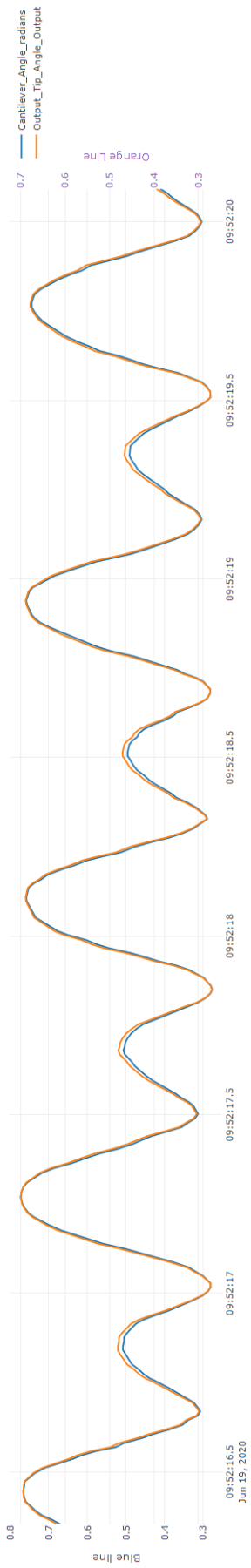

Figure F.3: Illustrates measured and calculated values for the physical response of "Tingen" with 100 samples a second and the K value at 0.9.

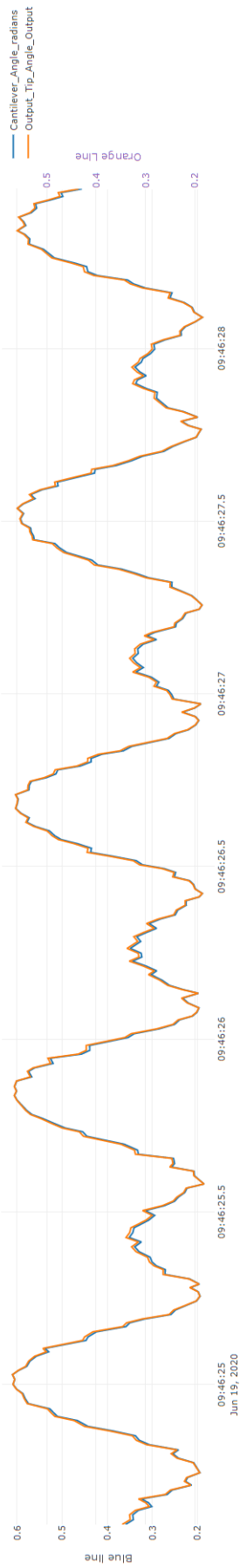

Figure F.4: Illustrates measured and calculated values for the physical response of "Tingen" with 100 samples a second and the K value at 0.1.

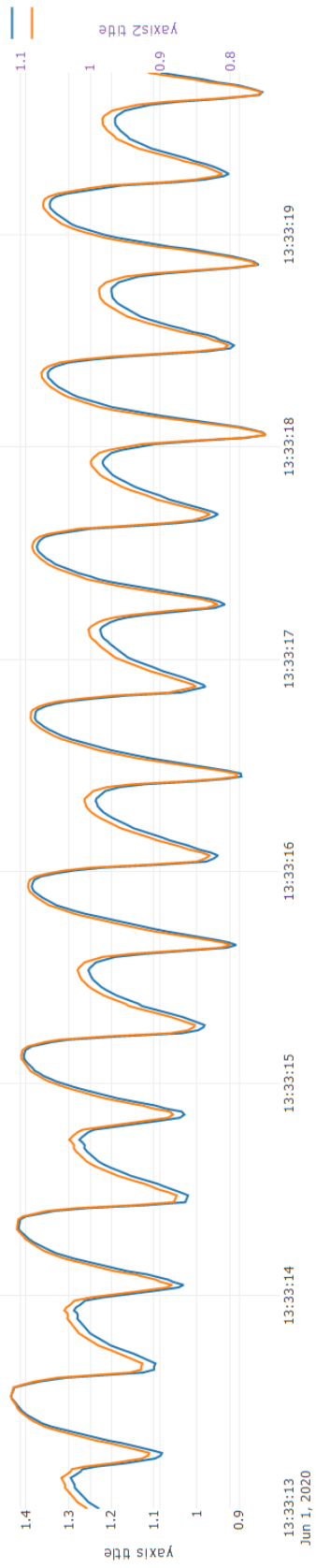

Figure F.5: Illustrates measured and calculated values for the physical response of "Tingen" with 20 samples a second.