

FMH606 Master's Thesis 2021

Electric Power Engineering

Digital Twin for Industrial Induction Heating Equipment

Ragnhild Eilertsen Moldesæther

Course: FMH606 Master's Thesis, 2021

Title: Digital Twin for Industrial Induction Heating Equipment

Number of pages: 100 (87 + Appendices)

Keywords: Digital twin, Induction heating, IoT, MQTT

Student: Ragnhild Moldesæther

Supervisor: Kjetil Svendsen, with Co-supervisor Nils-Olav Skeie

External partner: EFD Induction

Summary:

EFD Induction want to look at the possibility of developing a digital twin for their induction heating equipment. A digital twin has a variety of usages, such as monitoring, predictive maintenance, and testing what-if scenarios. This thesis will study how a digital twin can be developed for the benefit of the customers of EFD Induction regarding testing of their induction heating application and compare with existing testing methods.

A digital twin has been developed, consisting of an Application Model, Converter Model, Set Point Controller, Data Storage and Converter Interface. The focus has been on a connection between the programs through a central hub, called Data Exchange, with the implementation of an IoT messaging protocol.

A connection between the models has been established, and a base for the digital twin for induction heating equipment has been developed. Testing of the real application and the digital twin have been executed to compare the temperature development in the application. The usage for the digital twin for customers at this stage, is monitoring the induction heating process when testing. For future work, a Model Parameter Tuner can be included as an additional building block in the digital twin for optimizing the parameters in the Application Model.

Preface

This thesis was written in spring 2021 to complete the master's degree in Electrical Power Engineering at USN (University in South-Eastern Norway). The thesis has given me the opportunity to explore the digital twin technology and implement it into a real process. I have been fortunate to receive new knowledge in technology that has advanced in the recent years and are continuing to expand to include even more usages. I want to thank Dmitry Ivanov at EFD Induction for sharing the application model he has developed, which has been a big importance of the digital twin. Co-supervisor Nils-Olav Skeie has assisted me with his knowledge in programming, especially in C#, and databases. And finally, my supervisor Kjetil Svendsen, for his commitment to this thesis and giving me support throughout the process.

They have given me valuable inputs and guidance, and I am thankful for their help and support during this experience.

Porsgrunn, 18.05.21

Ragnhild Moldesæther

Contents

Preface	3
Contents.....	4
Nomenclature	6
1 Introduction	13
1.1 Background	13
1.2 Task description	14
1.3 Objectives.....	14
1.4 Report Structure	14
2 Theory	15
2.1 Induction Heating Equipment.....	15
2.1.1 <i>Electromagnetic Induction</i>	15
2.1.2 <i>Eddy Currents</i>	16
2.1.3 <i>Skin Depth</i>	16
2.1.4 <i>Induction Heating Equipment</i>	17
2.1.5 <i>Testing and evaluating of induction heating equipment</i>	19
2.2 Heat transfer	19
2.2.1 <i>Conduction</i>	19
2.2.2 <i>Convection and Radiation</i>	20
2.3 Modelling	20
2.4 Digital Twin.....	21
3 Implementation of the Digital Twin.....	23
3.1 System Design	23
3.2 Data Exchange with MQTT.....	25
3.2.1 <i>Data Exchange with MQTT</i>	25
3.2.2 <i>Set up of MQTT Broker</i>	26
3.2.3 <i>Connection to broker</i>	26
3.2.4 <i>Publish to topic</i>	27
3.2.5 <i>Quality of Service (QoS)</i>	27
3.2.6 <i>Subscribe to topic</i>	28
3.3 Application Model.....	28
3.3.1 <i>Heat transfer through workpiece</i>	28
3.3.2 <i>Application Model with Python</i>	31
3.3.3 <i>Application Model with Python using FEniCS</i>	38
3.4 Converter Model.....	43
3.4.1 <i>Introduction to the Converter Model</i>	43
3.4.2 <i>Converter Model with Python</i>	45
3.5 Set Point Controller with C#	48
3.5.1 <i>Set Point Controller</i>	48
3.5.2 <i>Set Point Controller with Timer</i>	50
3.5.3 <i>Set Point Controller with Display</i>	52
3.6 Model Parameter Tuner	54
3.7 Data Storage.....	54
3.7.1 <i>Data Storage Model with Python</i>	55
3.8 Converter Interface with Raspberry Pi 3	57
3.8.1 <i>Complete Setup of the Raspberry Pi 3</i>	57
3.8.2 <i>Headless setup of the Raspberry Pi 3</i>	57

3.8.3 Install MQTT on Raspberry Pi 3.....57
3.8.4 Raspberry Pi 3 with Automation HAT58
3.8.5 Converter Interface for Digital Twin62

4 Testing of the Digital Twin of Induction Heating Equipment64

4.1 Introduction to the setup and experiments.....64
 4.2 Heating System.....65
 4.3 Workpieces and Heating coils.....67
 4.3.1 Workpieces.....67
 4.3.2 Heating coils.....68
 4.4 Real-time data with MODBUS.....69
 4.5 Preparations before experiment.....70
 4.6 Experiments of the real converter.....76
 4.6.1 Comments on the experiments.....77
 4.6.2 Results of the experiments.....78

5 Conclusion81

5.1 Conclusion81
 5.2 Future work82

References.....84

6 Appendices.....88

Nomenclature

Symbol	Explanation and units
AC	Alternating current [A]
AI	Artificial Intelligence
AR	Argumented Reality
CPS	Cyber-Physical-System
CSV	Comma-Separated Values
DC	Direct current [A]
emf	Electromotive Force
FEM	Finite Element Method
GUI	Graphical User Interface
GPIO	General Purpose Input/Output
IP	Internet Protocol
IoT	Internet-Of-Things
M	Workpiece of magnetic material
MQTT	Message Queuing Telemetry Transport
ML	Machine Learning
NM	Workpiece of non-magnetic material
PDE	Partial Differential Equation
PWM	Pulse Width Modulation
P_{dis}	Heat dissipated by Joule losses [W]
QoS	Quality of Service
SQL	Structured Query Language

TCP

Transmission Control Protocol

List of figures

Figure 2.1: Illustration of a heating coil with supply of AC current with induced eddy current in the workpiece.....	17
Figure 2.2: Illustration of a frequency converter	18
Figure 2.3: Illustration of a Digital Twin with different usages it covers.....	21
Figure 3.1: Building blocks of Digital Twin.....	23
Figure 3.2: Information flow between the models of the Digital Twin	24
Figure 3.3: Illustration of client/broker communication in MQTT	25
Figure 3.4: Content of a CONNECT message.....	26
Figure 3.5: Content of a CONNACK message.....	26
Figure 3.6: Content of a PUBLISH message	27
Figure 3.7: Content of a SUBSCRIBE message.....	28
Figure 3.8: Illustration of heat flow in workpiece	30
Figure 3.9: Communication between Set Point Controller and Application Model through Data Exchange	31
Figure 3.10: Flow Diagram of communication between MQTT and Application Model in the Python script	32
Figure 3.11: Classes in the Python script for the Application model	33
Figure 3.12: Calculation of heat flow, Joule losses and the change in internal energy under class Model	34
Figure 3.13: Declaring variables for MQTT Paho client.....	34
Figure 3.14: Defining function for connecting to MQTT broker	35
Figure 3.15: Function from Paho client library for subscribing in Python.....	35
Figure 3.16: Paho client library for running the MQTT functions in Python.....	35
Figure 3.17: Mosquitto client function 'mosquitto_pub' in the Windows command line for publish value of current	36
Figure 3.18: Message received in the subscribing topic in the Spyder environment	36
Figure 3.19: Function from Paho client library for publishing in Python	36
Figure 3.20: Mosquitto client function 'mosquitto_sub' for subscribing on topic 'temperatureT1' in Windows command line	37
Figure 3.21: Mosquitto client function 'mosquitto_sub' for subscribing on topic 'temperatureT2' in Windows command line	37

Figure 3.22: Information flow between Set Point Controller, Converter Model, and the Application Model38

Figure 3.23: Defined variables for the Paho client library.....39

Figure 3.24: Paho client functions for handling received messages39

Figure 3.25: Functions from the FEniCS library implemented in the Application Model39

Figure 3.26: Function under Class TransientSolver for solving numerical solution of the temperature40

Figure 3.27: Class Material for defining material properties.....40

Figure 3.28: the PDE is solved for each time step and published to topic by MQTT41

Figure 3.29: Result from simulation of the Application Model with the FEniCS library41

Figure 3.31: Illustration of the communication between the Converter and Application model43

Figure 3.32: Output circuit of the converter44

Figure 3.33: Definitions of the variables for Paho client library in the Converter Model.....45

Figure 3.34: Using functions from Paho Client library for connecting and subscribing to topics46

Figure 3.35: Function for calculations in the Converter Model.....47

Figure 3.36: GUI for inputs to the model48

Figure 3.37: M2MQTT client library in Visual Studios48

Figure 3.38: Functions in the M2MQTT library for MQTT connection in Visual Studios.....49

Figure 3.39: Functions in the M2MQTT library for publishing messages49

Figure 3.40: GUI of the Set Point Controller with timers implemented.....50

Figure 3.41: Function behind Power button in the GUI application.....50

Figure 3.42: Event handler when the first timer is enabled in the GUI application51

Figure 3.43: Event handler when the second timer is enabled in the GUI application.....51

Figure 3.44: Start of sequence from the Set Point Controller52

Figure 3.45: Set Point Controller including display of parameters52

Figure 3.46: Information flow to the Set Point Controller through the Data Exchange53

Figure 3.47: Function for handling received messages to the Set Point Controller53

Figure 3.48: If-statement for report handling to the Data Storage model.....54

Figure 3.49: Function for 'Quit' button clicked.....54

Figure 3.50: Snippet of messages that are handled through MQTT in the Data Storage Model55

Figure 3.51: Defined parameters in class DataStorage that is to be stored in the model.....56

Figure 3.52: Functions for writing to file under class DataStorage56

Figure 3.53: Parameters written to the CSV file opened in Excel57

Figure 3.54: MQTT communication between computer and Raspberry Pi 358

Figure 3.55: The Automation HAT board attached to the Raspberry Pie Model B+ board59

Figure 3.56: A wire connected between the analog input channel and 5 V channel on the Automation HAT board60

Figure 3.57: Python script created in Nano editor with libraries for MQTT and Automation HAT included.....61

Figure 3.58: Subscribing on topic 'test' in the Windows command line.....61

Figure 3.59: Python script in Raspberry Pi for controlling relay at power on signal62

Figure 3.60: Python script in Raspberry Pi for reading analog inputs from the converter63

Figure 4.1: Illustration of the connection between the hardware, including communication protocols MQTT and MODBUS64

Figure 4.2: Control panel of the Sinac SM 18/25 Twin.....66

Figure 4.3: Workpieces of magnetic (M) and non-magnetic (NM) material, and long and short coil to be used in the experiments.....67

Figure 4.4: Long coil to be used in the experiments.....69

Figure 4.5: Python script with MODBUS client library implemented70

Figure 4.6: Raspberry 3 Model B+ board for analog output71

Figure 4.7: Python library for the Raspberry Pi GPIO pins.....71

Figure 4.8: RC circuit with Op-Amp to filter the PWM signal from the Raspberry Pi.....72

Figure 4.9: Circuit diagram of how the PWM signal from the Raspberry Pi is filtered and converted, before sending analog set point current to the converter73

Figure 4.10: Wired connections between the analog inputs and digital outputs of the Automation HAT board and the real converter74

Figure 4.11: Full setup of the experiment.....75

Figure 4.12: Plot of temperature development at outer surface of magnetic (M) and non-magnetic (NM) workpieces during a 5-minute interval from power on signal78

Figure 4.13: Comparison between NM workpiece and the Application Model (DT)79

Figure 6.1: Start up MQTT Mosquitto Broker.....90

Figure 6.2: Allow anonymous set to false in the configuration file.....90

Figure 6.3: Enable password file and path in the configuration file.....90

Figure 6.4: Test of Mosquitto client function 'mosquitto_pub' in Windows command line ...91

Figure 6.5: Test of Mosquitto client function 'mosquitto_sub' in Windows command line....91

Figure 6.6: Ubuntu terminal as an interface to the Linux operating system.....92

Figure 6.7: Run Python script with FEniCS from Ubuntu terminal92

Figure 6.8: Raspberry Pie 3 Model B+ board93

Figure 6.9: Full set-up of the Raspberry Pi 394

Figure 6.10: Raspberry Pi Imager is used for writing the operating system over to a SD card
.....95

Figure 6.11: Desktop of the Raspbian operating system96

Figure 6.12: wpa_supplicant.conf file to define wireless network[48]97

Figure 6.13: Raspberry Pi Configuration from the command line98

Figure 6.14: Remote access to the Raspberry Pi using Windows SSH Client99

Figure 6.15: Configuration for DHCP 100

List of tables

Table 4.1: Equipment used in the experiments	65
Table 4.2: Technical data of Sinac 18/25 SM Twin	65
Table 4.3: Geometric data of workpieces	67
Table 4.4: The material properties of 42CrMo4[40] and Stainless Steel 316L[41]	68
Table 4.5: Experiments executed and result of maximum temperature and power output in each experiment	76

1 Introduction

Introduction to the master thesis is given in the first chapter. The background behind digital twin and the connection to this thesis is explained, with reference to the task descriptions. Objectives of the thesis and how it differs from previously implementations is defined. At last, an overview of the report structure is given.

1.1 Background

Our society is now transforming into the fourth industrial revolution, where the digital and real world are being combined in a cyber-physical system (CPS). With Internet-of-Things (IoT), physical devices can be connected through internet and share data. Artificial Intelligence (AI) and Machine Learning (ML) uses mathematical algorithms to make machines learn and develop themselves. These technologies open the possibilities for new inventions and creation of technologies that make production more effective. For companies to be relevant and competitive, they are in a need to digitalize their business. Digital Twin is a concept that has advanced in the recent years, especially due to IoT. This is a technology that creates a virtual representation of a physical product or process, by combining different building blocks from both the digital and real world.

Several sectors are taking advantage of implementing a digital twin in their process. The aerospace industry was the pioneers in implementing digital twin in their systems in the 1970s, even if the concept was not established at that time. NASA had simulators with matching conditions of a real-life spacecraft to train their astronauts for different scenarios, which prevented failure for Apollo 11 and 13 [1]. The manufacturing industry has been taking this technology most in use, particularly in the automobile industry such as Volvo and Maserati. Maserati made a digital twin of their vehicle for testing the aerodynamics effect on the vehicle in a virtual air tunnel. This helped Maserati in reducing cost in the testing and design phase and reducing the development time. Volvo wanted to increase their flexibility in the manufacturing site for their customers to offer solutions based on their choices, making quality control critical. A digital twin improved the overall efficiency and saved cost [2]. Health care is the sector that may benefit the most of this technology by predictive care and personalized medication. By increasing use of wearable devices amongst individuals, it is possible to collect data and have storage of historical information [3]. The health care industry sees a lot of potential in this technology for improving patient care and hospital planning and is discovering processes that can be optimized by implementing a digital twin. Siemens Healthineers have made a digital twin of the radiology operations in the Mater Private Hospital in Dublin, Ireland [4]. This improved the efficiency of the utilization and reduced the patient waiting time and staffing cost.

EFD induction want to look at the possibilities of creating a digital twin for their induction heating products. Predicting operations of their products in actual applications can be challenging. By developing better simulators, the testing and evaluating of their products and applications can be improved. The simulators need to interact with the real world by sensors sending information in real-time and compared with the physical product. This is a technology that has potential to make testing and evaluating of their products easier for their customers.

1.2 Task description

The signed task description of the master thesis is in Appendix A.

1.3 Objectives

The main objective of the master thesis is to develop a digital twin for the induction heating equipment at EFD Induction. A digital twin can consist of different building blocks and represent a variety of processes. This thesis will make an overview of the building blocks and investigate the usage of a potential digital twin at EFD Induction. Since the induction heating process depends on material of the workpiece and type of coil to be used in testing, which will make the modelling of the application complex, the focus of this thesis has been on the communication between the different models. When a connection between the models have been established, the models can be improved for a better fit for each application. This thesis has for that reason no strict objectives on developing models that are accurate for the real application, but rather a starting point for developing the models into a complete digital twin.

Compared to digital twin developed in other industries, the focus of developing this digital twin will be at the customer and how a digital twin can make testing and evaluating of the induction heating equipment beneficial and easier for them. The digital twin will be designed as a tool for the customer to use for monitoring and controlling the testing of their application. Instead of testing in lab, the customer can use their own computer to test and simulate their application with different scenarios. This will make the process more efficient and less time consuming, considering no travelling expenses and the testing can be executed at their own premises.

A literature survey of existing testing and evaluating methods of induction heating applications is to be executed for research. The existing testing method will be compared with the testing of a digital twin. In addition, a literature survey on how digital twins can be modelled and what tools to use is also to be executed.

1.4 Report Structure

Chapter 2 explains the theory of the induction heating process, modelling and digital twin technology.

Chapter 3 investigate the different building blocks of the digital twin. Each of the building blocks are explained in the subchapters with how they are developed.

Chapter 4 compare the testing of the real application against the digital twin.

Chapter 5 gives a conclusion for this thesis and propose future work of the digital twin.

2 Theory

In this chapter, the theory behind the induction heating equipment and digital twin is explained. Electromagnetic induction, eddy currents, skin effect and heat transfer are the physics behind the induction heating equipment which make this application induce heating in the workpiece without contact. Mathematical models will be developed for the purpose of having a digital twin to represent this process.

2.1 Induction Heating Equipment

2.1.1 Electromagnetic Induction

It has been known that current induce a magnetic field, but due to the research of scientists Joseph Henry and Michael Faraday it was shown that also the magnetic field can induce current. But the current was only induced when the magnetic field changed over time. Faradays law, which is the central principle of electromagnetic induction, states[5]:

“The induced emf ε in a closed loop equals the negative of the time rate of change of magnetic flux through the loop” and is given by the formula:

$$\varepsilon = - \frac{\Delta\Phi_B}{\Delta t} \quad (2.1)$$

Where:

- ε : Emf - electromotive force
- $\Delta\Phi_B$: Change in magnetic flux
- Δt : Change in time

Emf causes electrons to move and form a current and is required to make a current flow in a circuit. This formula let us calculate how much emf, and therefor how much current, that will be induced in a loop of wire by a change in magnetic flux. If the magnetic field is constant, Emf can also be induced by changing the area of the loop or changing the angle between the loop and the magnetic field.

Magnetic flux, Φ_B , is a measure of the magnetic field running through a loop of wire and most directly induces emf.

Magnetic flux through element of area $d\vec{A}$:

$$d\Phi_B = \vec{B} \cdot d\vec{A} = B_{\perp} \cdot d\vec{A} = B \cdot dA \cdot \cos \varphi \quad (2.2)$$

Where:

- \vec{B} : Magnetic field
- $d\vec{A}$: Element of area A
- $\cos \varphi$: Angle between \vec{B} and $d\vec{A}$

There are three factors that affect the magnetic field, and therefore the magnetic flux:

- The strength of the magnetic field, B
- The area of the loop, A
- The angle between the magnetic field and the coil, $\cos \varphi$

From the formula (2.2) it is shown that the maximum magnetic flux occurs when the coil is perpendicular to the magnetic field ($\varphi = 0^\circ$ and $\cos \varphi = 1$). And the minimum magnetic field occur when the coil is parallel to the magnetic field ($\varphi = 90^\circ$ and $\cos \varphi = 0$).

While Faradays law (2.1) tells us how much emf and current is induced, Lenz law gives an alternative method for determining the direction of the induced emf or current [5]:

“The direction of any magnetic induction effect is such to oppose the cause of the effect”.

This means that if the magnetic flux is decreasing through a loop of wire over time, the emf will increase accordingly. Depending on if the magnetic flux is decreasing or increasing, the current will go, respectively, in clockwise or counterclockwise direction according to the right-hand rule to oppose the change.

2.1.2 Eddy Currents

The current that is induced in the workpiece by the varying magnetic field is called eddy currents. It is called eddy currents due to the swirling pattern in the volume of the material. Eddy currents circulate through the workpiece and produces heat due to the resistance of the material [5]. The heat dissipated is called Joule losses and are given by the formula:

$$P_{dis} = I^2 \cdot R \quad (2.3)$$

2.1.3 Skin Depth

The skin effect causes the current density of the eddy currents to be larger at the surface of the workpiece. In the center of workpiece, the eddy currents get cancelled out because of the opposite directions of current flow. The depth of this larger distribution of eddy currents is called skin depth.

By the formula for skin depth, δ , it is shown that the skin depth is dependent on the frequency:

$$\delta = \sqrt{\frac{2 \cdot \rho}{\omega \cdot \mu}} = \sqrt{\frac{2 \cdot \rho}{2 \cdot \pi \cdot f \cdot \mu}} \quad (2.4)$$

Where:

- ρ is the resistivity of the material of the workpiece
- ω is the angular frequency, which is equal to $2\pi f$
- μ is the permeability of the material

As seen by the formula (2.3), when the frequency increases, the skin depth will become smaller, and the opposite with decreasing frequency. This formula is valid as long as the skin depth is smaller than the radius of the workpiece to be heated [6].

2.1.4 Induction Heating Equipment

The induction heating equipment is based in the theory of electromagnetic induction, eddy currents and skin depth resulting in Joule losses. With these effects it is possible to heat the workpiece with the purpose of changing the material properties, without needing to use open fire. Figure 2.1 illustrates the process of heating a workpiece by induction.

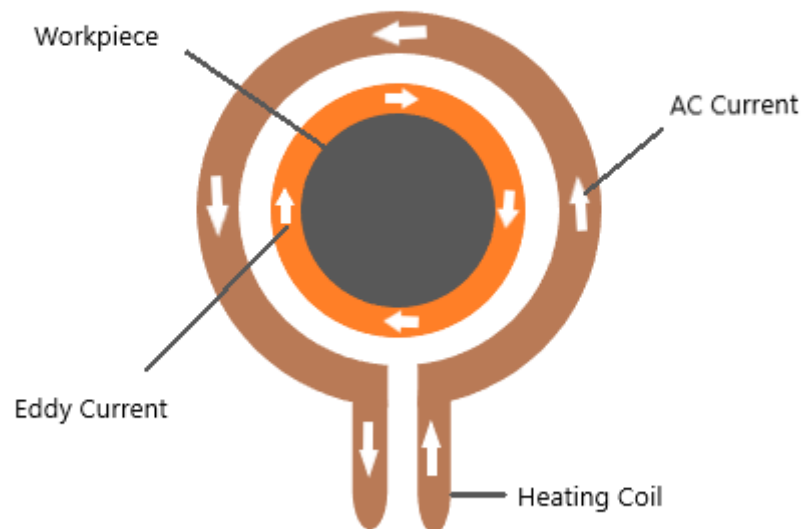


Figure 2.1: Illustration of a heating coil with supply of AC current with induced eddy current in the workpiece

An AC current (alternating current) will flow through the coil of the induction heating equipment, generating a magnetic field. When the workpiece is exposed to the changing magnetic field by the AC current in the coil, eddy currents are induced and start circulating in the workpiece. This produces Joule heating in the surface of the workpiece, with the skin depth depending on how high the frequency of the AC current is set to run in the coil. The heating is produced exactly where the workpiece is exposed to the magnetic field of the coil, allowing for a precise execution. The workpiece to be heated must be of a conductive material to allow the induced current to flow. The amount of heat dissipated depends on the resistivity of the material, with a higher resistance more heat is dissipated.

While the coil is part of the output circuit, the frequency converter is an important component of the induction heating equipment. Since the frequency is contributing to deciding how deep the eddy currents will flow in the workpiece and dissipate heat, a frequency converter is needed to adjust the frequency for the application.

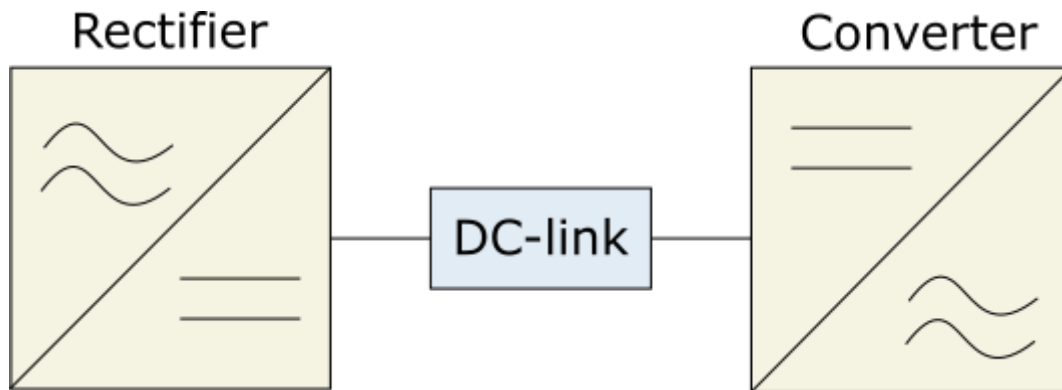


Figure 2.2: Illustration of a frequency converter

The basic components of a frequency converter are a rectifier, DC-link and a converter, shown in Figure 2.2. From the mains, an AC power is supplied to the equipment. To be able to adjust the frequency for the specific application, the voltage needs to be converted to DC voltage. The rectifier is a diode bridge which allows only one direction of the AC wave to let through. The result is a constant DC voltage. To filter the AC ripple, a DC link with a capacitor is between the rectifier and converter. The DC voltage is then converted back to AC by the rectifier. The rectifier consists of transistors that switch on and off and will replicate an AC current. Even if the output is not real AC, the switching on and off will act as an AC current and induce current in the workpiece. Before the current enter the coil, a step-up transformer converts the primary current to a higher secondary current by a ratio depending on the transformer's windings [7].

2.1.4.1 Applications of the induction heating equipment

The induction heating equipment has several advantages over other heating equipment. Instead of using open fire to heat, the induction equipment heats only the workpiece itself. This reduces the risk for the personnel for burns and intoxication from gas. The heating is centered where it is needed and will give an accurate and controllable application [6].

Induction heating equipment is used in many applications for heating conductive metals, such as hardening, brazing and welding. Hardening involves rapid heating and cooling of the material to increase the hardness and durability of steel, and is widely used by car manufactures [8]. Brazing joins to metal pieces together by melting a filler in between by induction heating [9]. Induction welding is used to heat the edges of pipes and tubes to then join them [10]. Induction heating also extends to tempering, bonding and other applications involving heating conductive metals.

2.1.5 Testing and evaluating of induction heating equipment

EFD induction have a range of standard induction heating equipment that can be used in a variety of processes, from heating of small components to large manufacturing processes. When a customer wants an induction heating equipment for their application, it normally involves testing in lab, depending on how customized the solution needs to be. In the lab, there is specialized personnel in power electronics, metallurgy, and power-control software, which together will design the most efficient solution.

Both the power source and heating coil can be customized for the specific solution. The heating coil can be designed to have any shape that is necessary for heating the workpiece. A typical heating coil will have a round shape and surrounds the workpiece, but the coil can be designed to any shape fitted for the solution. The power source depends on how high frequency and power the workpiece requires to achieve the optimal depth and structure from the heating. Computer simulation in advance removes trial and error-processes of the physical heating coil and reduce cost and time by not testing on real workpieces.

Other companies that supply induction heating equipment design their equipment in a similar way, by using computer simulation in advanced and testing in lab for process development such as Inductoheat [11] and Plustherm [12].

2.2 Heat transfer

The three mechanisms of heat transfer are conduction, convection, and radiation.

2.2.1 Conduction

When heat flow from a hot region to a colder region by direct contact, the heat transfer through the material is by conduction. There is no movement of mass, but the atoms in the hotter regions have more kinetic energy which is transferred to the atoms in the cooler region. The heat transfer is only when there is a temperature difference, and the direction is always from higher to lower temperature.

$$\frac{Q}{A} = -k \cdot \frac{dT}{dx} \quad (2.5)$$

Q [W] is the heat flow rate and Q/A [W/m^2] is the heat flux. The constant k [$W/m \cdot K$] is the thermal conductivity of the material and dT [K] is the temperature difference. The negative sign show that the heat flow is always in the direction of decreasing temperature. When the heat flow (Q) is negative, the heat is flowing out of the system. If the heat flow is positive, the heat is flowing into the system [13].

Conduction is the main heat transfer mechanism in the induction heating process since the heat is developed in the material of the workpiece and is transferred by direct contact to the colder regions in the bulk volume.

2.2.2 Convection and Radiation

Convection is the heat transfer by movement of mass from one region to another. The convection can either be forced or free. Forced convection is when the flow is caused by a blower or a pump. Free convection is a natural flow, like hot air rising upwards. Heat transfer caused by electromagnetic waves such as visible light, is called radiation [13].

Convection will be the main heat transfer mechanism when the heat has been distributed in the workpiece, and heat is flowing out from surface to the ambient and the workpiece eventually return to room temperature.

2.3 Modelling

The Digital Twin consist of building blocks that together complete the real asset. What type of building blocks the Digital Twin have depend on the product or process it is developed to represent. In this case, the digital twin will replicate the induction heating process. Software models for the converter and application is then necessary to simulate this process. These models will be mathematical models with inputs of data and outputs of the solution, which are solved through equations. The mathematical models can be grouped to be either physic-based or data-driven, based on knowledge about the system to simulate.

2.3.1.1 Physic-Based Model

A physic-based model is a conventional mathematical model. With this approach the physics is explained through equations which normally involves assumptions to get a solution. Due to often complex differential equations, the solution needs to be solved numerically. Methods that are commonly used for numerical solutions are FEM (Finite Element Method), FDM (Finite Difference Method) and FVM (Finite Volume Method). With increasing complexity of the equations, the more demanding the computation gets, and the solutions may get unstable and have errors. However, computational efficiency has over the years increased to handle more advanced and demanding equations, making physic-based models a good option for developing a digital twin [3]. Since the physics behind the converter is well known, it is natural to develop a physic-based model for this process. The application depends on what material to heat and will be more difficult to model because of unknown parameters that affect the temperature development, but a physic-based model is a good starting point for this process.

2.3.1.2 Data-Driven Model

Data-driven models are developed for ML and AI, which needs a large amount of data to learn from and are trained to eventually execute tasks on their own. These types of models are normally used when there is not sufficient information about the system, and e.g. linear and nonlinear regression models are used to fit the data [14]. Databases are important to store the data, and it requires a lot of filtering to get data of good quality for the algorithms to work properly. Once the model is trained, it is stable and able to make predictions, and are well suited for digital twin technology[3].

2.4 Digital Twin

The term “Digital Twin” first originated from Dr. Michael Grieves in 2003 at his course in Product Lifecycle Management (PLM) at the University in Michigan[15]. Grieves described the digital twin as a virtual representation of a physical product. The digital twin has evolved to not only represent products, but also processes and systems. IoT is the key for the advancement of digital twin technology, with widespread use of Wi-Fi and making it more affordable to collect data from sensors. For development of the digital twin, a model of the physical asset is connected to the real world enabled by data and IoT sensors with real-time information. This flow of updated information from sensor data gives the digital twin possibilities for optimizing performance and more informed decision making. This also allows the digital twin to predict how the asset will evolve or behave in the future [3]. A digital twin can have many usages, such as monitoring, predictive maintenance, and testing what-if scenarios. With the increasing implementation of digital twins in different sectors, the usage is expected to grow into a variety of areas in the coming years.

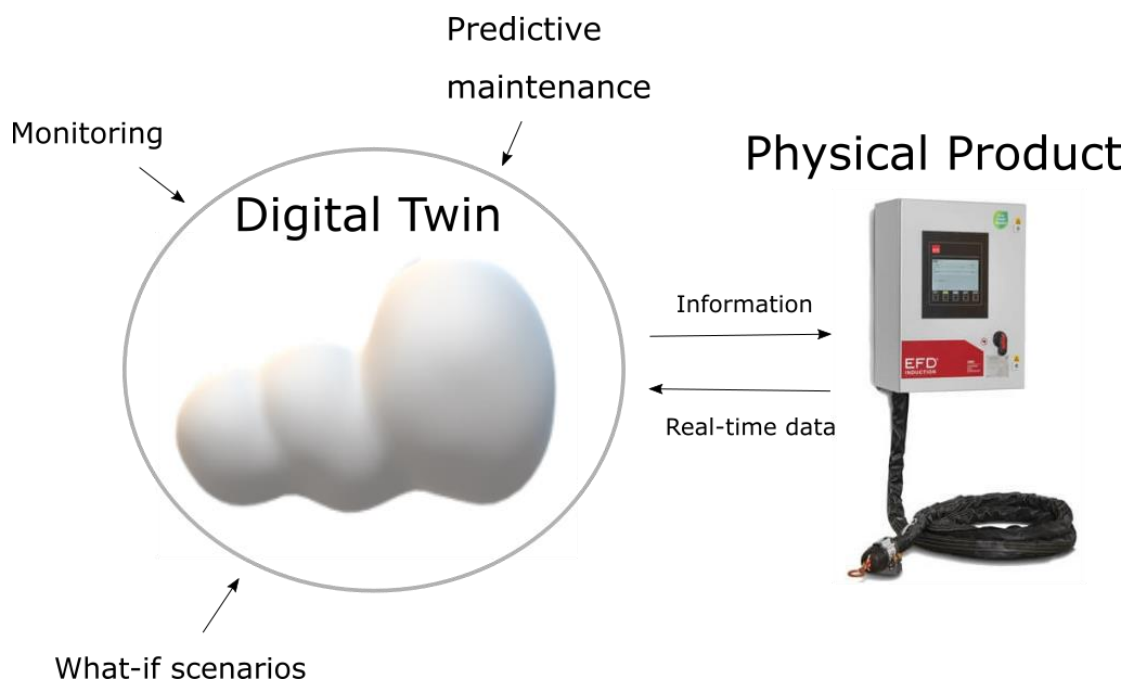


Figure 2.3: Illustration of a Digital Twin with different usages it covers

Companies in all sectors of the industry can have an advantage of implementing a digital twin in their production. In the manufacturing industry, a digital twin can benefit the total life cycle of a product. It can benefit the design, prototyping and testing phase of a product, but also production and usage of the digital twin [16]. Testing of the digital twin with the use of software models in different scenarios gives information about unexpected behaviors and can be improved before production of the actual product. This will reduce the cost and material by not producing prototypes during development and testing. Predictive maintenance of a product can reduce cost by scheduling maintenance after receiving information from the digital twin about present status and prevent changing parts unnecessarily. This usage of a

2 Theory

digital twin is especially beneficial for offshore windfarms that is not easily accessed, which result in a high cost on maintaining the wind turbines [17]. A digital twin can be implemented for monitoring large system, such as cities, aircraft, and buildings, and remotely control by feedback mechanisms, which would not be possible in real-time physically [3]. It can be simulated as a full system, or divided into subsystems and simulate with different environments to monitor and predict behavior [16]. Equinor is developing smart platforms through a digital twin called Echo [18]. By using HoloLens, an AR (argumented reality) solution from Microsoft, operators can view a 3D model of the platform together with real images. This allows them to monitor the platform and execute quality control with updated changes, without construction drawings.

There are different levels of digital twins that make them as detailed and complex as desired, without necessarily increase the complexity with a larger product or system. The digital twin is not required to be a representation of a complete asset but can represent parts that is of interest. Some examples in manufacturing are the propulsion system of a ship by Siemens [19], to a complete vehicle like Tesla. Every vehicle from Tesla receives updated software based on information from the vehicles sensor data to provide better service for the customer [20]. It has also been made a digital twin of big cities such as Singapore and Shanghai, to improve energy consumption and traffic flow [21]. Smart cities are increasing as a tool for planning and monitoring, and Dublin and Barcelona are currently in the development of designed their own smart cities [22].

Digital twin technology is increasing as a part of the industry and is expected to replace many previous methods for optimizing production and processes. There are few limitations on what a digital twin can represent, and it can be as detailed and accurate to the real asset that it is desired, which make them applicable for a range of uses.

3 Implementation of the Digital Twin

In this chapter, the building blocks of the digital twin is described. A short first introduction to the complete system is given, before each model is described in detail in the subchapters, including tutorials for installing necessary software and set up of hardware.

3.1 System Design

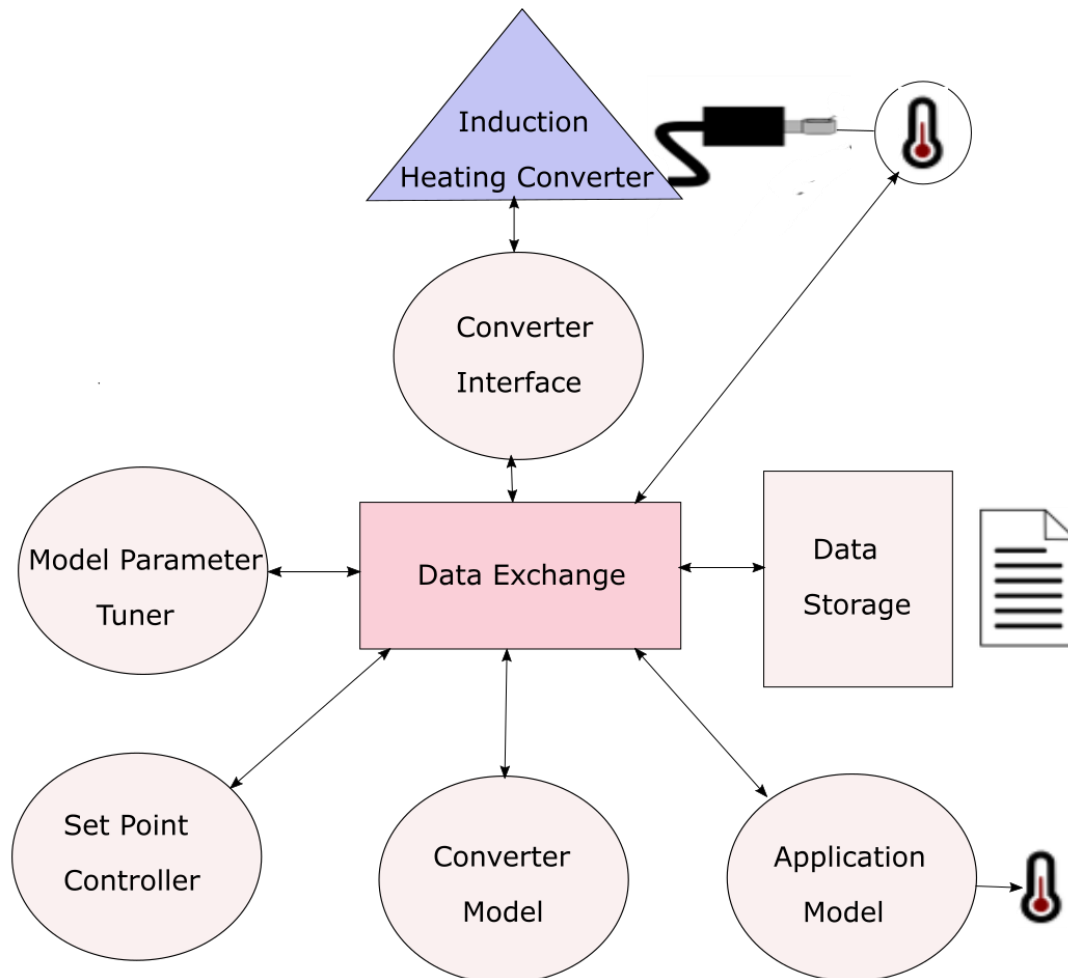


Figure 3.1: Building blocks of Digital Twin

The digital twin can be represented with the building blocks as shown in Figure 3.1. The different models are connected through a central hub, named Data Exchange, which will exchange the data flow. The Converter Model and Application Model will be the simulators of the actual system. These simulators will estimate how the system behaves when receiving inputs from the Set Point Controller. Real-time data is essential for the digital twin, and sensors from the real application will send information through the Data Exchange about the present status, and give the other models access to this information. Connection to the real system will be enabled by the Converter Interface. To store all information about the system

3 Implementation of the Digital Twin

and results after testing the application, a model for Data Storage is included. It is also of interest to optimize the Application Model from a Model Parameter Tuner by comparing the results from the Application Model and the real system with the error being as small as possible. To display and monitor the process, the Set Point Controller will also function as a user interface. The customer can then easily monitor the results from the experiments.

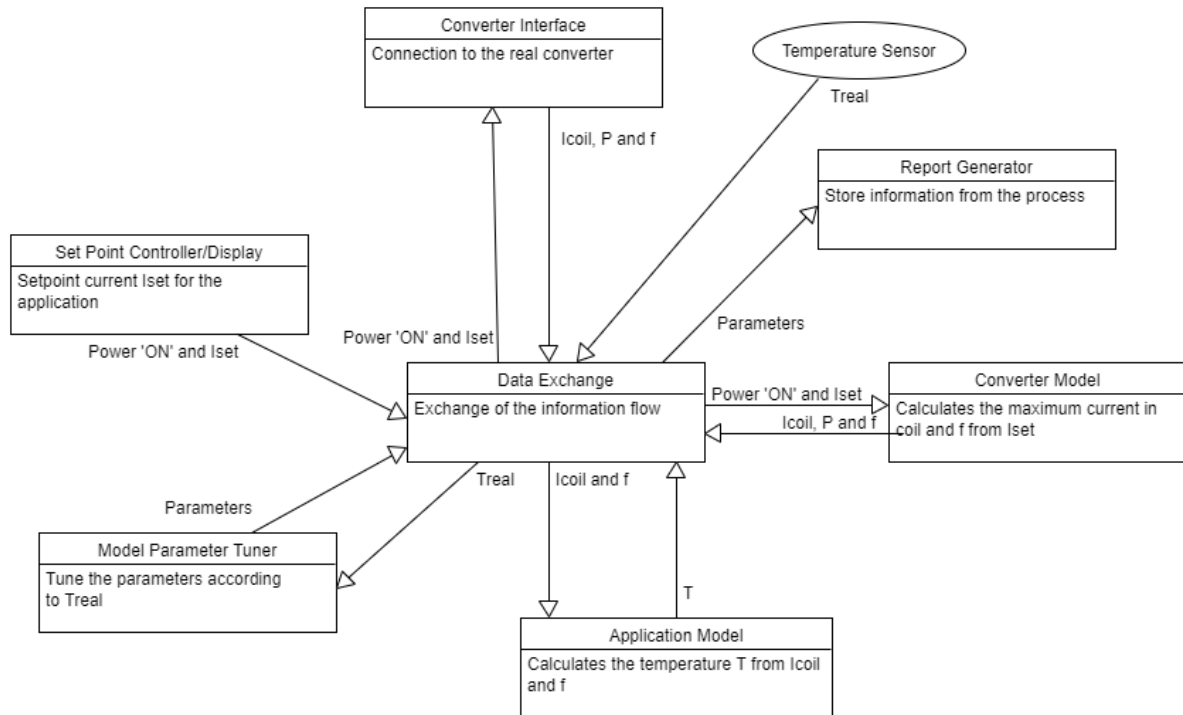


Figure 3.2: Information flow between the models of the Digital Twin

Figure 3.2 illustrates the information flow between the models of the Digital Twin. The main goal is to have a connection between these models to replicate the real induction heating process, and in the future improve the models to represent the actual application more accurately.

Tools used for these models are Spyder with Python 3.7 and Visual Studios 2019 with C#. It has not been focused on a literature search for tools to use for developing digital twins, but rather use known tools that are non-commercial and focus on developing this digital twin. Spyder and Visual Studios are basic software that is free of charge and is therefore chosen as the tools for developing the digital twin. The models developed for this digital twin can be accessed from Github: <https://github.com/ragnhildmold/MT-51-21.git>

3.2 Data Exchange with MQTT

3.2.1 Data Exchange with MQTT

To have a communication between the devices and exchange data, a messaging protocol called MQTT is used, which is a lightweight protocol for TCP/IP network (internet/intranet) used in many IoT devices. This allows several devices to be connected through internet, which is essential for IoT technology. MQTT authorize messages to be transported between clients connected through a network by a publish and subscribe model. There is no direct connection between the clients and messages are distributed through a server, named a broker. Clients can publish messages to a topic, while other clients can subscribe to the same topic and receive the messages, as shown in Figure 3.3. [23]

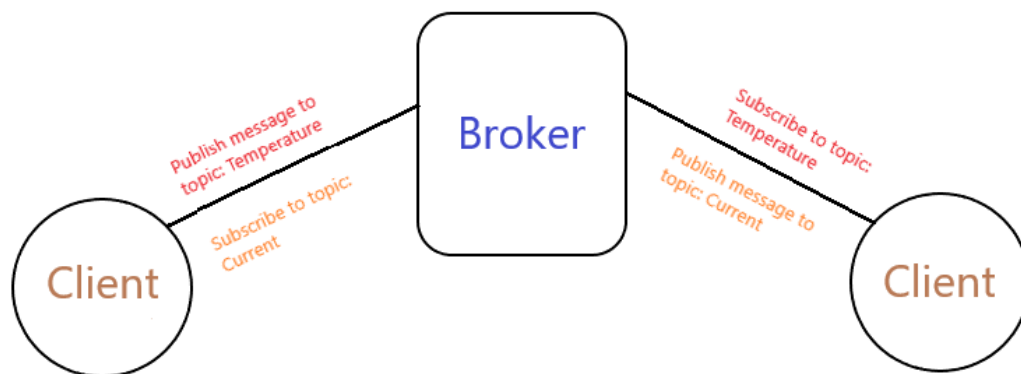


Figure 3.3: Illustration of client/broker communication in MQTT

As mentioned, a client can both be a publisher and a subscriber. The client can be any device that implements the MQTT library and is connected to a broker through internet. The MQTT library is available for several programming languages such as Python, C#, C++ and others. It is an easy and straightforward method and is applicable for a large range of devices [24].

The broker is responsible for receiving, filtering, and distributing the messages that are published and subscribed to a topic. It stores messages from all the clients that have an active session. The broker is also responsible for authentication and authorization of clients. All messages must go through a broker, which make it a central hub in MQTT [24]. Several brokers are available and open source for users, for example Mosquitto, HiveMQ and EMQ X.

MQTT is available for most operating systems, such as Windows, Linux, and Mac OS. Raspbian, the operating system for Raspberry Pie, has also MQTT solutions for add-on boards to enable automation control and monitoring. This will be explained more in detail in section 3.8.3 regarding development of the Converter Interface.

3.2.2 Set up of MQTT Broker

Tutorial for setup of MQTT Broker is given in Appendix B.

3.2.3 Connection to broker

MQTT runs over TCP/IP (Transmission Control Protocol/Internet Protocol) protocol, which is necessary for internet access. TCP/IP defines how data is communicated through network. To establish connection in MQTT, the client and broker must have a TCP/IP stack. There is never direct connection between clients, but the information is delivered through the broker. To initiate connection to a broker, the client must send a CONNECT message. While the broker must respond with a CONNACK message and a status code. The CONNECT message is a request to connect to broker, and the CONNACK message is an acknowledgement to the connection. These messages are part of the MQTT control packet types in the protocol [25].



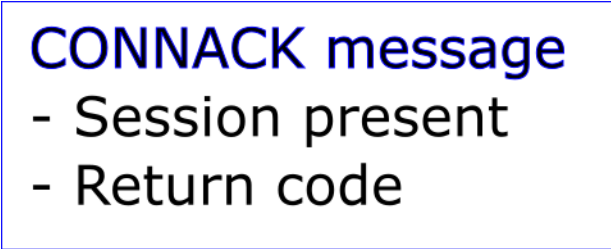
CONNECT message

- ClientId
- Clean Session

Figure 3.4: Content of a CONNECT message

When the client sends the CONNECT message, some information is necessary for the broker to establish connection, mentioned in Figure 3.4. A client identifier (ClientId) gives identification and status of the client to the broker. The ClientId should be unique for the client. It is possible to send an empty ClientId, while that will result in a connection with no state. The consequences of this, is if the network connection is temporarily lost, the broker can refuse reconnection with the client [24].

A clean session flag tells if the client wants a persistent session. The clean session can either be true or false. A clean session set to true, will result in a non-persistent session and the broker will not store any information from the client. This is a requirement when the ClientId is empty, or else the broker will reject the connection. When the clean session is set to false, the broker will store all information the client has subscribed with a QoS (Quality of Service) level of 1 or 2 [24]. QoS will be explained more in detail in 3.2.5.



CONNACK message

- Session present
- Return code

Figure 3.5: Content of a CONNACK message

3 Implementation of the Digital Twin

The brokers response with the CONNACK message (Figure 3.5), which contains two data entries; the session present flag and the connect return code. The session present flag tells if the client has a previous persistent session with the broker. If the clean session is set to true from the CONNECT message, the session present flag will always be set to false since there will never be a previous persistent session. If the clean session is set to false, the session present flag can either be true or false, depending if there is a previous persistent session and stored information from the broker [24].

The connect return code is a returned number which indicates a specified response. To have a successful connection, the return code needs to be 0 [24].

3.2.4 Publish to topic



Figure 3.6: Content of a PUBLISH message

When the connection is established, the client can publish to a topic of interest. The client can only publish to one topic at a time and is not possible to publish to several topics. The message to be published by the client, known as the PUBLISH message (Figure 3.6), is required to contain some information. The topic for publish of message, payload, the QoS level and the retain flag must be set true or false. The payload is the actual message to be sent. MQTT is data-agnostic, meaning that information from different types of databases can be sent [26]. The QoS level indicated the quality of service of the message. When the retain flag is set to false, the message is not stored with the broker. Is the flag set to true, the message will be stored [27]. While the topic and payload are necessary to be defined for the PUBLISH package to be published, the QoS level and retain flag will by default be set to 0 and false, respectively, if nothing else is specified. There can only be published one retained message at a time per topic by the broker. When a client start subscribing on a topic, the retained message will be published immediately.

3.2.5 Quality of Service (QoS)

The reliability of receiving the messages is handled through the Quality of Service (QoS) level. There are three levels of security in MQTT, from zero to two. QoS level 0 is the minimal level and have no guarantee that the message, the PUBLISH message, is delivered. There is no feedback if the message is received, and the message is not stored with the sender. QoS level 1 guarantees delivery at least once. The sender then stores the message

3 Implementation of the Digital Twin

until it has received a response from the receiver, known as a PUBACK message. If the sender receives no response, the sender will resend the PUBLISH message. The highest level is QoS level 2, and this guarantees delivery exactly once. This is also the slowest and safest level. The sender and receiver have two sent/received flows, which will confirm the delivery. The receiver stores a reference to the original PUBLISH message to ensure that the message is not processed a second time. If the clients (sender and receiver) use different QoS level, the broker will use the lower level for delivery. Which QoS level to define, depends on the message to be delivered. If the messages are not of big importance and some can be lost occasionally, the level zero can be used. If it is important to receive the messages and it is possible to handle duplicates, the level 1 can be used. Level 2 is used when it is critical to receive the messages only once. Every level requires internet connection to be able to send the messages. But if there is offline clients, only level 1 and 2 can queue the messages and send when the client is available again [28].

3.2.6 Subscribe to topic



Figure 3.7: Content of a SUBSCRIBE message

To receive a message from the topic of interest, the client must send a subscribe message to the broker, called a SUBSCRIBE message (Figure 3.7). This message contains the topic to subscribe to and the QoS level. Unlike for publishing messages, it is possible to subscribe to several topics. The broker will respond with a SUBPACK message, which is a confirmation of the subscription to the client [26].

A feature that can be used in a subscribe function, is defining root topic and subtopics. The root topic is the main topic, while the subtopics are the lower levels of the root topic. By using a slash '/', several subtopics can be defined under the root topic. To subscribe to all subtopics, a slash and hashtag '/#' can be inserted after the root topic. This is called a wildcard and can only be used to subscribe to topics [29]. The subscriber will then receive all messages published to the root topic.

3.3 Application Model

3.3.1 Heat transfer through workpiece

To see how the temperature distributes through the workpiece, the model is represented by a system of DAEs (Differential-Algebraic-Equations). This system contains differential and algebraic equations that explains mathematically how a dynamic system behaves. The heat flow in the workpiece is of interest and the thermal energy balance is an essential equation for

3 Implementation of the Digital Twin

this case. As stated by the first law of thermodynamics, the total energy of a closed system is constant, and energy can only be transformed from one form to another. The change of internal energy is equal the heat flow and work done by the system to the surroundings. The sum of heat flowing into the system will then be equal heat flowing out of the system.

The thermal energy balance is stated as this:

$$\frac{dU}{dt} = \dot{H}_i - \dot{H}_e + \dot{W}_v + \dot{W}_f + \dot{Q} \quad (3.1)$$

Where:

- $\frac{dU}{dt}$: Change of internal energy [J/s]
- \dot{H}_i : Influent enthalpy flow [J/s]
- \dot{H}_e : Effluent enthalpy flow [J/s]
- \dot{W}_v : Work of volume change [W]
- \dot{W}_f : Work of friction [W]
- \dot{Q} : Heat flow [W]

Since there is no mass flow in the workpiece, the terms \dot{H}_i and \dot{H}_e can be neglected from the balance equation. There will also be no friction since there is no movement of mass and the volume of the workpiece will be constant and not expanded, thus the terms \dot{W}_f and \dot{W}_v can also be neglected.

Because of these assumptions, the thermal energy balance can be reduced to only heat flow:

$$\frac{dU}{dt} = \dot{Q} \quad (3.2)$$

And the algebraic equations to represent the model will be:

$$U = m \cdot \hat{H} \quad (3.3)$$

$$m = \rho \cdot V \quad (3.4)$$

$$V = A \cdot \Delta x \quad (3.5)$$

$$\hat{H}_i = \hat{c}_p \cdot (T_i - T^\circ) \quad (3.6)$$

3 Implementation of the Digital Twin

$$\dot{Q}_i = A \cdot (\ddot{Q}_{i-1} - \ddot{Q}_i) \quad (3.7)$$

$$\ddot{Q}_i = -k \cdot \left(\frac{T_{i+1} - T_i}{dx} \right) \quad (3.8)$$

Where:

- U: Internal energy [J]
- m: Mass [g]
- V: Volume [m³]
- \widehat{H}_i : Specific influent enthalpy [J/g]
- \dot{Q}_i : Heat flow [W]
- \ddot{Q}_i : Heat flux [W/m²]

The source from the temperature increase in the workpiece is due to Joule heating. This originates from the power dissipated from the eddy currents flowing through the resistance of the material:

$$\text{Joule heating} = P_{diss} = I^2 \cdot R \quad (3.9)$$

The heat flow in the workpiece can be illustrated as in Figure 3.8. The heat affected volume will be the surface of the workpiece. While heat is dissipated by the induced current flowing in the material, the heat will start to flow inwards to the bulk volume by the temperature difference.

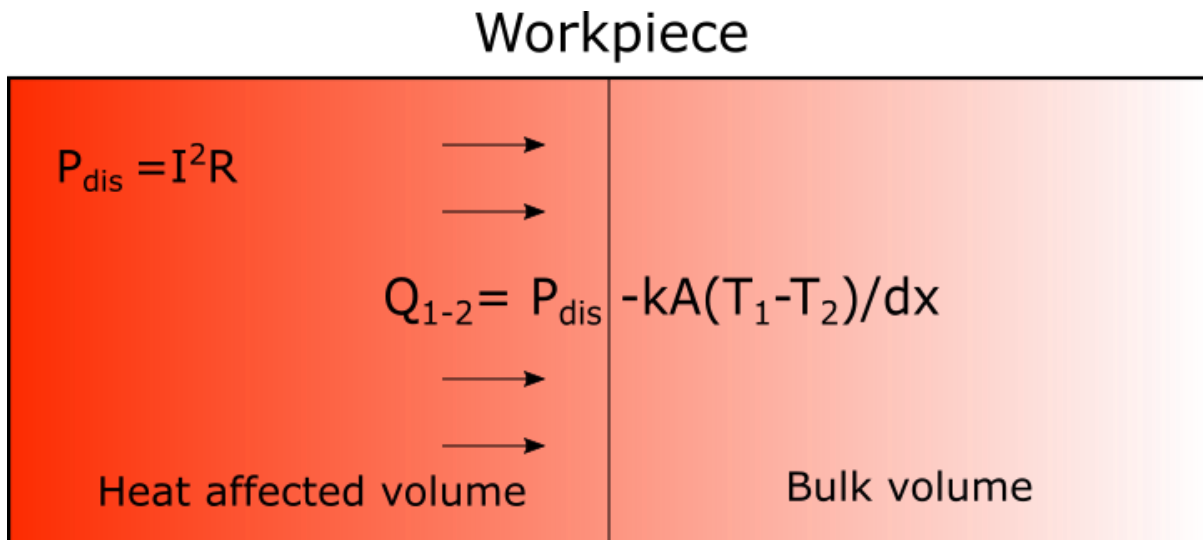


Figure 3.8: Illustration of heat flow in workpiece

3.3.2 Application Model with Python

The first Application Model will be a simple simulator of the heating process. To start this process, an alternating current will flow in the coil and induce eddy currents in the surface of the workpiece. Heat will be dissipated at the surface due to the resistance as the induced current is flowing through the material. The heat will flow inwards in the workpiece by conduction since there will be a temperature difference. Heat will always flow from hotter to colder regions until there is no longer a temperature difference. The heat will then flow to the surroundings and the temperature in the workpiece will return to room temperature. For this simulator, the workpiece is divided in two elements. One element is the surface of the workpiece and the other element is the bulk volume of the workpiece. Each element is assumed homogenous and treated with constant properties. The thermal energy balance is used at each element and the sum of heat flow is calculated over time. From this change in heat flow over time, the temperature in the elements can be estimated.

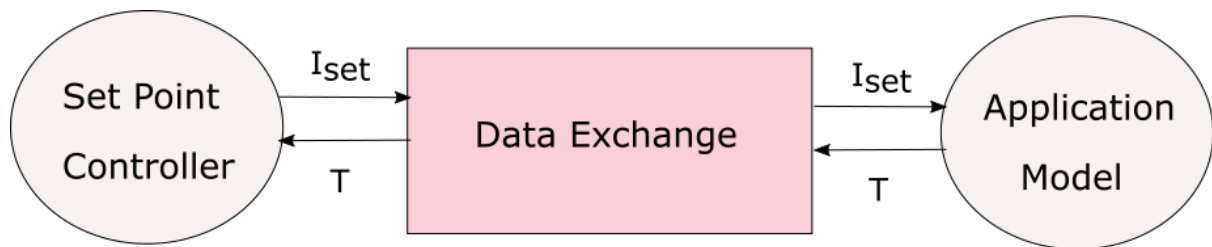


Figure 3.9: Communication between Set Point Controller and Application Model through Data Exchange

The Application Model will receive a setpoint current I_{set} from the Set Point Controller and estimate the temperature T and publish back through Data Exchange, as illustrated in Figure 3.9. The Application Model is built in the Spyder environment using Python as the programming language. Since this is a basic model and the workpiece are not known, some simplifications have been done in the script:

- The induced current is the same as the coil current.
- The mass and heat capacity are combined in one factor, mc , when calculating internal energy:

$$U = m \cdot \hat{c}_p \cdot (T - T^\circ) = mc \cdot (T - T^\circ) \quad (3.10)$$

- The heat flux is joined in the heat flow equation by combining the thermal conductivity and geometric properties in one factor, k :

$$\dot{Q}_i = \left(\frac{k \cdot A}{dx} \right) \cdot (T_{i+1} - T_i) = k \cdot (T_{i+1} - T_i) \quad (3.11)$$

The material properties depend on what workpiece is to be heated. But the mechanism of heat flow will still be the same due to the heat flow equation. The heat will start to flow as long there is a temperature difference. The first element in the outer surface will have an increase in temperature since the induced current will dissipate power as heat in the element. Heat

3 Implementation of the Digital Twin

always flows from hotter to colder regions, and due to this difference, the heat will start to flow inwards the bulk volume by conduction. The temperature in the second element will then start to increase. This flow of heat will continue until there is no temperature difference between the elements and the temperature is the same. This application model is to give a representation of how the temperature in the workpiece will respond to the change in current in the coil.

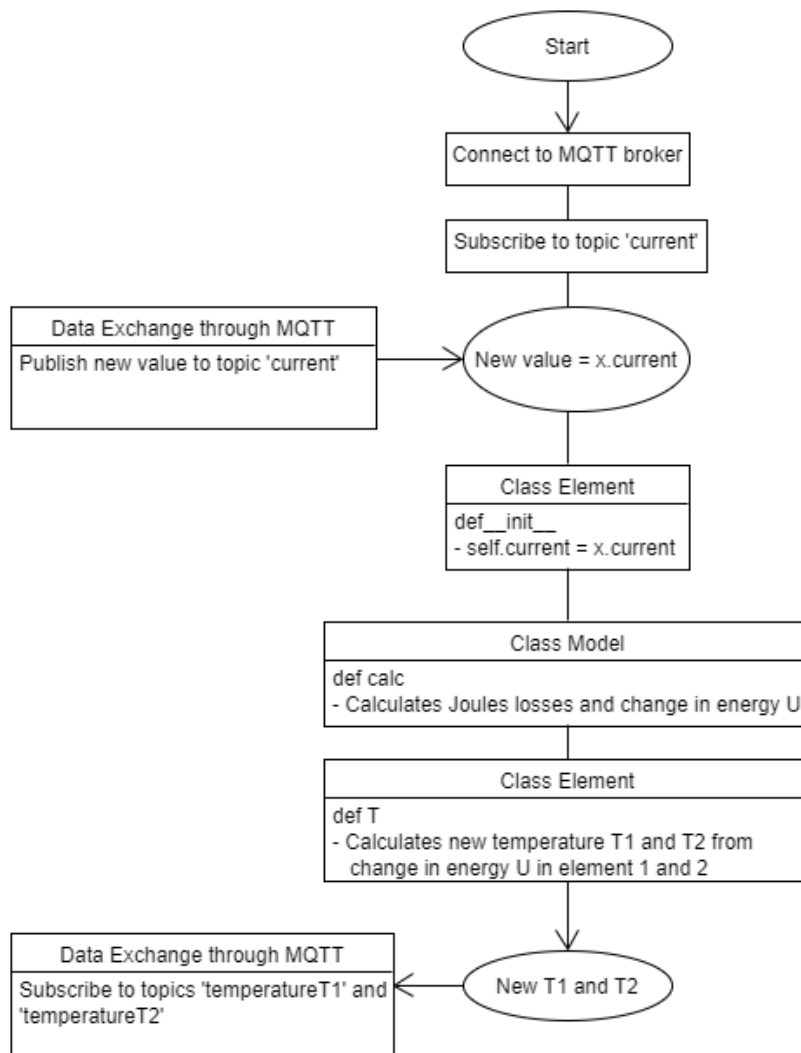


Figure 3.10: Flow Diagram of communication between MQTT and Application Model in the Python script

The flow diagram in Figure 3.10 illustrates the basic steps in the Application Model. At first the Application Model connects to the MQTT broker and subscribe to the topic ‘current’ for receiving new value for the current from the Set Point Controller through Data Exchange. This new value is updated in the model and included in the calculation of Joule losses and change in energy in the elements. The new temperature in the elements is then updated and published to the topics ‘temperatureT1’ and ‘temperatureT2’.

3 Implementation of the Digital Twin

```
67 #Class for each element
68 class Element:
69     def __init__(self):
70         self.U = 0
71         self.sourceTerm = 0
72         self.heatFlux = 0
73         self.mc = 1000
74     def T(self):
75         return self.U/self.mc
76     def initT(self,temperature):
77         self.U=self.mc *temperature
78         return self.U/self.mc
79
80 #Class for the total model
81 class Model:
82     def __init__(self):
83         self.run=False
84         self.initTime=time()
85         self.lastCalcTime=time()
86         self.newCalcTime=self.lastCalcTime
87         self.e1 = Element()
88         self.e2 = Element()
89         self.k=0
90         self.current=0
91         self.resistance=0
92         self.calcInterval=0.1
93         self.q=False
94         self.filename="SimData.csv"
95         self.__dataToFile__=False
96         self.__fileHandle__=""
```

Figure 3.11: Classes in the Python script for the Application model

The script has two classes, show in Figure 3.11. One class is for the elements the workpiece is divided into. In this class the variables for the thermal energy balance are defined with initial values and functions for calculating the initial temperature, and the temperature as the internal energy starts to change by the heat flow. The other class is for the total model. The elements are defined as e1 and e2 and calls to the Element class for it to initiate the functions.

```

135     def calc(self):
136         #store current time
137         self.newCalcTime=time()
138
139         # heatflux from element 1 to 2
140         heatflux12=self.k*(self.e1.T()-self.e2.T())
141
142         # Joule heating
143         jouleLosses=float(self.current)**2*self.resistance
144
145         #Calculate timestep
146         timeSinceLastCalc=self.newCalcTime-self.lastCalcTime
147
148         #Calculate change in energy in elements
149         self.e1.U += (jouleLosses-heatflux12)*timeSinceLastCalc
150         self.e2.U += heatflux12*timeSinceLastCalc
151
152         #store calculation time for next step
153         self.lastCalcTime=self.newCalcTime

```

Figure 3.12: Calculation of heat flow, Joule losses and the change in internal energy under class Model

Figure 3.12 show the function for calculation of the heating process under class ‘Model’ and will be the main task of the simulator. In this function the heat flow and Joule losses are calculated. For each time step, the internal energy is calculated and the temperature in each element will be recalculated.

```

17 from paho.mqtt import client as mqtt_client
18
19
20 broker = 'localhost'
21 # port can be defined, or 1883 will be chosen by default
22 port = 1883
23 # topic to subscribe messages from
24 subscribe_topic = "current"
25 # topics to publish messages to
26 publish_topic1 = "temperatureT1"
27 publish_topic2 = "temperatureT2"
28 # generate client ID
29 client_id = f'python-mqtt-{random.randint(0, 100)}'

```

Figure 3.13: Declaring variables for MQTT Paho client

To connect to a MQTT broker, the Paho client library is implemented in the Python script (Figure 3.13). This is a MQTT client library which provides functions to publish and subscribe to topics which are necessary for a client/broker connection [30].

Since the Mosquitto software is installed, ‘localhost’ can be used as MQTT broker. Defining a port is normally not necessary, and port 1883 will always be chosen as default if nothing else is defined. The topics to publish and subscribe to must be defined. Since the workpiece is represented with two elements, there will be a temperature calculated in each element. Each calculated temperature must then be published to a topic for each element. The messages published from the Set Point Controller to the topic ‘current’, will be received to this model by the subscribe function.

3 Implementation of the Digital Twin

At last, a client id must be specified. This client id should be unique for each client and broker to ensure that the messages are stored and information on what state the client is (connected or disconnected). The client id can also be left blank, which will result in that the broker doesn't store any messages or state. [24]

```
30 def connect_mqtt() -> mqtt_client:
31     def on_connect(client, userdata, flags, rc):
32         if rc == 0:
33             print("Connected to MQTT Broker!")
34         else:
35             print("Failed to connect, return code %d\n", rc)
36
37     client = mqtt_client.Client(client_id)
38     client.on_connect = on_connect
39     client.connect(broker, port)
40     return client
```

Figure 3.14: Defining function for connecting to MQTT broker

Before the client can publish messages or subscribe to topic, the client needs to initiate a connection to the broker by sending a CONNECT message, shown in Figure 3.14. In general, if the CONNECT message is malformed or uses too much time to send, the broker closes the connection [24]. A client object is created for the call-back function, 'on_connect'. This callback function is to ensure successful connection to the broker. If the return code (rc) is equal zero, the connection is successful. To verify this connection, the call-back is bonded. Now the client is safe to connect to the broker [31].

When there is a connection to the broker, the client can subscribe to the topic 'current' and receive messages (Figure 3.15).

```
43 def subscribe(client: mqtt_client):
44     def on_message(client, userdata, msg):
45         print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
46         x.current = float(msg.payload.decode())
47
48     client.subscribe(topic)
49     client.on_message = on_message
```

Figure 3.15: Function from Paho client library for subscribing in Python

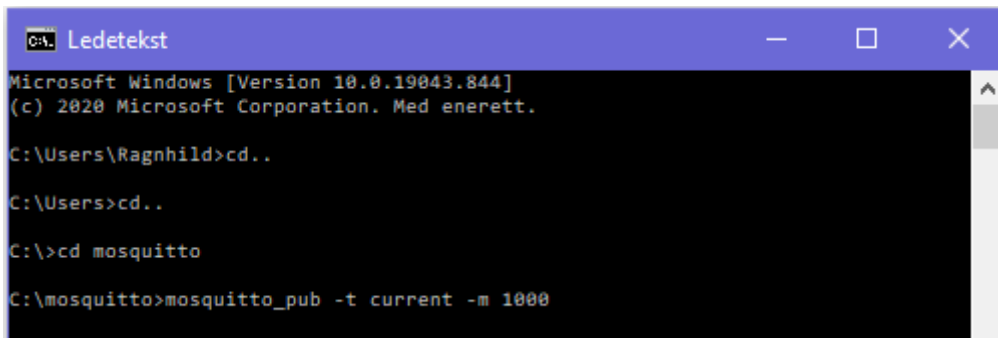
The 'on_message' is a call-back function for when a message is received from the broker. The received value 'float(msg.payload.decode())' will be set as the new value 'x.current' in the script for calculating Joule losses and temperature as shown in Figure 3.12.

```
52 client = connect_mqtt()
53 subscribe(client)
54 client.loop_start()
```

Figure 3.16: Paho client library for running the MQTT functions in Python

When the connection is established, the loop can start and run the script (Figure 3.16). By using the 'mosquitto_pub' function in the Windows command line, as seen in Figure 3.17, a new value of the current is published to the topic 'current'.

3 Implementation of the Digital Twin



```
C:\Users\Ragnhild>cd..
C:\Users>cd..
C:\>cd mosquitto
C:\mosquitto>mosquitto_pub -t current -m 1000
```

Figure 3.17: Mosquitto client function 'mosquitto_pub' in the Windows command line for publish value of current

Since this model subscribe to the same topic, the messages are received in the IPython console in the Spyder environment (Figure 3.18).

```
In [12]: runfile('C:/SPB_Data/.spyder/Master MT-51-21/
CalcTemperature.py', wdir='C:/SPB_Data/.spyder/Master
MT-51-21')
Connected to MQTT Broker!
      Time ,      Icoil ,      T e1 ,      T e2
Received `1000` from `current` topic      20.0
```

Figure 3.18: Message received in the subscribing topic in the Spyder environment

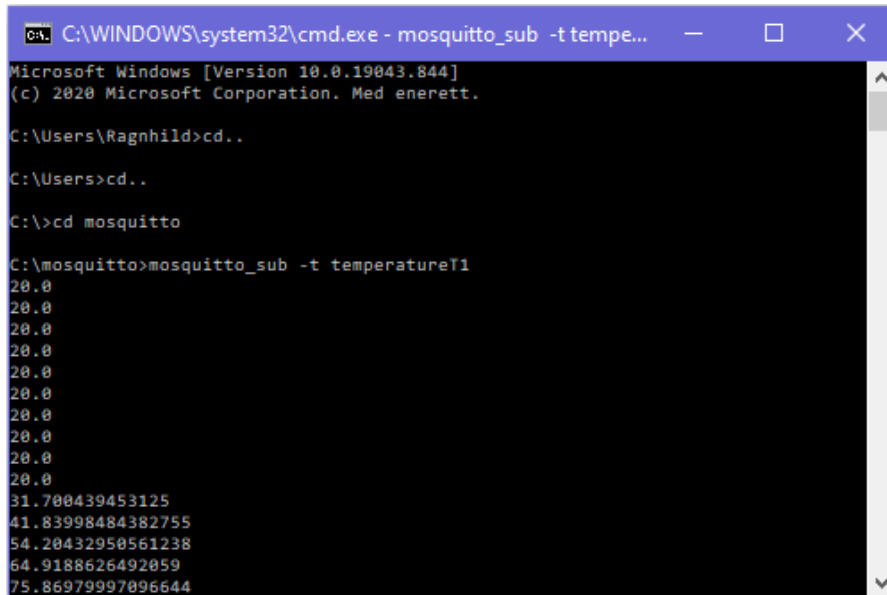
After the temperature has been calculated, the values will be published to the topics that are defined (Figure 3.19). The publish function can have four parameters; topic, payload, QoS level and retain flag. The parameters that are required is the topic and payload, which is the message to be sent.

```
164     client.publish(publish_topicT1, x.e1.T())
165     client.publish(publish_topicT2, x.e2.T())
```

Figure 3.19: Function from Paho client library for publishing in Python

To display the temperature calculated in the first element, the 'mosquitto_sub' function can be used in the Windows command line. As mentioned earlier, the topic must be the same topic the model publishes to, which is 'temperatureT1' and 'temperatureT2'.

3 Implementation of the Digital Twin



```
C:\WINDOWS\system32\cmd.exe - mosquitto_sub -t tempe...
Microsoft Windows [Version 10.0.19043.844]
(c) 2020 Microsoft Corporation. Med enerett.

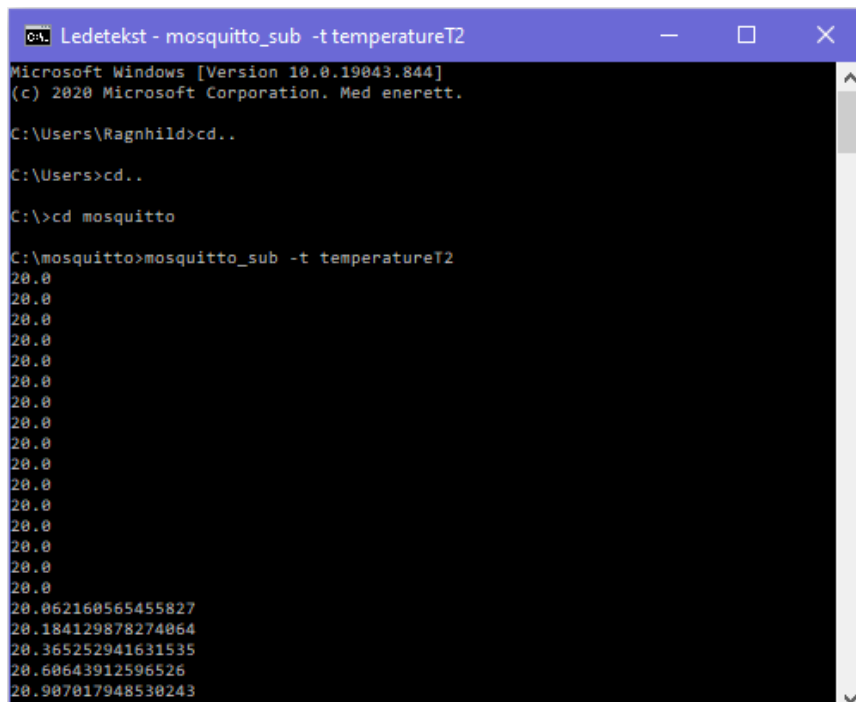
C:\Users\Ragnhild>cd..

C:\Users>cd..

C:\>cd mosquitto

C:\mosquitto>mosquitto_sub -t temperatureT1
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
31.700439453125
41.83998484382755
54.20432950561238
64.9188626492059
75.86979997096644
```

Figure 3.20: Mosquitto client function 'mosquitto_sub' for subscribing on topic 'temperatureT1' in Windows command line



```
C:\Users\Ragnhild>cd..

C:\Users>cd..

C:\>cd mosquitto

C:\mosquitto>mosquitto_sub -t temperatureT2
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.0
20.062160565455827
20.184129878274064
20.365252941631535
20.60643912596526
20.907017948530243
```

Figure 3.21: Mosquitto client function 'mosquitto_sub' for subscribing on topic 'temperatureT2' in Windows command line

The temperatures are recalculated and published every second. The defined initial value of the current is 0 and the temperature of the workpiece will be equal the ambient temperature, which is defined as 20°C. When the value of the current is published to the topic, the value in the model is updated to 1000 A in the 'on_message' function shown in Figure 3.15.

3 Implementation of the Digital Twin

This will increase the Joule heating and cause the temperature increase in the first element (Figure 3.20), which then will cause a temperature increase in the second element by calculated heat transfer (Figure 3.21).

3.3.3 Application Model with Python using FEniCS

To build a more realistic and accurate application model of the heating process of the workpiece, FEniCS has been implemented in the Python script. Dmitry Ivanov at EFD Induction has shared his model with FEniCS implemented, which has been further developed for this Application Model. FEniCS is a software library for solving partial differential equations (PDEs) using the finite element methods (FEM) [32]. The PDE to be solved is the heat equation, which is one-dimensional and time-dependent differential equation, and is expressed as:

$$\frac{\partial u}{\partial t} = k \cdot \frac{\partial^2 u}{\partial x^2} \quad (3.12)$$

FEniCS consists of several components that have different roles and together form the software. The benefit of implementing FEniCS in the Python script, is the simplification of the programming code. Even if it is a complex mathematical code to program, the FEniCS library helps with keeping the code short and compact without complicated coding [32].

FEniCS is currently not available for Windows. To get access to the FEniCS software library, the program must be run in a Linux operating system. Instead of having a virtual machine, Windows Subsystem for Linux (WSL) gives access to Linux software programs in the Windows environment through a Linux distributor. Ubuntu is chosen as a Linux distributor since it is user-friendly and suitable for beginners. A tutorial for installing FEniCS in Ubuntu is given in Appendix C.

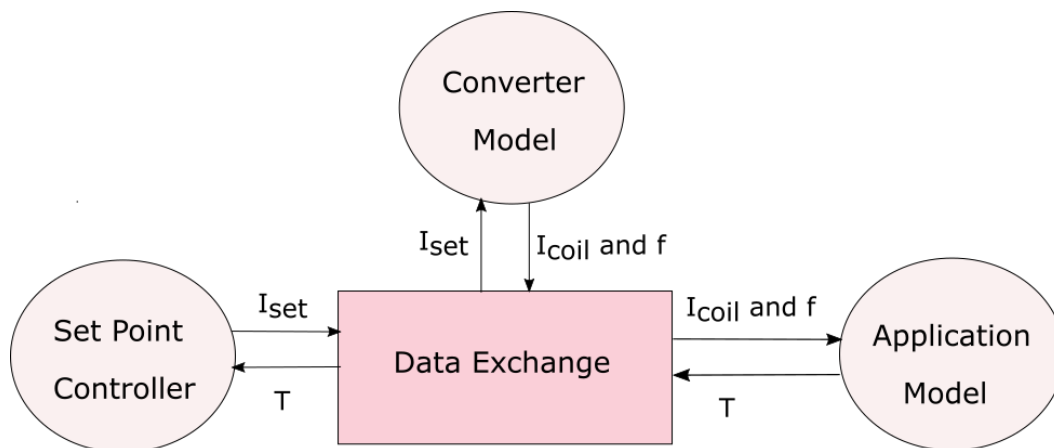


Figure 3.22: Information flow between Set Point Controller, Converter Model, and the Application Model

In the previous Application Model, the current was the input to the calculations. In this model, the frequency will also be a necessary input for the calculations. Figure 3.22 illustrates the information flow through the Data Exchange. The setpoint current I_{set} is delivered from the Set Point Controller to the Converter Model. This model will calculate the

3 Implementation of the Digital Twin

maximum current that can run in the coil, including the frequency. These parameters are then published to the Application Model through Data Exchange.

The Paho client library is imported in the model to use the functions for MQTT communication, seen in Figure 3.23.

```

23 broker = '192.168.10.129'
24 # port can be defined, or 1883 will be chosen by default
25 port = 1883
26 # topic to subscribe messages from
27 subscribe_topics = [("convmodel/frequency",0), ("convmodel/current",0),("setpoint/mqtt",0)]
28 # topic to publish messages to
29 publish_topic1 = "appmodel/temp"
30 # generate client ID
31 client_id = f'python-mqtt-{random.randint(0, 100)}'

```

Figure 3.23: Defined variables for the Paho client library

The Application Model will subscribe to the topics that the Converter Model will publish to with the updated values of current and frequency.

```

54 def on_message(client, userdata, msg):
55     if msg.topic == "convmodel/frequency":
56         print(f"Received '{msg.payload.decode()}' from '{msg.topic}' topic")
57         I.F = float(msg.payload.decode())
58     if msg.topic == "convmodel/current":
59         print(f"Received '{msg.payload.decode()}' from '{msg.topic}' topic")
60         Iset = float(msg.payload.decode())
61         I.Hs = Iset/0.071

```

Figure 3.24: Paho client functions for handling received messages

The received parameters from the Converter Model will be updated in the Application Model through the 'on_message' function, shown in Figure 3.24.

```

103 class TransientTSolver:
104     def __init__(self, mesh, ds, Ti, P, theta, Hconv, Text, material):
105         self.mesh = mesh
106         self.ds = ds
107         self.Ti = Constant(Ti)
108         self.V = FunctionSpace(self.mesh, 'Lagrange', 1)
109         self.P = project(P, self.V)
110         self.T = TrialFunction(self.V)
111         self.v = TestFunction(self.V)
112         self.T_n = Function(self.V)
113         self.T_n = project(self.Ti, self.V)
114         self.r = Expression('x[0]', degree=2)
115         self.Hconv = Constant(Hconv)
116         self.Text = Constant(Text)
117         self.theta = Constant(theta)
118         self.InitTime = time()
119         self.LastCalcTime = time()
120         self.NewCalcTime = self.LastCalcTime
121         self.dt = 1
122         self.m = material
123         self.F = (1.0/self.dt)*self.r*self.m.C*self.m.gamma*(self.T-self.T_n)*self.v*dx + \
124             self.theta*self.steady(self.T, self.v, self.P)+ \
125             (1.0-self.theta)*self.steady(self.T_n, self.v, self.P)
126         self.Tsol = Function(self.V)
127         self.T_n.interpolate(self.Ti)

```

Figure 3.25: Functions from the FEniCS library implemented in the Application Model

3 Implementation of the Digital Twin

Figure 3.25 show the implementation of FEniCS in the Application Model. FEM requires the PDE to be expressed as a variational problem. To formulate it as a variational problem, the PDE first needs to be multiplied with a test function 'self.v'. The resulting equation is integrated over the domain, denoted with the differential element dx, and the second derivatives is partial integrated. The trial function 'self.T' is the unknown function to be approximated. The test and trial function belongs to the function space 'self.V', which defines the properties of the function. A mesh will be created, while 'Lagrange' is the type of the finite element and '1' is the degree of the element. The element will be a one-dimensional triangle with nodes at the three vertices. The source term 'self.r' has an expression object representing a formula and the degree of accuracy. The formula is a mathematical expression defining a known analytical solution of the PDE, which is 'x'. 'self.F' defines the equation to be solved for each time step 'dt' in the loop and is assigned to 'self.Tsol' for each numerical solution of the temperature, to contain the previous value, seen in Figure 3.26 [32].

```
141 def solve(self, dt):
142     self.dt = dt
143     self.F = (1.0/self.dt)*self.r*self.m.C*self.m.gamma*(self.T-self.T_n)*self.v*dx + \
144             self.theta*self.steady(self.T,self.v,self.P)+ \
145             (1.0-self.theta)*self.steady(self.T_n, self.v, self.P)
146     solve(lhs(self.F) == rhs(self.F), self.Tsol)
147     self.T_n.assign(self.Tsol)
```

Figure 3.26: Function under Class TransientSolver for solving numerical solution of the temperature

The material properties of the workpiece to simulate is given in its own class, shown in Figure 3.27. Stainless Steel 316L is the material of the workpiece in this case.

```
77 class Material:
78     def __init__(self, ro=7.4e-7, mur=1., C=500., gamma=8000., k=14.):
79         self.ro = ro
80         self.mur = mur
81         self.C = C
82         self.gamma = gamma
83         self.k = k
```

Figure 3.27: Class Material for defining material properties

For each time step, the matrix is compiled and solves a new solution for the temperature, seen in Figure 3.28. In addition, the temperature is published to the topic 'appmodel/temp' by the publish function.

3 Implementation of the Digital Twin

```
201 solv1 = TransientTSolver(mesh, ds, Ti, P, theta, Hconv, Text, steel)
202
203 while (done != True):
204     solv1.NewCalcTime = time()
205     TimeSinceLastCalc = solv1.NewCalcTime - solv1.LastCalcTime
206     I.targettime = TimeSinceLastCalc
207     solv1.solve(TimeSinceLastCalc)
208     P.acceptNewHs(Hs=I.Hs)
209     P.acceptNewF(F=I.F)
210     solv1.acceptNewPower(P)
211
212     client.publish(publish_topic1, str(solv1.Tsol.vector().get_local()))
213
214     solv1.NewCalcTime = solv1.LastCalcTime
215     sleep(1)
```

Figure 3.28: the PDE is solved for each time step and published to topic by MQTT

The benefit with FEM is the possibility to observe the temperature distribution inwards the workpiece. This model takes one point at the surface and simulate the distribution inwards in one dimension. This is not possible with a normal temperature sensor. The result from the simulation is shown in Figure 3.29, where the temperatures are represented by the y-axis, and the depth by the x-axis.

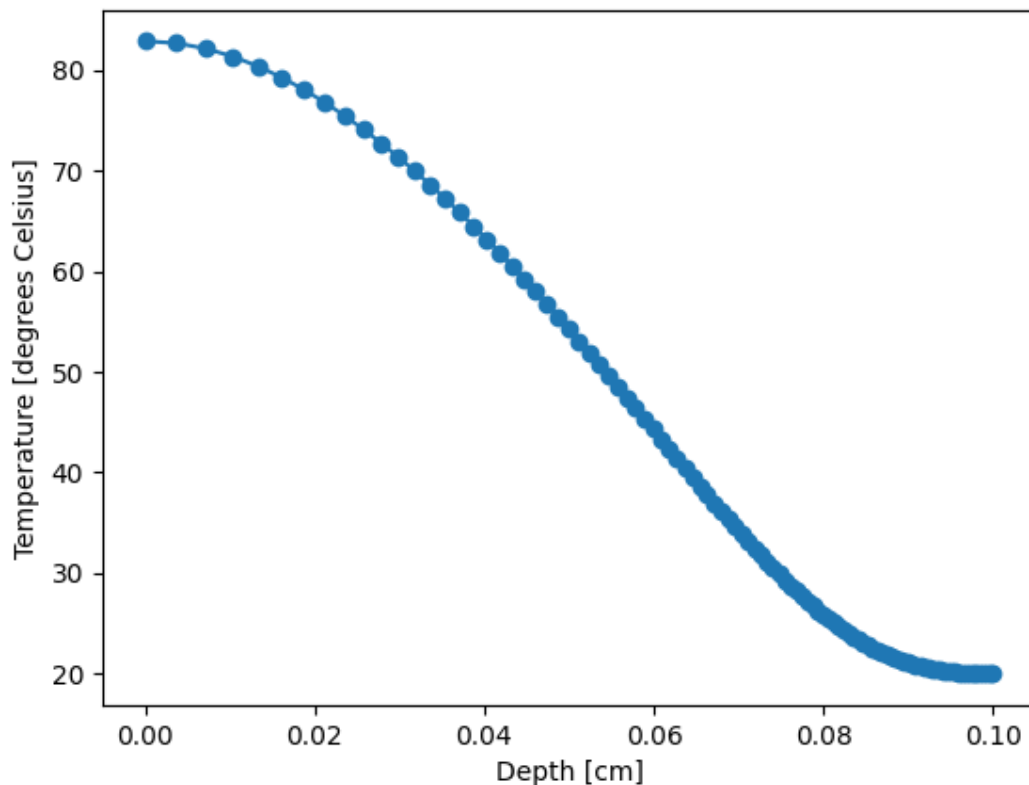


Figure 3.29: Result from simulation of the Application Model with the FEniCS library

3.3.3.1 Limitations of the Application Model

In the Application Model, the material properties are set constant throughout the total simulation and it does not consider magnetic properties. In the real material, most of the material properties will change as it gets heated. The resistivity in the material will increase as the temperature increase, causing higher Joule heating and thus a higher temperature in the material. As the temperature increase, the specific heat capacity will also change. The amount of energy required for the material to increase the temperature by 1 K will decrease as it is heated, resulting in a higher temperature rise. The change of these properties will have a more notable effect when the temperature reaches around 100 °C. After this temperature range, the temperature rise will become more nonlinear due to these effects. However, in the 20-100 °C temperature range, the material properties do not have a noticeable change regarding nonmagnetic materials. The temperature rise can then be said to be approximately linear. The reason for this is that the nonmagnetic material has material properties that are more resistant to heat, such as larger heat capacity and lower thermal conductivity. The Application Model is therefore valid in the temperature range up to 100 °C compared with heating of a non-magnetic material.

The design of the heating coil has also an effect on the temperature distribution in the material. When a current run in the coil, a magnetic field is generated. The strength of this field is derived from Amperes law for a long solenoid [33]:

$$\oint H ds = \frac{N \cdot I}{L} \quad (3.13)$$

- H: Magnetic field strength [A/m]
- N: Number of turns of the coil
- I: Current through the coil [A]
- L: Length of coil [m]

The magnetic field enclosed by the integration path is the sum of the number of turns and current per unit length. With a longer coil, the magnetic field strength will be weaker than by a shorter coil. The current density will be more intense with a shorter coil, causing larger temperature differences between the segments outside that are not affected by the coils magnetic field. A shorter coil will be more difficult to model since the magnetic field lines has different curvatures as the field distances from the coil. The Application Model assumes an infinitely long coil with a uniform magnetic field in all segments. This will result in magnetic field lines with similar intensity along the coil. The edges will normally have the highest current density since the magnetic fields will be the densest in this area, causing a stronger magnetic field. Since it is one dimensional, it only calculates the temperature distribution from the surface of the workpiece and inwards.

3.4 Converter Model

The Converter Model will be a model of the frequency converter and adjust the frequency for the application to be tested. The real converter will be set at a voltage level at a given frequency and then be tuned until it achieves resonance. The current that can run in the coil is determined by the available voltage in the output circuit and total resistance. By adjusting the output voltage, the current in the coil is set as close to the set point current by considering the limitations. If the total resistance becomes too high, the current will be adjusted by the available output voltage and total resistance. It will also consider the total power output.

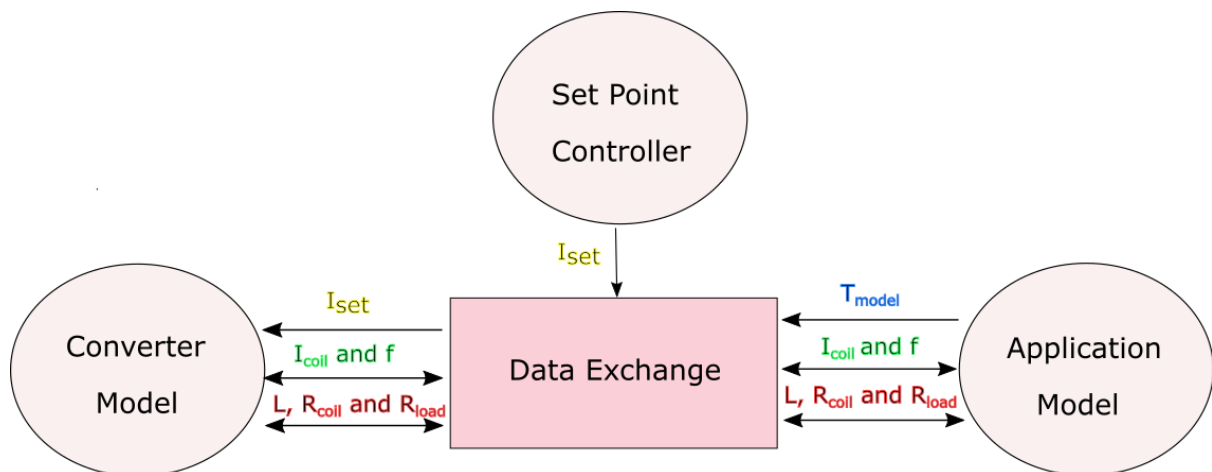


Figure 3.30: Illustration of the communication between the Converter and Application model

Figure 3.30 illustrates the communication between the Converter and Application Model through MQTT. At first, the Converter Model receives a set point current, I_{set} , and a power ON signal from the Set Point Controller. The Converter Model publishes a test current and frequency to the Application Model, which will recalculate with the new values. The inductance and resistance of the workpiece to be tested will be updated, and published back to the Converter Model, in addition to the total resistance. After recalculating in the Converter Model, the maximum current in the coil and frequency is published back to the Application Model.

3.4.1 Introduction to the Converter Model

3 Implementation of the Digital Twin

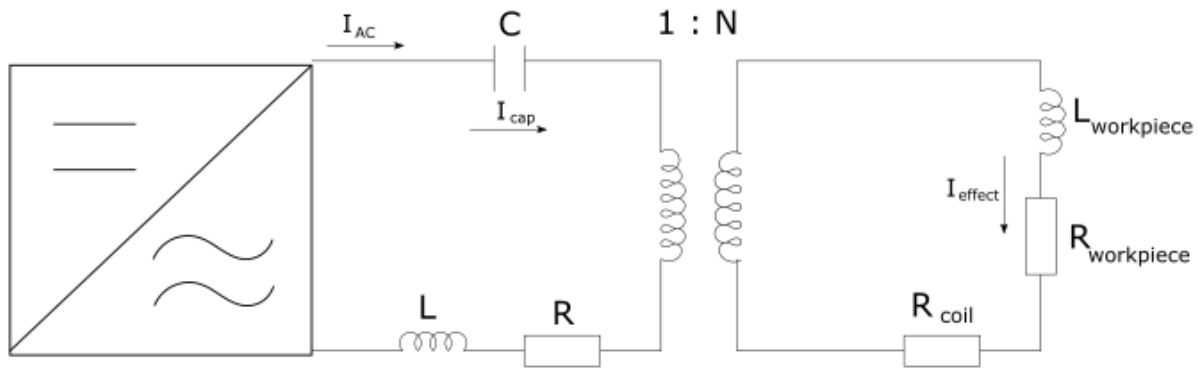


Figure 3.31: Output circuit of the converter

Figure 3.31 illustrates the output circuit of the converter. The output circuit is an LRC circuit, with capacitor, inductor, and resistor in series. The purpose of this is to achieve resonance, where the total impedance is at its lowest and current at its maximum. The inductive and capacitive reactance depends on the frequency. As shown in Equations (3.14) and (3.15), by increasing the frequency, the inductive reactance will proportionally increase. While the capacitive reactance will decrease inversely proportional as the frequency increase.

$$X_L = 2\pi fL \quad (3.14)$$

$$X_C = \frac{1}{2\pi fC} \quad (3.15)$$

At the point where the inductive and capacitive reactance are equal, resonance occur. There will be no phase difference between voltage and current since the capacitor and inductor cancel each other out. The voltage from the converter will then be the same as the voltage over the workpiece. The frequency when resonance occur can then be derived as in Equation (3.16).

$$X_L = X_C \rightarrow 2\pi fL = \frac{1}{2\pi fC} \rightarrow f = \frac{1}{2\pi\sqrt{LC}} \quad (3.16)$$

The total impedance will be at its minimum with only resistance in the circuit, which will give the maximum current at the given voltage [34].

There are several limitations on how high the current can be set to run in the coil. In the script there are three currents that must be calculated:

- Maximum current through capacitor, I_c
- Maximum current regarding power, I_{effect}
- Maximum current from the converter based on the voltage, I_{AC}

The maximum current flowing through the capacitor will be the source voltage over the capacitive reactance:

3 Implementation of the Digital Twin

$$I_c = \frac{V_{AC}}{X_C} \quad (3.17)$$

To avoid damage to the capacitor, the voltage over the component cannot exceed a certain limit. The capacitor can handle voltage peaks, but a consistent higher voltage will cause failure and must be considered when calculating the maximum current in the coil.

The maximum current regarding power will be the squared of the total resistance in the circuit over the power:

$$I_{effect} = \sqrt{\frac{R_{total}}{P}} \quad (3.18)$$

And the maximum output current from the converter will be the source voltage over the total resistance in the circuit:

$$I_{AC} = \frac{V_{AC}}{R_{total}} \quad (3.19)$$

The maximum current that can run in the coil, must be the minimum of the three currents calculated to avoid not exceeding any limitations. Before the current is to run in the coil, it must be transformed by multiplying with the transformer ratio 'N'.

3.4.2 Converter Model with Python

Same as with the Application Model, the Converter Model is built in the Spyder environment using Python as the programming language. The Paho Client library for MQTT is implemented in the model and variables are defined as in Figure 3.32.

```
16 broker = '192.168.10.129'
17 port = 1883
18 # topic to subscribe messages from
19 subscribe_topics = [("appmodel/resistance",0), ("appmodel/inductance",0),
20                    ("setpoint/power/convmodel",0), ("setpoint/current",0),
21                    ("setpoint/mqtt",0)]
22 # topics to publish messages to
23 publish_topic_current = "convmodel/current"
24 publish_topic_frequency = "convmodel/frequency"
25 publish_topic_power = "convmodel/power"
26 # generate client ID
27 client_id = f'python-mqtt-{random.randint(0, 100)}'
```

Figure 3.32: Definitions of the variables for Paho client library in the Converter Model

The Converter Model will subscribe to the topics 'setpoint/power/convmodel' and 'setpoint/current' and receive messages published by the Set Point Controller. Messages received from topics 'appmodel/resistance' and 'appmodel/inductance' are published from the Application Model. After calculations, The Converter Model will then publish to topics 'convmodel/current' and 'convmodel/frequency' subscribed in the Application Model.

3 Implementation of the Digital Twin

```
38 def on_connect(client, userdata, flags, rc):
39     if rc == 0:
40         print("Connected to MQTT Broker!")
41         client.subscribe(subscribe_topics)
42     else:
43         print("Failed to connect, return code %d\n", rc)
44
45 # Handling received messages from topics
46
47 def on_message(client, userdata, msg):
48     if msg.topic == "setpoint/power/convmodel":
49         print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
50         C.Power_on = int(msg.payload.decode())
51         if C.Power_on == 1:
52             print("Power is on")
53         else:
54             print("Power is off")
55
56     if msg.topic == "setpoint/current" and C.Power_on == 1:
57         print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
58         C.Iset = float(msg.payload.decode())*C.N
59         C.Calc()
60
61     if msg.topic == "appmodel/resistance" and C.Power_on == 1:
62         print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
63         C.Rtotal = float(msg.payload.decode())
64         C.Calc()
65
66     if msg.topic == "appmodel/inductance" and C.Power_on == 1:
67         print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
68         C.L = float(msg.payload.decode())
69         C.Calc()
```

Figure 3.33: Using functions from Paho Client library for connecting and subscribing to topics

Values that are received through the defined topics are handled as shown in Figure 3.33. To start the calculations, the Converter Model must first receive a power on signal from the Set Point Controller. Together with the received value from the mentioned topics, the power on signal must be set as 1 before the received value is updated as the new value in the script and the calculations can be executed.

3 Implementation of the Digital Twin

```
99 def Calc(self):
100     self.F = 1/(2*pi*sqrt(C.L/self.N*self.C)) # Resonance frequency
101     self.Xc = 1/2*pi*self.F*self.C # Reactance in capacitor
102     Ic = self.Vac/self.Xc # Current through capacitor
103     Ieffect = sqrt(self.P/C.Rtotal) # Total effect
104     Iac = self.Vac/C.Rtotal # Current from the converter
105     Current = [Ic, Ieffect, Iac]
106     self.Icoil = min(Current)*self.N # Max current in coil
107     self.Vac = self.Icoil*C.Rtotal # Voltage from converter
108     self.Vc = self.Icoil*self.Xc # Voltage across capacitor
109     if self.Icoil > self.Iset:
110         self.Pmax = self.Iset**2*C.Rtotal # Max effect with coil current
111         client.publish(publish_topic_current, self.Iset*self.N)
112         client.publish(publish_topic_frequency, self.F)
113         client.publish(publish_topic_power, self.Pmax)
114         print('Icoil: ', self.Iset)
115         print('Frequency: ', self.F)
116         time.sleep(1)
117     else:
118         self.Pmax = self.Icoil**2*C.Rtotal # Max effect with coil current
119         client.publish(publish_topic_current, self.Icoil)
120         client.publish(publish_topic_frequency, self.F)
121         client.publish(publish_topic_power, self.Pmax)
122         print('Icoil: ', self.Icoil)
123         print('Frequency: ', self.F)
124         time.sleep(1)
```

Figure 3.34: Function for calculations in the Converter Model

The function for calculation is shown in Figure 3.34. The capacitance 'C' and transformer ratio 'N' are parameters from the converter, while the inductance 'L' and resistance 'R' is received from the Application Model. 'L' must be transformed to primary value by 'N' when calculating the resonance frequency. From the received values, the three different currents are calculated, including the frequency and power. The minimum current of those that are calculated, is the maximum current that can run in the coil. If the setpoint current is lower than the maximum current, this will be set as the current to run in the coil since it will not cause any conflict with the limitations. Together with the calculated frequency, these values will be published to the topics defined in Figure 3.32.

It is also of interested to calculate the total power that will be generated by the applied current to compare with the actual converter.

3.5 Set Point Controller with C#

The Set Point Controller will be the user interface for the customer. This controller will deliver settings for the application to be tested and display the results. To build this model, Microsoft Visual Studios with C# is used. The purpose of this is to make a GUI (Graphic User Interface) to make it more user-friendly for a potential customer.

3.5.1 Set Point Controller

The first version of the Set Point Controller is shown in Figure 3.35. The settings the user insert in the textboxes will be publish to the topics of interest and received by the other models through Data Exchange. For the GUI to be able to have communication with the other clients, a MQTT client library must be implemented in the program. M2MQTT is a client library that works well with C# and is used for this model [35].

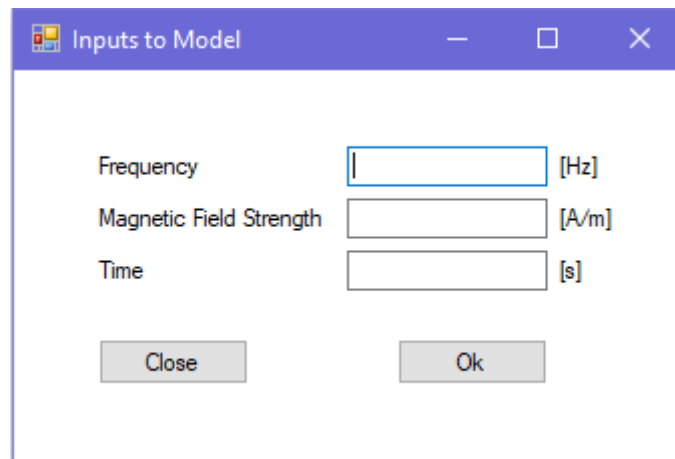


Figure 3.35: GUI for inputs to the model

The M2MQTT client library must be installed through the NuGet Package Manager in Visual Studios to get access to the tools, shown in Figure 3.36.

```
using uPLibrary.Networking.M2Mqtt;
using uPLibrary.Networking.M2Mqtt.Messages;
```

Figure 3.36: M2MQTT client library in Visual Studios

Figure 3.37 show the functions for MQTT connection in Visual Studios. For each time the model starts, a unique client ID is generated and connected to the broker.

3 Implementation of the Digital Twin

```
2 references
public partial class PublishAppModel : Form
{
    MqttClient client;
    string clientId;

    0 references
    public PublishAppModel()
    {
        InitializeComponent();

        string BrokerAddress = "test.mosquitto.org";

        client = new MqttClient(BrokerAddress);

        // a unique clientId for each time the application start
        clientId = Guid.NewGuid().ToString();

        client.Connect(clientId);
    }
}
```

Figure 3.37: Functions in the M2MQTT library for MQTT connection in Visual Studios

This model will only publish messages to the Application Model and will not subscribe to any topics. Since there are three values to publish, there must be three publish functions with three corresponding topics for the values. The functions for publishing the settings is shown in Figure 3.38.

```
1 reference
private void OkButton_Click(object sender, EventArgs e)
{
    if (Frequency.Text != "")
    {
        // whole topic
        string Topic_frequency = "frequency";

        // publish a message with QoS 0
        client.Publish(Topic_frequency, Encoding.UTF8.GetBytes(Frequency.Text), MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE, true);
    }

    if (FieldStrength.Text != "")
    {
        // whole topic
        string Topic_fieldstrength = "fieldstrength";

        // publish a message with QoS 0
        client.Publish(Topic_fieldstrength, Encoding.UTF8.GetBytes(FieldStrength.Text), MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE, true);
    }

    if (Time.Text != "")
    {
        // whole topic
        string Topic_time = "time";

        // publish a message with QoS 0
        client.Publish(Topic_time, Encoding.UTF8.GetBytes(Time.Text), MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE, true);
    }
}
```

Figure 3.38: Functions in the M2MQTT library for publishing messages

When the 'Ok' button is clicked, these functions will be executed. The values that are inserted in the textboxes, is published to the corresponding topics, and subscribed to in the Application Model.

3.5.2 Set Point Controller with Timer

In the real application the converter will run in a sequence with the power going on and off. To develop the Set Point Controller, two timers is implemented in the model. One timer retains the power on with the set point current for a given interval. After the interval, a second timer is enabled. The second timer retains the power off and current at zero. When the second interval is completed, the first timer is enabled again. The switching between these two timers continue until the power button is clicked to stop both the timers. The GUI application is shown in Figure 3.39.

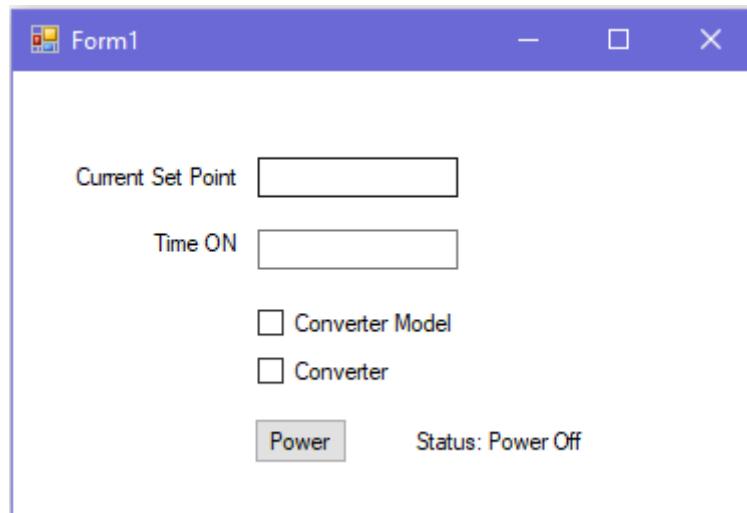


Figure 3.39: GUI of the Set Point Controller with timers implemented.

The Power button is a checkbox, appearing as a button. When the button is checked, a function to start the sequence is activated as shown in Figure 3.40.

```

1 reference
private void Power_CheckedChanged(object sender, EventArgs e)
{
    if (PowerButton.Checked)
    {
        status.Text = "Status: Power On";
        timer.Enabled = true;
        timer.Interval = 1000; // 1 second
        timer.Tick += new EventHandler(timer_Tick);
    }
    else
    {
        status.Text = "Status: Power Off";
        timer.Enabled = false;
        timer1.Enabled = false;
        string Topic_power_convmodel = "setpoint/power/convmodel";
        string Topic_power_converter = "setpoint/power/converter";
        string Topic_current = "setpoint/current";
        string Power = Convert.ToString(0);
        string Current = Convert.ToString(0);

        client.Publish(Topic_power_convmodel, Encoding.UTF8.GetBytes(Power), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, true);
        client.Publish(Topic_power_converter, Encoding.UTF8.GetBytes(Power), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, true);
        client.Publish(Topic_current, Encoding.UTF8.GetBytes(Current), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, true);
    }
}

```

Figure 3.40: Function behind Power button in the GUI application

3 Implementation of the Digital Twin

The first timer is enabled and ticks an event handler which will execute a function after the interval is completed. This event handler is shown Figure 3.41, with an if-statement for the Converter Model checkbox. The Converter checkbox has also the same if-statement.

```
2 references
private void timer_Tick(object sender, EventArgs e)
{
    if (ConverterModel.Checked)
    {
        status.Text = "Status: Power On";
        string Topic_power = "setpoint/power/convmodel";
        string Topic_current = "setpoint/current";
        string Power = Convert.ToString(1);
        client.Publish(Topic_power, Encoding.UTF8.GetBytes(Power), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, true);
        client.Publish(Topic_current, Encoding.UTF8.GetBytes(Current.Text), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, true);

        timer.Enabled = false;
        timer1.Enabled = true;
        timer1.Interval = Convert.ToInt32(time.Text) * 1000; // in seconds
        timer1.Tick += new EventHandler(timer1_Tick);
    }
}
```

Figure 3.41: Event handler when the first timer is enabled in the GUI application

If one checkbox, or both, is checked, the Set Point Controller will publish a power on signal and current to the subscribers. After the interval, the second timer is enabled. The time entered in the ‘Time ON’ textbox, will be set as the interval for when the second timer will be enabled. This will tick the next event handler, shown in Figure 3.42.

```
3 references
private void timer1_Tick(object sender, EventArgs e)
{
    status.Text = "Status: Power Off";
    string Topic_power_convmodel = "setpoint/power/convmodel";
    string Topic_power_converter = "setpoint/power/converter";
    string Topic_current = "setpoint/current";
    string Power = Convert.ToString(0);
    string Current = Convert.ToString(0);

    client.Publish(Topic_power_convmodel, Encoding.UTF8.GetBytes(Power), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, true);
    client.Publish(Topic_power_converter, Encoding.UTF8.GetBytes(Power), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, true);
    client.Publish(Topic_current, Encoding.UTF8.GetBytes(Current), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, true);

    timer1.Enabled = false;
    timer.Enabled = true;
}
```

Figure 3.42: Event handler when the second timer is enabled in the GUI application

The Set Point Controller will publish a power off signal and current with value zero to the subscribers. After this is executed, the first timer will be reenabled and the sequence will start over again. This will continue until the Power button is checked again, and the else-statement shown in Figure 3.40 will be executed.

An example of the application is shown in Figure 3.43. The label ‘Status’ indicates what power signal is sent to the subscribers.

3 Implementation of the Digital Twin

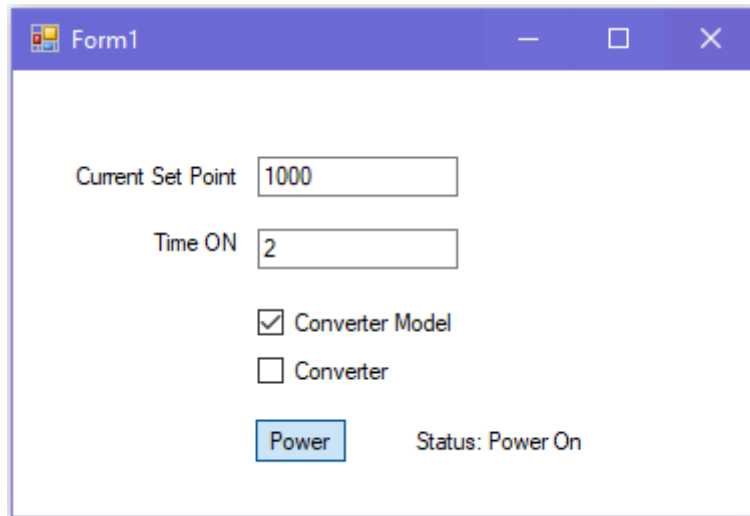


Figure 3.43: Start of sequence from the Set Point Controller

3.5.3 Set Point Controller with Display

It is of interest to display the results when testing both the digital twin and the real application for comparison, as seen in Figure 3.44.

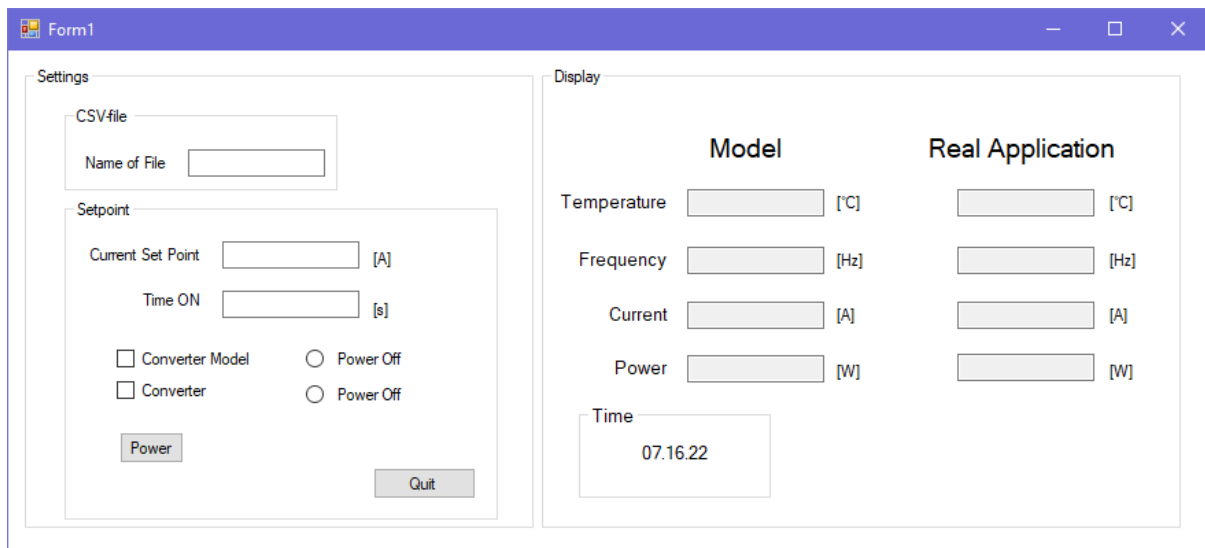


Figure 3.44: Set Point Controller including display of parameters

The Set Point Controller must then subscribe to the parameters from the real application and the digital twin, as showed in Figure 3.45. To store the data from the testing, the name of the file to write is also included and published to the Data Storage model.

3 Implementation of the Digital Twin

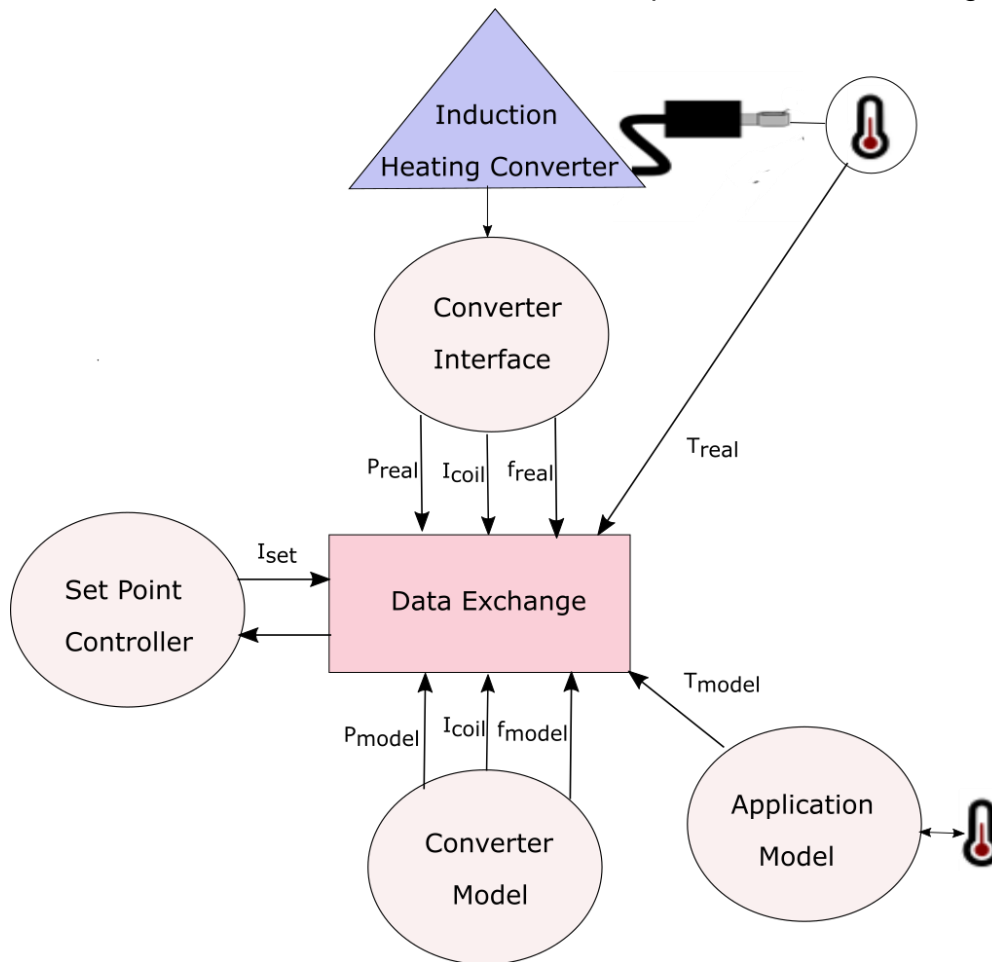


Figure 3.45: Information flow to the Set Point Controller through the Data Exchange

Figure 3.46 show the function for handling received messages from a selection of topics subscribed in the Set Point Controller, here the topics for receiving temperatures from the Application Model and the real application. The message associated with the topic is displayed in the corresponding textbox in the GUI.

```

1 reference
public void client_MqttMsgPublishReceived(object sender, MqttMsgPublishEventArgs e)
{
    if (this.InvokeRequired)
    {
        if (e.Topic == "appmodel/temp")
        {
            this.Invoke(new Action(() => this.DisplayTemperatureModel1.Text = Encoding.UTF8.GetString(e.Message)));
        }

        if (e.Topic == "real/temp")
        {
            this.Invoke(new Action(() => this.DisplayTemperature.Text = Encoding.UTF8.GetString(e.Message)));
        }
    }
}

```

Figure 3.46: Function for handling received messages to the Set Point Controller

3 Implementation of the Digital Twin

The report enabling and name of file is handled as shown in Figure 3.47. When the ‘Power’ button in the GUI is checked, the name of the file and signal to report is published. The Data Storage model will subscribe to these topics and receive the signal and name. It is important that the name of the file when entered in the textbox ends with ‘.csv’, in order that the Data Storage model handles it as a CSV-file.

```
if (File.Text != "")
{
    string Topic_file = "setpoint/file";
    string Topic_report = "setpoint/report";
    string Report = Convert.ToString(1);
    client.Publish(Topic_file, Encoding.UTF8.GetBytes(File.Text), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, false);
    client.Publish(Topic_report, Encoding.UTF8.GetBytes(Report), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, false);
}
```

Figure 3.47: If-statement for report handling to the Data Storage model

A ‘Quit’ button is also included in the GUI. When this button is clicked, the Set Point Controller will publish a message to the subscribing models connected through the Data Exchange to end the simulation, and close the application, seen in Figure 3.48.

```
1 reference
private void Quit_Click(object sender, EventArgs e)
{
    string Topic_MQTT = "setpoint/mqtt";
    string MQTT = Convert.ToString(0);
    client.Publish(Topic_MQTT, Encoding.UTF8.GetBytes(MQTT), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, false);

    string Topic_report = "setpoint/report";
    string Report = Convert.ToString(0);
    client.Publish(Topic_report, Encoding.UTF8.GetBytes(Report), MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE, false);

    System.Windows.Forms.Application.Exit();
}
```

Figure 3.48: Function for 'Quit' button clicked

3.6 Model Parameter Tuner

It has not been developed an own model for Model Parameter Tuner in this thesis. It would be ideal to have a model parameter tuner for adjusting the parameters according to the application to be tested. A simplification is to include a factor for e.g., adjusting the magnetic field strength according to the coil used for the application. With a factor equal 1, the magnetic field strength would be according to a long coil. By reducing this factor to adapt to the application, the magnetic field is reduced, and less heat is dissipated.

3.7 Data Storage

It is desirable to store the data that are produced from the simulations. This can be achieved by either using a database or log the data in a CSV (Comma Separated Values) file. MQTT broker do not have a mechanism to store data that are distributed through it, but it is possible to use a MQTT client as a storage by subscribing to topics and receiving the data [36].

Ideally an SQL (Structured Query Language) database would be used for data storage. SQL is a language that are designed to work and communicate with databases. It can either be SQL or NoSQL, depending on the application. Difference between SQL and NoSQL is the

3 Implementation of the Digital Twin

system of storing data. SQL is used for relational databases, where the data is stored in tables. The tables have key elements that links them together to avoid duplications [37]. While NoSQL is non-relational and object-based database, and there is no fixed system of storing the data. NoSQL is more dynamic, where different types of unstructured data can be stored in the database [38]. For the development of the digital twin, NoSQL would be the preferred database to use for data storage. The reason behind is that NoSQL is better suited for real time and fast data, which is important for the digital twin.

A simplification of a data storage is having a model that logs data in a CSV file. CSV file is a text file that separate values by commas. The file writes one line for each data record and is table-based [39]. The CSV file is compatible with Microsoft Excel and can be converted to a spreadsheet. This is an easy method for organizing and storing the data that are flowing through the Data Exchange.

3.7.1 Data Storage Model with Python

The Data Storage Model is built in the Spyder environment using Python as the programming language. The Paho MQTT Client library is implemented in the script to receive the values that are published from the other models. Figure 3.49 show a snippet of the subscribing topics that receive values to be handled by the Data Storage Model.

```
76     if msg.topic == "appmodel/temp":
77         print(f"Received `{msg.payload.decode()}` from `{msg.topic}` topic")
78         T = float(msg.payload.decode())
79         if I.T != T:
80             I.T = T
81             I.WritetoFileOnChange()
82
83
84     if msg.topic == "convmodel/current":
85         print(f"Received `{msg.payload.decode()}` from `{msg.topic}`")
86         Icoil = float(msg.payload.decode())
87         if I.Icoil != Icoil:
88             I.Icoil = Icoil
89             I.WritetoFileOnChange()
90
91
92     if msg.topic == "convmodel/power":
93         print(f"Received `{msg.payload.decode()}` from `{msg.topic}`")
94         P = float(msg.payload.decode())
95         if I.P != P:
96             I.P = P
97             I.WritetoFileOnChange()
```

Figure 3.49: Snippet of messages that are handled through MQTT in the Data Storage Model

All the parameters that are flowing through the Data Exchange is received in the Data Storage Model and is defined in the 'DataStorage' class, seen in Figure 3.50.

3 Implementation of the Digital Twin

```
177 class DataStorage:
178     def __init__(self, Iset, Icoil, Power_on, Time, Rtotal, L, T, F, P, Treal,
179                 Freal, Preal, Ireal, File_name, mqtt_off):
180         self.Iset = Iset
181         self.Icoil = Icoil
182         self.Power_on_conv = 0
183         self.Power_on_convmodel = 0
184         self.Time = Time
185         self.Rtotal = Rtotal
186         self.L = L
187         self.T = T
188         self.F = F
189         self.P = P
190         self.Treal = Treal
191         self.Freal = Freal
192         self.Preal = Preal
193         self.Ireal = Ireal
194         self.File_name = File_name
195         self.mqtt_off = mqtt_off
196         self.ReportOnChange = False # Set to true to report on change
197         self.ReportOnTime = True # Set to true to report on time interval
198         self.ReportEnable = False
199         self.file = 0
```

Figure 3.50: Defined parameters in class DataStorage that is to be stored in the model

The name of the file and signal for 'ReportEnable', is given from the Set Point Controller. When 'ReportEnable' is set to true, the function 'WriteFileOnTime' is enabled, shown in Figure 3.51. The header is written at the top and defines all the parameters for each column. For each time stamp, the time and value of the parameters are written to the CSV file on one line. In addition, if parameters are changed since the previous received value, the function 'WriteToFileOnChange' will be called.

```
200     def WriteFileOnTime(self):
201         if (self.ReportEnable):
202             print("WriteOnTime()")
203             Changed = datetime.now().strftime('%Y %m %d_%H:%M:%S:%f ')
204             self.file.write(str(Changed)+","+str(I.Power_on_conv)+","+
205                             str(I.Power_on_convmodel)+","+ str(I.Iset)+","+
206                             str(I.Treal)+","+ str(I.T)+","+ str(I.Preal)+","+
207                             str(I.P)+","+ str(I.Freal)+","+str(I.F)+","+
208                             str(I.Icoil)+","+str(I.Ireal)+"\n")
209     def WritetoFileOnChange(self):
210         if (self.ReportOnChange) and (self.ReportEnable):
211             Changed = datetime.now().strftime('%Y %m %d_%H:%M:%S:%f ')
212             self.file.write(str(Changed)+","+str(I.Power_on_conv)+","+
213                             str(I.Power_on_convmodel)+","+ str(I.Iset)+","+
214                             str(I.Treal)+","+ str(I.T)+","+ str(I.Preal)+","+
215                             str(I.P)+","+ str(I.Freal)+","+str(I.F)+","+
216                             str(I.Icoil)+","+str(I.Ireal)+"\n")
```

Figure 3.51: Functions for writing to file under class DataStorage

Figure 3.52 show the CSV file opened in Excel when parameters have been written to the file. Each parameter has its own column with the row being filled from each sample with a time stamp.

3 Implementation of the Digital Twin

Time	Power_on_conv	Power_on_convmodel	lset	Treal	T	Preal	P	Freal	F	Icoil	Ireal
2021 04 22 15:29:55:796022	0	0	0	26,5	-20	50	-1	49,8	-1	-1	0,3
2021 04 22 15:29:55:904503	1	1	60	26,5	-20	1675	-1	28,1	-1	-1	60,6
2021 04 22 15:29:56:014561	1	1	60	26,5	-20	1675	-1	28,1	-1	-1	60,45
2021 04 22 15:29:56:123897	1	1	60	26,5	-20	4700	-1	28,4	-1	-1	60,45
2021 04 22 15:29:56:233124	1	1	60	26,6	-20	6025	-1	28,4	-1	-1	60,45
2021 04 22 15:29:56:341713	1	1	60	26,8	-20	6025	-1	27,9	-1	-1	60,45
2021 04 22 15:29:56:451673	1	1	60	27	-20	6025	-1	28,1	-1	-1	60,45
2021 04 22 15:29:56:559935	1	1	60	27,3	-20	6025	-1	28,4	-1	-1	60,6
2021 04 22 15:29:56:667974	1	1	60	27,7	-20	6025	-1	28,1	-1	-1	60,45

Figure 3.52: Parameters written to the CSV file opened in Excel

3.8 Converter Interface with Raspberry Pi 3

To have connection to the real converter, Raspberry Pi 3 is included to function as the Converter Interface. With the add-on board, Automation HAT, inputs and outputs from the real converter can be connected to the channels of the board. The Raspberry Pi 3 will also act as the MQTT broker for the digital twin and ensure connection between all the models the digital twin consists of.

Appendix D and E will give a tutorial on how Raspberry Pi 3 is set up, both a complete set up and a headless setup. Explanation and tutorial for installing the Mosquitto software and Automation HAT board is given in this subchapter.

3.8.1 Complete Setup of the Raspberry Pi 3

Tutorial for a complete setup of the Raspberry Pi 3 is given in Appendix D.

3.8.2 Headless setup of the Raspberry Pi 3

Tutorial for a headless setup of the Raspberry Pi 3 is given in Appendix E. The Raspberry Pi 3 will be used with a headless setup for the Converter Interface.

3.8.3 Install MQTT on Raspberry Pi 3

Mosquitto is installed on the Raspberry Pi 3 since it will act as MQTT broker for the experiments. The Mosquitto software is available from the Raspbian repository and is installed on the Raspberry Pi by entering the command in command line:

```
'sudo apt-get install mosquitto'
```

It is also desirable for the Raspberry Pi 3 to be able to publish and subscribe to topics, and therefore the client software is also installed by the command:

```
'sudo apt-get install mosquitto-clients'
```

The Raspberry Pi 3 can now act as a broker by using its own IP address as host address and use the functions for publishing and subscribing to topics, as shown in Figure 3.53.

3 Implementation of the Digital Twin

The image shows two terminal windows side-by-side. The left window is titled 'Ledetekst - mosquito_sub -h 192.168.10.200 -t test' and shows a Windows command prompt where the user navigates to the 'mosquito' directory and runs 'mosquitto_sub -h 192.168.10.200 -t test', which outputs 'hello'. The right window is titled 'pi@raspberrypi ~' and shows an SSH session from the PC to the Raspberry Pi. It displays the SSH password prompt, system information for Linux raspberrypi 5.10.17-v7+, and the output of the 'mosquitto_pub -h 192.168.10.200 -t test -m hello' command, which outputs 'hello'.

Figure 3.53: MQTT communication between computer and Raspberry Pi 3

3.8.4 Raspberry Pi 3 with Automation HAT

Automation HAT (Hardware Attached on Top) is an add-on board for the Raspberry Pi, shown in Figure 3.54. This board is for monitoring and automation control and can be attached to the Raspberry Pi by fitting the board to the GPIO pins. Components included on the Automation HAT board is three relays, three ADC (analog to digital converter) inputs, three sinking outputs, three digital inputs and 18 LED indicators.



Figure 3.54: The Automation HAT board attached to the Raspberry Pi Model B+ board

Before installing the software for the Automation HAT, it is recommended to update the Raspbian operating system. By entering ‘`sudo apt update`’ in the command line, the list of packages that need upgrading is displayed. Entering ‘`sudo apt full-upgrade`’ will upgrade the packages.

Pimoroni has made a Python library for the Automation HAT and is installed through the URL link by typing[40]:

```
‘curl https://get.pimoroni.com/automationhat/ | bash’
```

After reboot, the Automation HAT board is ready for use.

3.8.4.1 Test of Automation HAT with MQTT

A wire is connected between the analog input and 5 V channel to measure the voltage, and publish the values using the Raspberry Pi 3 as a MQTT broker.

3 Implementation of the Digital Twin

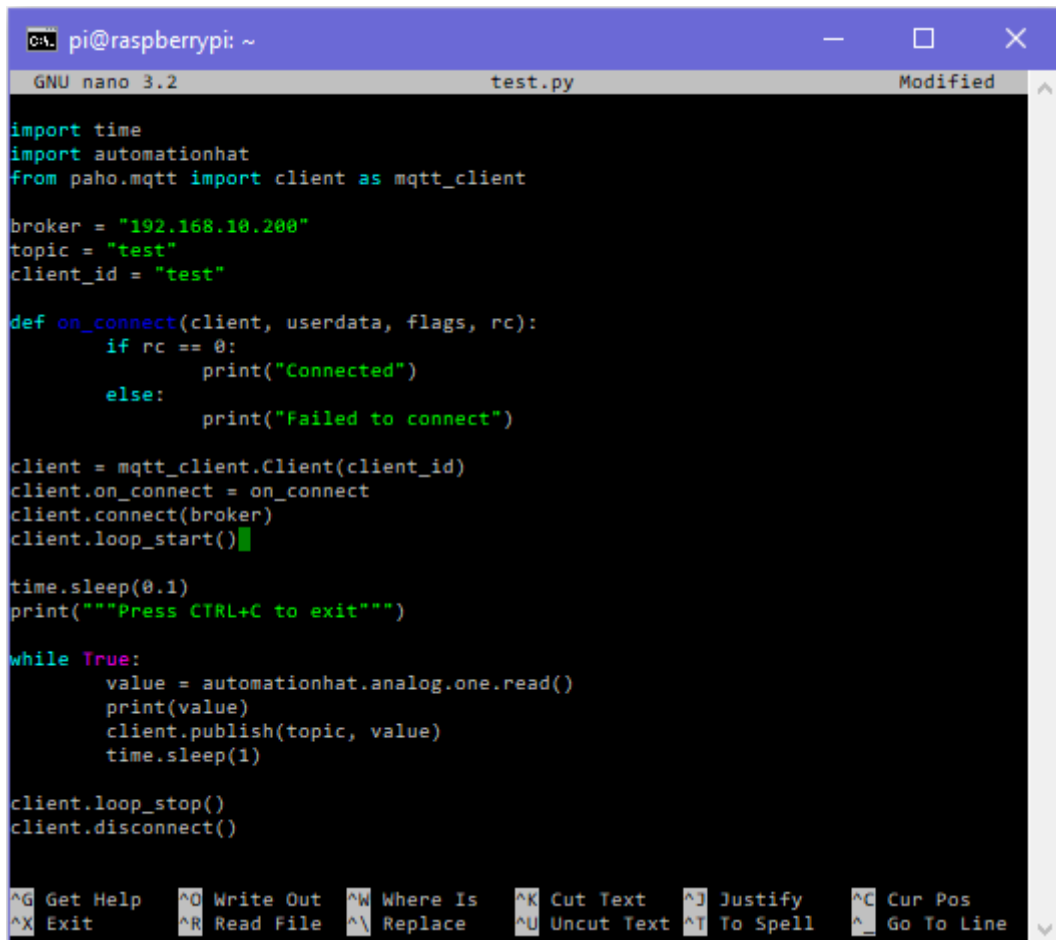


Figure 3.55: A wire connected between the analog input channel and 5 V channel on the Automation HAT board

To create a Python script on the Raspberry Pi 3, Nano editor is used. This is command-line editor which is necessary when the Raspberry Pi is set up headless. Nano is installed by default with the Raspbian operating system. The Python script for testing is created by typing:

```
'sudo nano test.py'
```

After the file is created, the Nano editor appear in the command window. The test script will read value from the first analog input channel and publish to the topic defined. Libraries included in the script is Paho MQTT Client and Automation HAT, seen in Figure 3.56.



```

pi@raspberrypi: ~
GNU nano 3.2      test.py      Modified
import time
import automationhat
from paho.mqtt import client as mqtt_client

broker = "192.168.10.200"
topic = "test"
client_id = "test"

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected")
    else:
        print("Failed to connect")

client = mqtt_client.Client(client_id)
client.on_connect = on_connect
client.connect(broker)
client.loop_start()

time.sleep(0.1)
print("""Press CTRL+C to exit""")

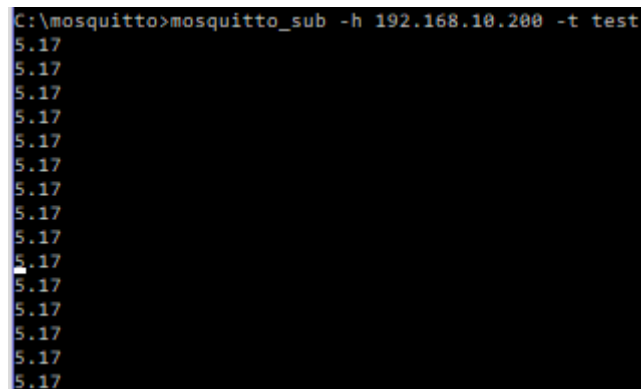
while True:
    value = automationhat.analog.one.read()
    print(value)
    client.publish(topic, value)
    time.sleep(1)

client.loop_stop()
client.disconnect()

```

Figure 3.56: Python script created in Nano editor with libraries for MQTT and Automation HAT included ‘value = automationhat.analog.one.read()’ is a function from the Automation HAT library which reads the value from the channel. The first channel is referenced to by ‘one’, and the other channels would be referenced by ‘two’ and ‘three’.

After saving the script, the Python script is executed by typing ‘python3 test.py’ in the command line. From the Windows command line, the Raspberry Pi’s IP address and topic to subscribe is defined, and the voltage measured from the channel is published, shown in Figure 3.57.



```

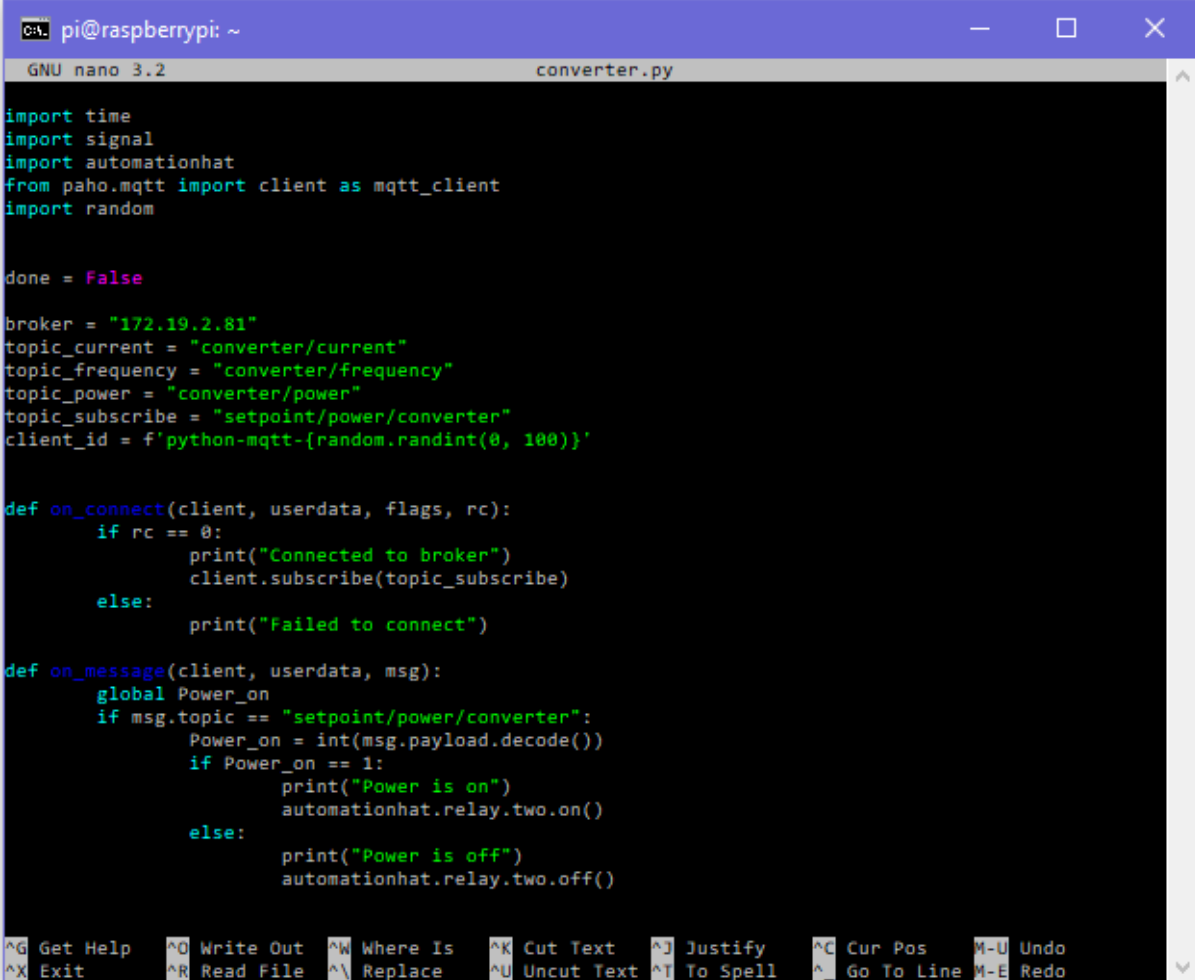
C:\mosquitto>mosquitto_sub -h 192.168.10.200 -t test
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17
5.17

```

Figure 3.57: Subscribing on topic ‘test’ in the Windows command line

3.8.5 Converter Interface for Digital Twin

The Raspberry Pi 3 will handle the power on signal from the Set Point Controller and read the current, frequency and power from the converter by the three analog inputs of the Automation HAT board. Based on the script, shown in Figure 3.57, it is developed to read on all channels for the converter, and controlling relay at power on signal.



```

pi@raspberrypi: ~
GNU nano 3.2 converter.py
import time
import signal
import automationhat
from paho.mqtt import client as mqtt_client
import random

done = False

broker = "172.19.2.81"
topic_current = "converter/current"
topic_frequency = "converter/frequency"
topic_power = "converter/power"
topic_subscribe = "setpoint/power/converter"
client_id = f'python-mqtt-{random.randint(0, 100)}'

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to broker")
        client.subscribe(topic_subscribe)
    else:
        print("Failed to connect")

def on_message(client, userdata, msg):
    global Power_on
    if msg.topic == "setpoint/power/converter":
        Power_on = int(msg.payload.decode())
        if Power_on == 1:
            print("Power is on")
            automationhat.relay.two.on()
        else:
            print("Power is off")
            automationhat.relay.two.off()

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos M-U Undo
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line M-E Redo

```

Figure 3.58: Python script in Raspberry Pi for controlling relay at power on signal

To handle the power on signal from the Set Point Controller, the relay on the Automation HAT board will turn on and close the circuit by the function ‘automationhat.relay.two.on()’, shown in Figure 3.58. Same with power off signal, the function ‘automationhat.relay.two.off()’ will turn off the relay and open the circuit.

Since the power of the converter normally will be off, the relay on the Automation HAT board will be connected to normally open and close when the power on signal is published.

3 Implementation of the Digital Twin

```
while (done != True):
    current = automationhat.analog.one.read()*15
    print("current: ", current)
    client.publish(topic_current, current)
    frequency = automationhat.analog.two.read()*10
    print("frequency: ", frequency)
    client.publish(topic_frequency, frequency)
    power = automationhat.analog.three.read()*2500
    print("power: ", power)
    client.publish(topic_power, power)
    time.sleep(0.1)
```

Figure 3.59: Python script in Raspberry Pi for reading analog inputs from the converter

Figure 3.59 show the functions for reading the analog input channels on the Automation HAT boards. In addition, the current, power and frequency must be scaled according to the real converter to replicate the actual values.

4 Testing of the Digital Twin of Induction Heating Equipment

In this chapter the testing of the induction heating equipment is compared with the digital twin. The preparations for the setup and equipment used in the experiments are explained, including heating system, coils, and workpieces. At last, the results from the experiments are displayed and compared with the results from the digital twin.

4.1 Introduction to the setup and experiments

The experiments will be executed in the Application lab at EFD Induction. By the Converter Interface with the Raspberry Pi 3 and Automation HAT board, this model will remotely control the real converter and deliver set point settings from the Set Point Controller. Real temperature measurements from the workpiece will be stored in the Data Storage model and be used to compare with temperature results from the Application Model.

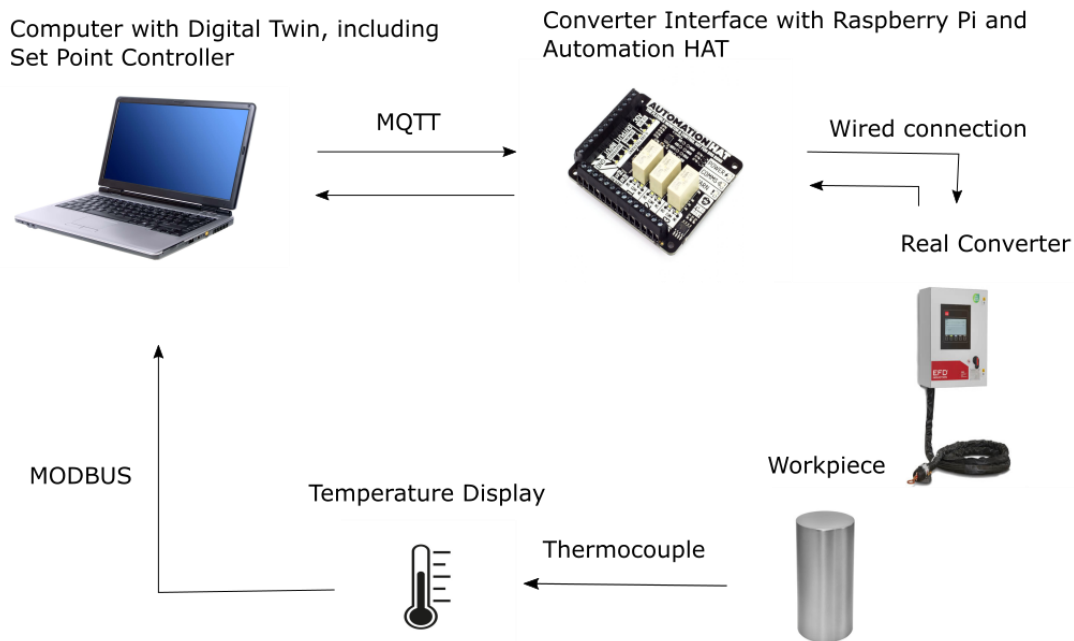


Figure 4.1: Illustration of the connection between the hardware, including communication protocols MQTT and MODBUS

The setup of the installation will be as illustrated in Figure 4.1. Settings for the experiment will be executed from the digital twin's Set Point Controller and published through MQTT. The connection to the real converter is enabled through the Converter Interface with Raspberry Pi 3 and the Automation HAT board with wired connection. To measure the temperature development, thermocouples are attached to the workpiece and displayed on a monitor. The temperature measurements from the monitor are sent to the computer by the

4 Testing of the Digital Twin of Induction Heating Equipment

communication protocol MODBUS. Equipment used in the installation are listed in Table 4.1 and will be explained in the coming subchapters.

Table 4.1: Equipment used in the experiments

Manufacturer	Product	Description
EFD Induction	Sinac SM 18/25 Twin	Heating System
Cooperheat	STORK Tau	Thermocouple Attachment Unit
Yokogawa	SMART DAC+ GP20	Temperature Display

4.2 Heating System

The heating system used for the experiments is the Sinac SM 18/25 Twin heating generator. This model is for medium frequency heating applications and have a maximum output power of 25 kW. Technical data for this model is given in Table 4.2.

Table 4.2: Technical data of Sinac 18/25 SM Twin

Model	Sinac SM 18/25 Twin
Maximum output power	25 kW
Continuous output power	18 kW
Frequency range	10-25 kHz

Inside this generator, a frequency converter and capacitor are included. By having the capacitor in series or parallel, the generator can either be serial- or parallel-compensated.

Setup of the generator can be done either local or remotely. By a touch screen, the setup can easily be done through the control panel, shown in Figure 4.2.

4 Testing of the Digital Twin of Induction Heating Equipment



Figure 4.2: Control panel of the Sinac SM 18/25 Twin

For the experiments, the setup will be controlled remotely through the second output.

4.3 Workpieces and Heating coils

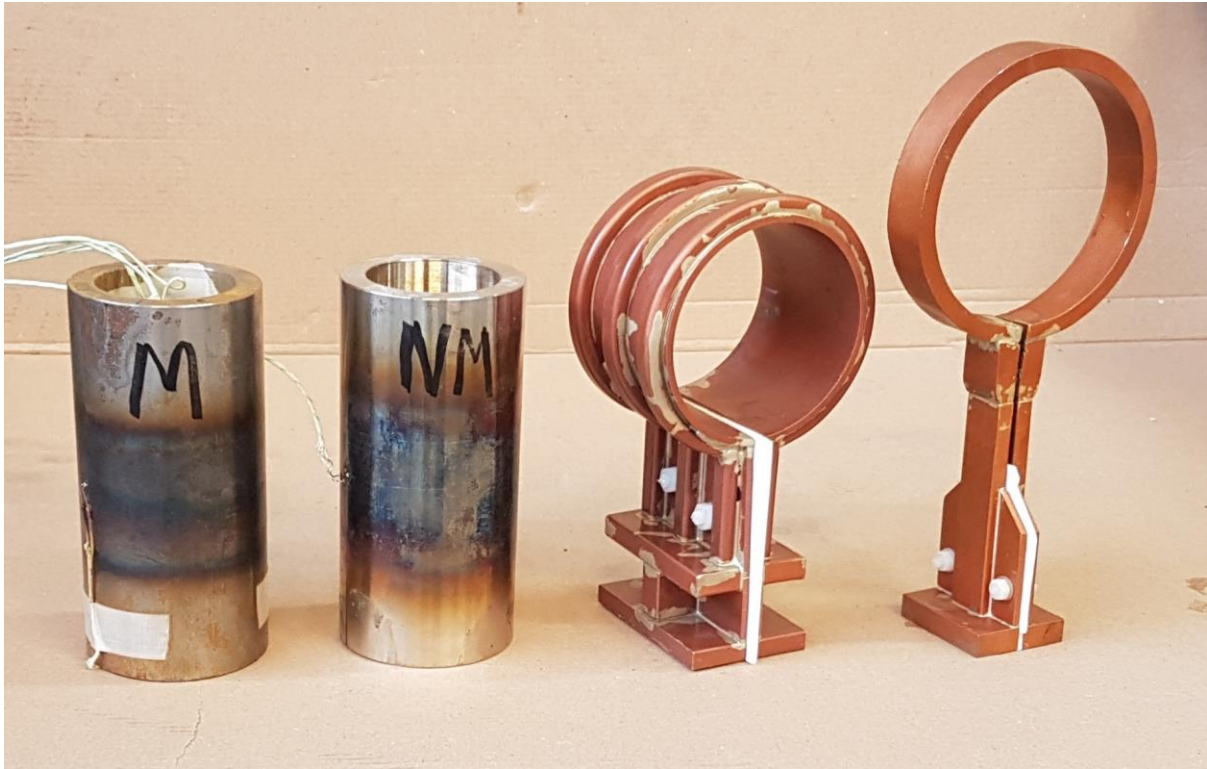


Figure 4.3: Workpieces of magnetic (M) and non-magnetic (NM) material, and long and short coil to be used in the experiments

4.3.1 Workpieces

Two hollow cylinders with the same proportions are used in the experiments as workpieces, shown to the left in Figure 4.3. The geometric data of the workpieces are given in Table 4.3.

Table 4.3: Geometric data of workpieces

Length [cm]	15
Outer radius [cm]	7.2
Inner radius [cm]	5.2

The difference between the workpieces are the material properties. Stainless steel 316L and 42CrMo4 are the two materials to be used in the experiments. None of these materials have linear temperature rise when heated up, but Stainless Steel 316L have a slower temperature rise than 42CrMo4 until 100 °C. Because of the slow temperature rise in this range, the rise can be said to be approximately linear. The Application Model can then be used for comparison in this range. With 42CrMo4 the temperature rise is higher and nonlinear, making the Application Model less good fit for comparison. Stainless Steel 31L will therefore

4 Testing of the Digital Twin of Induction Heating Equipment

be of most interest in the experiments. The comparison between Stainless Steel 316L and 42CrMo4 is given in Table 4.4.

Table 4.4: The material properties of 42CrMo4[41] and Stainless Steel 316L[42]

Material	Magnetic Properties	Specific Heat Capacity [J/(kg·K)]	Thermal Conductivity [W/m·K]	Resistivity [$\mu\Omega\cdot\text{m}$]	Density [kg/m ³]
Stainless Steel 316	Non-Magnetic	500 (0-100°C)	14	0.74	8000
42CrMo4 Steel	Magnetic	460-480 (50-100°C)	40-45	0.20	7800

Stainless Steel 316L have a higher heat capacity than 41CrMo4, meaning it requires more energy to increase the temperature by one degree. In addition, it has a lower thermal conductivity then 41CrMo4. This contributes to the lower temperature rise.

4.3.2 Heating coils

Two different coils are being used in the induction system, shown at the right side beside the workpieces in Figure 4.3. The first coil used in the experiments are the short coil with a smaller length. This coil will give a higher current density in a smaller area of the workpiece, causing a larger temperature difference between the different segments of the workpiece. The second coil has a longer length, shown also in Figure 4.4. This will cause a lower current density in the workpiece and a lower temperature increase. The heat will be distributed more evenly in the workpiece and a smaller temperature difference between the segments. The Application Model assume the workpiece is homogenous with the same temperature distribution inwards at every segment. The long coil will therefore be of most interest for the experiment since it is more applicable for the Application Model and give results that can be used for comparison.

4 Testing of the Digital Twin of Induction Heating Equipment



Figure 4.4: Long coil to be used in the experiments

4.4 Real-time data with MODBUS

STORK TAU (Thermocouple Attachment Unit) allows direct attachment to the workpiece by thermocouples. The thermocouples is placed at the outer surface of the workpieces and is attached by the capacitive discharge method and creates a welded connection [43]. To display the temperature, the thermocouples is connected to SMART DAC + GP20. This is a recorder that display the data through its channels. SMART DAC+ GP20 allows communication through the MODBUS TCP protocol by Ethernet connection. MODBUS TCP is a communication protocol based on the client/server model. This protocol is widely used in communication between automation equipment [44]. SMART DAC+ GP20 will act as the server, while the computer will be the client to request information. One requirement is that both server and clients are in the same IP network. To enable this communication, Python has a MODBUS client library named pyModbusTCP which is installed by 'pip install PyModbusTCP' [45]. This library is implemented in the script, shown in Figure 4.5.

4 Testing of the Digital Twin of Induction Heating Equipment

```
8 from pyModbusTCP.client import ModbusClient
9 import time
10 import random
11 from paho.mqtt import client as mqtt_client
12
13
14 broker = '172.19.2.81'
15 publish_topic_temp = "real/temp"
16 client_id = f'python-mqtt-{random.randint(0, 100)}'
17
18 def on_connect(client, userdata, flags, rc):
19     if rc == 0:
20         print("Connected to MQTT Broker!")
21     else:
22         print("Failed to connect, return code %d\n", rc)
23
24 client = mqtt_client.Client(client_id)
25 client.on_connect = on_connect
26 client.connect(broker)
27 client.loop_start()
28
29 c = ModbusClient(host="172.19.2.61", port=502, unit_id=1, auto_open=True)
30
31 while True:
32     regs = int(c.read_holding_registers(2, 1)[0])/10.0
33     time.sleep(0.1)
34     print(regs)
35     client.publish(publish_topic_temp, regs)
```

Figure 4.5: Python script with MODBUS client library implemented

The function 'ModbusClient' has the IP address of the server defined, which is the SMART DAC+ GP20. Port 502 is the default port for connection to the client. Auto open is set to 'True', to keep the connection always open and not close after each request. 'c.read_holding_registers(2, 1)' is the function for reading values from the SMART DAC+ GP20 at channel 2 and one temperature value [46]. The temperature values will be published every 0.1s by MQTT to the Set Point Controller and be displayed and monitored at the interface.

4.5 Preparations before experiment

The main challenge of the setup is the need of an analog output from the Automation HAT board on the Raspberry Pi 3 to the converter. When the power on signal is sent to the converter, it will also need a set point current for the application. The Automation HAT board does not have an analog output, but it is possible to use the GPIO pins of the Raspberry Pi 3. Since the GPIO pins is required for the Automation HAT board, a second Raspberry Pi 3 is setup to have available GPIO pins for the analog output, shown in Figure 4.6. This will be in a separate circuit from the other Raspberry Pi 3 and will be connected directly to the real converter. Another possibility is to set the set point current on the converter manually before the power on signal is sent.

4 Testing of the Digital Twin of Induction Heating Equipment

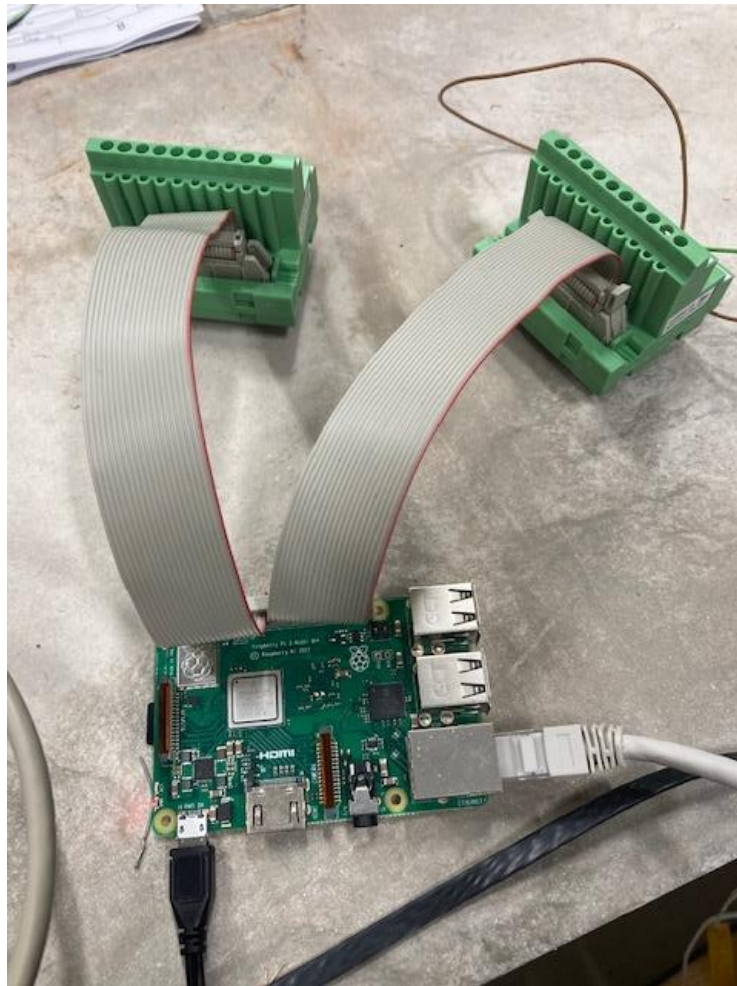


Figure 4.6: Raspberry 3 Model B+ board for analog output

To send the set point current from the Raspberry Pi 3, the signal needs to be converted to an analog signal by PWM. The Raspberry Pi 3 Model B+ board shown in Figure 4.6 has available GPIO pins for PWM. RPi.GPIO is a Python library which allows the GPIO pins on the Raspberry Pi 3 board to be controlled and is installed by default with the Raspbian operating system [47].

```
8 import RPi.GPIO as GPIO
9
10 OUTPUT_1 = 12
11
12 GPIO.setmode(GPIO.BCM)
13 GPIO.setup(OUTPUT_1, GPIO.OUT)
14
15 pwm = GPIO.PWM(OUTPUT_1, 25000)
16
17 pwm.start(50)
```

Figure 4.7: Python library for the Raspberry Pi GPIO pins

4 Testing of the Digital Twin of Induction Heating Equipment

Figure 4.7 show the Python script for controlling the GPIO pins of the Raspberry Pi 3. Pin 12 is the channel on the Raspberry Pi 3 GPIO pins, available for PWM signal. GPIO.setmode is set to use the BCM (Broadcom SOC Channel) channel names. GPIO.setup defines the channel name and signal as output. GPIO.PWM enables the toggling of the output pin at the defined frequency. Pwm.start start the PWM mode with a duty cycle that is defined between 0 and 100 percent [47].

The PWM signal needs to be filtered and converter into an analog signal. This is achieved by the RC circuit with operational amplifier shown in Figure 4.8.

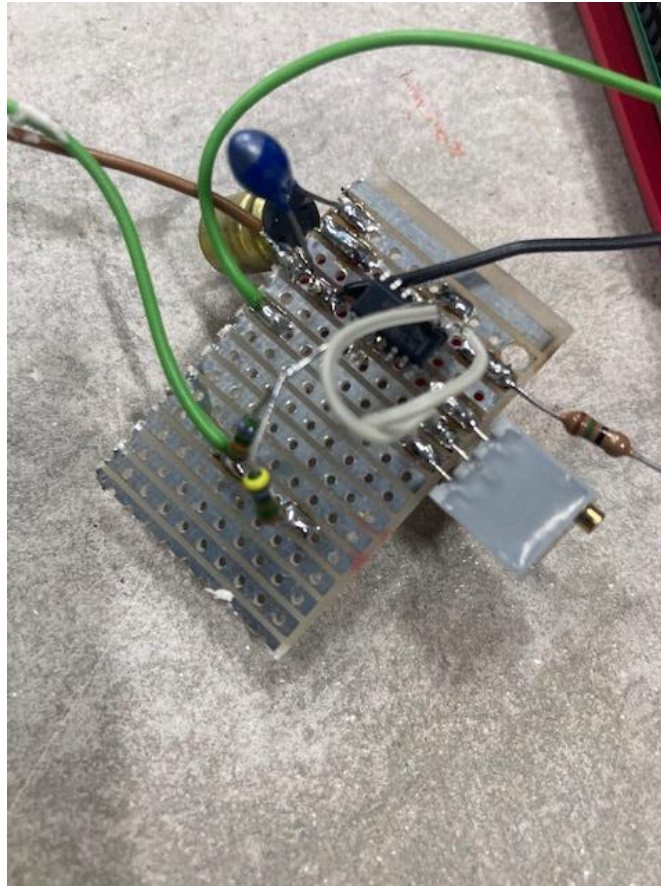


Figure 4.8: RC circuit with Op-Amp to filter the PWM signal from the Raspberry Pi

The green and brown wire is connected from the Raspberry Pi 3 GPIO pins; pin 12 and ground, respectively. The PWM signal is filtered by the resistor and capacitor in parallel, before its amplified by the Op Amp, MC 33074. Out from the Op Amp, the red, black, and green wire is connected directly to the converter. Figure 4.9 illustrates the filtering and converting of the PWM signal from the Raspberry Pi 3 to the converter.

4 Testing of the Digital Twin of Induction Heating Equipment

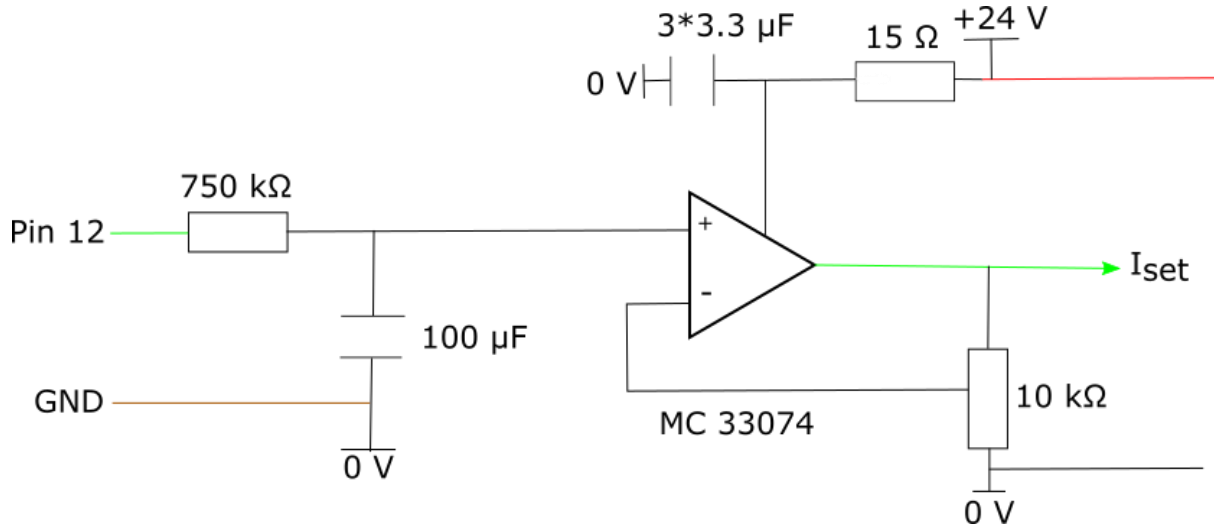


Figure 4.9: Circuit diagram of how the PWM signal from the Raspberry Pi is filtered and converted, before sending analog set point current to the converter

As mentioned earlier, the Raspberry Pi 3 with the Automation HAT board will function as the Converter Interface. Figure 4.10 show the wired connections between the Converter Interface and the real converter. To read the current, frequency and power from the converter, the Automation HAT board has three analog inputs. The green, red, and white wire read the current, frequency and power from the converter, respectively. The relay is output to the real converter and set the power on/off. Since the power of the converter is normally off, the relay on the Automation HAT board is connected to normally open (NO) and closes when a power on signal is received through MQTT to the COM terminal with the green wire to the converter. The NO terminal with the brown/yellow wire is connected to the 24V channel from the converter.

4 Testing of the Digital Twin of Induction Heating Equipment

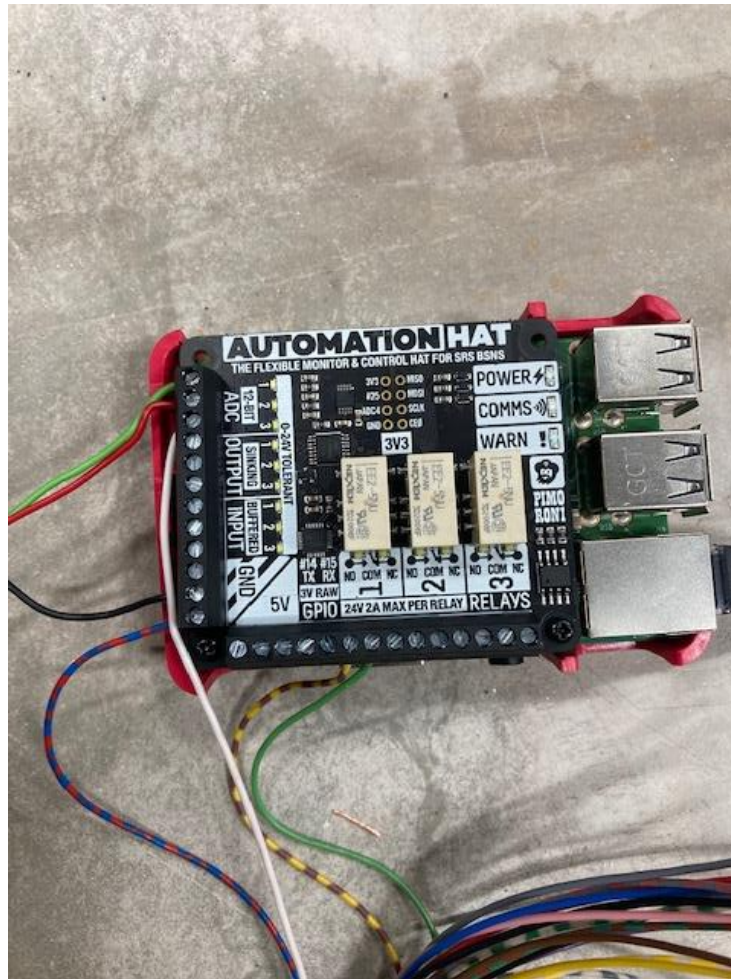


Figure 4.10: Wired connections between the analog inputs and digital outputs of the Automation HAT board and the real converter

An overlook of the connections between the hardware is shown in Figure 4.11. The secondary Raspberry Pi 3 board with the connections to the circuit for converting to analog signal is most adjacent, then the Raspberry Pi 3 with the Automation HAT board attached. Both circuits have wired connections to the real converter. The workpiece to be tested has wired connection to the SMART DAC+ GP20, attached by thermocouples.

4 Testing of the Digital Twin of Induction Heating Equipment

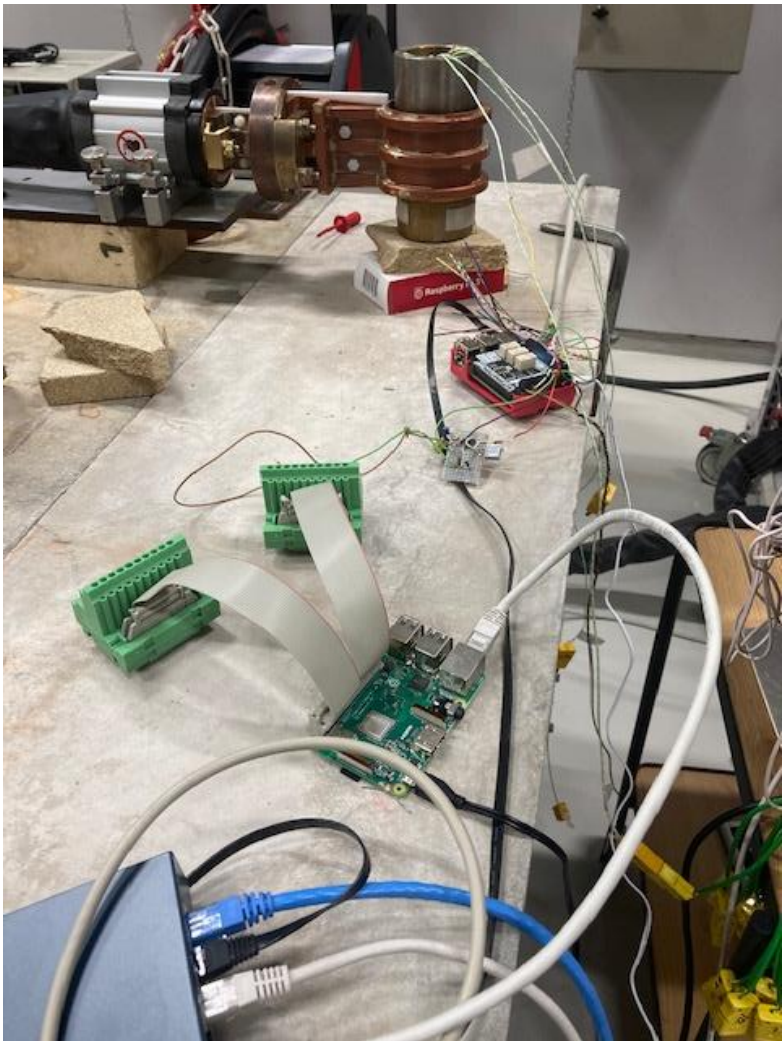


Figure 4.11: Full setup of the experiment

4.6 Experiments of the real converter

The experiments that were executed of the real converter are listed in Table 4.5, including result of the maximum temperature and power output from each experiment. The data recording starts when the power signal is set on from the Set Point Controller and each of the experiment are recorded for five minutes in the Data Storage model to have the same comparisons between all the results.

Table 4.5: Experiments executed and result of maximum temperature and power output in each experiment

Nr.	Current Set Point	Time Power On	Material: Magnetic/Non-Magnetic (M/NM)	Coil: Short/Long	Max. Power [kW]	Max. Temperature [°C]
1.	15 A	2 s	M	Short	1.6	36.4
2.	15 A	2 s	NM	Short	1.725	40.7
3.	15 A	10 s	M	Short	1.65	60.7
4.	15 A	10 s	NM	Short	0.6	38.7
5.	15 A	2 s	M	Long	2.2	34.7
6.	15 A	2 s	NM	Long	0.675	25.6
7.	15 A	10 s	M	Long	2.22	61.6
8.	15 A	10 s	NM	Long	0.675	29.8
9.	30 A	2 s	M	Short	4.825	65.4
10.	30 A	2 s	NM	Short	1.6	38.3
11.	30 A	10 s	M	Short	4.925	148.9
12.	30 A	10 s	NM	Short	1.6	48.5
13.	30 A	2 s	M	Long	6.3	55.4
14.	30 A	2 s	NM	Long	1.925	41.9
15.	30 A	10 s	M	Long	6.5	121.6
16.	30 A	10 s	NM	Long	1.875	74.7

4 Testing of the Digital Twin of Induction Heating Equipment

18.	45 A	2 s	NM	Short	3.3	45
19.	45 A	10 s	M	Short	9.65	275.6
20.	45 A	10 s	NM	Short	3.325	76
21.	45 A	2 s	NM	Long	3.85	73
21.	45 A	10 s	M	Long	11.9	200.9
22.	45 A	10 s	NM	Long	3.725	59.4
23.	60 A	2 s	NM	Short	5.525	51.4
24.	60 A	10 s	NM	Short	5.6	113
25.	60 A	2 s	NM	Long	6.025	41.9
26.	60 A	10 s	NM	Long	6.075	85.7
27.	45 A	2 x 10 s	NM	Long	3.725	81.6

4.6.1 Comments on the experiments

Ideally the workpieces should be at room temperature before every experiment to have the similar temperature development from start. In the experiments there are only one workpiece of each material to experiment on and it will then not be possible to have the same temperature of the workpieces at start of every experiment. To cool down the workpieces after an experiment, they have been submerged in water and then placed on the floor to stabilize the surface temperature before the next experiment. Especially if the workpiece has been heated up to several hundred degrees, this process of reaching room temperature again can take hours. The bulk volume will have a higher temperature than the surface temperature when the workpiece has been cooled down in water and will affect the temperature development when heated again. When a workpiece has been heated up sufficiently, the material properties have been permanently changed and will not give the same results when the process is repeated. To limit the heating of the workpieces, the setpoint currents has been relatively low to avoid too much power dissipated at the surface of the workpieces. The purpose of heating the workpieces has been to observe the respond in the materials compared to the Application Model with the different settings. Since the material properties change as the workpiece is heated, the Application Model will not be valid at high temperature since the properties are set as constant.

When simulating the Application Model, the frequency was set equal as in the experiment of the real application in the Converter Model. The Converter Model depends on receiving resistance and inductance of the workpiece from the Application Model to calculate the frequency. It was of more interest to compare the temperature development between the Application model and the real application, and it was therefore necessary to have identical settings when testing.

4 Testing of the Digital Twin of Induction Heating Equipment

The set point current from the Set Point Controller had a deviation from the real settings on the Sinac heating system. This may result from error in the circuit that replicate the AC current. The set point current was therefore set manually from the control panel to have the same setting.

4.6.2 Results of the experiments

4.6.2.1 Comparison between magnetic (M) and non-magnetic (NM) workpieces

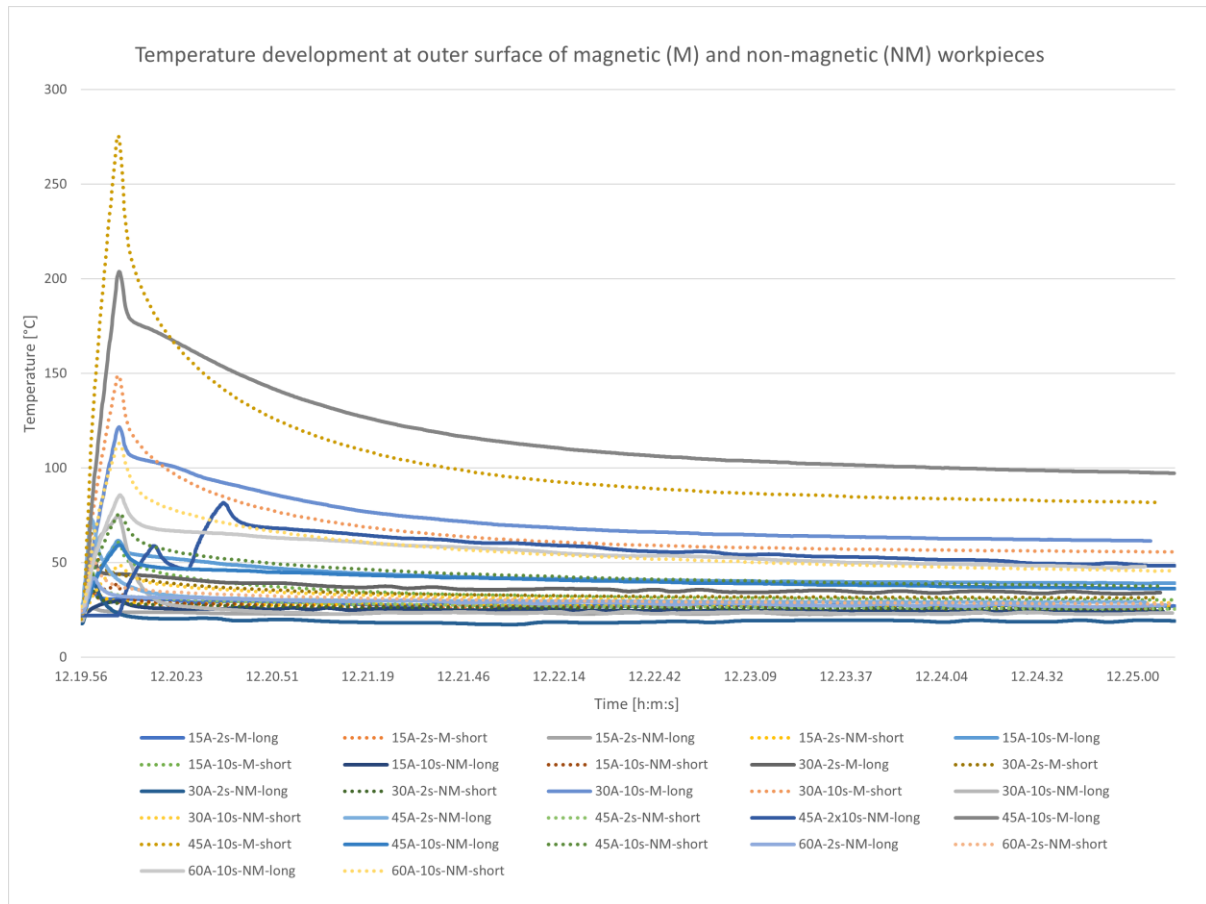


Figure 4.12: Plot of temperature development at outer surface of magnetic (M) and non-magnetic (NM) workpieces during a 5-minute interval from power on signal

The results from all the experiments are plotted in the same diagram, shown in Figure 4.12. To separate between the long and short coil, the results from the short coil has stapled lines, while the long coil has a full line. The maximum temperature was reached from the experiment ‘45A-10s-M-short’ at 275 °C which was expected since the workpiece is magnetic, and the coil was short. The following experiments with the highest temperature are the same workpiece with different settings and coil. For the non-magnetic material, the maximum temperature reached is 87 °C with the experiment ‘60A-10s-NM-short’.

All the experiments show the same pattern in the temperature development, with a steep temperature rise when the power is set on and a drop when the power is off. Except ‘45A-2x10s-NM-long’, which have a sequence with power on and off with 10s interval before stabilizing.

4 Testing of the Digital Twin of Induction Heating Equipment

It is a higher temperature difference between the outer surface and the bulk volume in the beginning that results in a higher heat transfer by conduction. When the heat has started to flow inwards, the temperature difference gets smaller. Since the difference is smaller, the heat conduction will also be smaller. After the heat has been distributed in the workpiece, the heat will start to flow to the surroundings by convection. This process is slower since only the surface of the workpiece is exposed to the air and temperature development slows down with a small decrease of temperature with time.

Different factors can contribute to errors in the measurement of the real application. Heat transfer from the workpiece is by natural convection, but it can be affected by the ventilation system or someone passing by at the moment of measuring. The measurement device, STORK TAU, was calibrated recently before the experiments and should measure the temperatures with good accuracy, but there is a possibility that the device did not measure accurately. There may also be a time delay of receiving the measurement through the MQTT and MODBUS protocols, resulting in a wrong impression of the temperature development.

4.6.2.2 Comparison between the non-magnetic (NM) workpiece and the Application Model

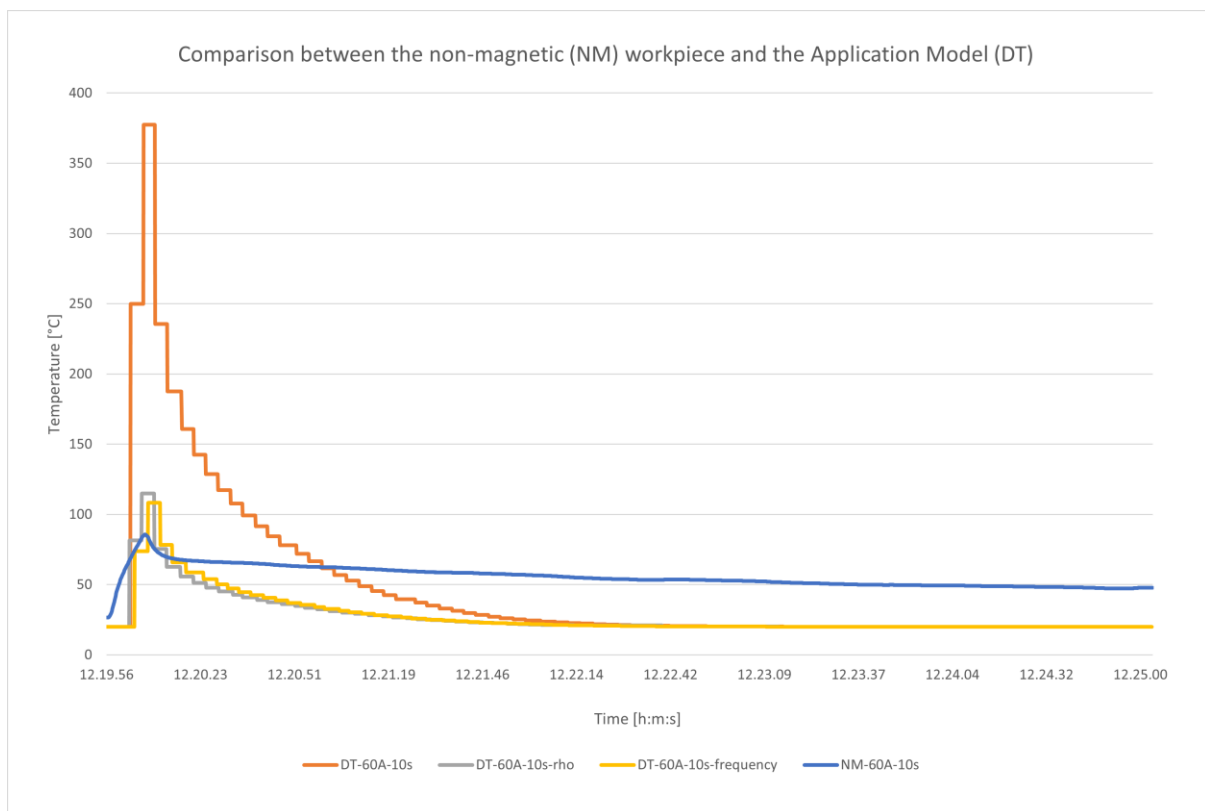


Figure 4.13: Comparison between NM workpiece and the Application Model (DT)

Figure 4.13 show the comparison between the non-magnetic (NM) material and the Application Model (DT), with set point current $I_{\text{set}} = 60 \text{ A}$ and simulation time $t = 10 \text{ s}$. While the non-magnetic material has a maximum temperature at 85°C from the experiment, the Application Model reached a maximum temperature at 377°C with the same settings.

4 Testing of the Digital Twin of Induction Heating Equipment

The results from the Application Model show that the heat transfer is much higher in the model than in the real application or at a higher speed. It stabilizes back to the initial temperature after a couple of minutes, while the workpiece has a much slower decline in the temperature fall.

To demonstrate how a change in material properties or setting influence the simulation results of the Application Model, the resistivity and frequency was changed isolated. First the resistivity was decreased by a tenth, keeping the other settings the same. Then the same with the frequency, decreasing it by a tenth and keeping the other setting the same. The temperature decreased to 115 °C and 108 °C at maximum temperature regarding resistivity and frequency, respectively. The decrease in frequency is shown to have more impact on the temperature than the resistivity. This demonstrates that the parameters can be adjusted to replicate the real temperature development.

The deviation results from several factors, such as different material properties, magnetic properties, heat transfer, computational error, and time delay through MQTT. As mentioned earlier, it is challenging to develop a model that simulates the real application because it depends on the material of the workpiece and there are unknown parameters that may influence the temperature development.

5 Conclusion

The last chapter of this thesis will give a conclusion of the usage of the digital twin at this stage and how it can benefit the customers of EFD Induction, and proposals for future work of the development of the digital twin.

5.1 Conclusion

The focus of this thesis has been on developing a digital twin for the induction heating equipment. A digital twin can have a variety of uses, and in this case the digital twin has been mainly used for monitoring the induction heating process when testing. This digital twin has been developed to be a base for the induction heating equipment, with the building blocks consisting of an Application Model, Converter Model, Set Point Controller, Data Storage and Converter Interface, and a connection between the models through the Data Exchange model. The digital twin has compared the temperature distribution between the real application and the Application Model and is a starting point for further development. It could eventually include more usages, such as testing what-if scenarios. As long a communication between the models have been established, there is a lot of potential of including additional models, such as a Model Parameter Tuner, and advancing the digital twin to represent a real application.

In the experiments, the digital twin was able to deliver set point settings from the Set Point Controller to the real converter through the Converter Interface and store the results from the experiments in the Data Storage model. The Set Point Controller received and displayed the parameters from the real application, making it easy to monitor the experiments. It was also able to publish a sequence with the power going on and off, through the implementation of timers in the model. The Converter Model was able to calculate the frequency, power and the current in the coil and communicate with the other models through Data Exchange. Since it requires information about the resistance and inductance of the workpiece, The Converter Model was not included in the experiments with the digital twin. It became more of an interest to compare the temperature development between the Application Model and the real application with the same settings. The Converter Interface with the Raspberry Pi 3 enabled the connection to the real converter and handled the power on and off signal from the Set Point Controller by the relay closing and opening on the Automation HAT board. Through the analog inputs on the Automation HAT board, the current, power and frequency was published through the Data Exchange and displayed in the Set Point Controller. Since there is not an available analog output on the Raspberry Pi 3, an additional Raspberry Pi 3 was included with available GPIO pins. With the help of an RC circuit for filtering and converting the PWM signal to an analog signal, it was able to deliver the set point current to the real converter. But since there was a deviation from the actual set point current, it was rather set manually on the control panel of the converter.

Results from the Application Model compared with the real application did correspond and showed similar temperature development, while not accurately. When the power 'on' signal with the set point current was published, the Application Model had a similar response as the real application with a rapid temperature increase in the workpiece, and a steady temperature decrease after the power was set off. The comparison between the temperature developments had a deviation, which was as expected since the Application Model has limitations regarding

constant material properties and neglecting magnetic properties. It is also shown how dependent the temperature development is on the material of the workpiece to be tested, comparing Stainless Steel 316L and 42CrMo4. This makes it challenging to develop a model that simulate the real application, since it is nonlinear and there are unknown parameters that have an influence on the temperature development. The shape of the coil had also an impact on the temperature development, where the experiments with the shorter coil resulted in a higher temperature increase than the experiments with the longer coil. It is therefore limited on what application the digital twin can be used for testing. The results after testing with different frequency and resistivity in the Application Model, showed that by adjusting the parameters it could be possible to replicate the temperature development from the real applications. A Model Parameter Tuner would be beneficial for the digital twin to optimize the parameters for the Application Model. The benefit with the Application Model is that it is possible to simulate how the temperature distribution is inwards in the workpiece, which is not possible with conventional temperature measurement with thermocouples that is attached on the surface. This can be valuable information when testing an application.

Literature survey on existing methods for testing and evaluating induction heating equipment showed that testing of application in other companies is executed in a similar way as in EDF Induction, with computer simulation in advance and testing the application in lab. It has not been focused on a literature survey on tools for developing digital twins in this thesis. The focus has rather been on developing the digital twin for this process and using known tools that is non-commercial and can easily be downloaded by the costumers of EFD Induction.

EFD Induction would benefit with having a digital twin regarding their customers. A digital twin would make them more attractive when choosing a manufacturer for developing their induction heating application. By having a digital twin that is accessible for the customers, they will be more involved in the process of testing their application and potentially develop a better solution. This will also improve their experience and impression of EFD Induction and would more likely choose EDF Induction when developing a new application at a later stage. It would also been beneficial if the digital twin could replace the testing of the real application in a lab. Often the workpieces that has been tested on, is no longer usable and must be replaced. This can be expensive if a lot of testing is required for the application, or the workpiece to be tested is of a large scale and cannot be easily replaced.

5.2 Future work

To replicate the real application more accurately, the Application Model should calculate the inductance and resistance of the workpiece to be tested from the material properties. The Converter Model will then be able to calculate the resonance frequency, power and current in the coil more accurately. Another possibility is to store information from previous experiments with real applications to train the Application Model by ML with a data-driven model to acknowledge realistic temperature distribution in the workpiece to be tested.

For controlling and monitor the induction heating process, the Set Point Controller should have a chart for plotting the results from both the digital twin and the real application. It could also be accessible from a web-browser to have an easy access. The customer can then easily monitor the results from different scenarios when testing.

5 Conclusion

A Model Parameter Tuner has not been developed in this Digital Twin. If this were included, it would benefit the Application Model for adjusting the parameters for the workpiece to be tested and simulate more accurately for the application.

The Data Storage model could be improved by developing a NoSQL database to store the data. This could have a module for generate reports of the testing of the application from the database.

References

- [1] VikramD, “Apollo 13: The First Digital Twin,” *Simcenter*, Apr. 14, 2020. <https://blogs.sw.siemens.com/simcenter/apollo-13-the-first-digital-twin/> (accessed May 04, 2021).
- [2] “Application of Digital Twin in Industrial Manufacturing,” *FutureBridge*, Feb. 06, 2020. <https://www.futurebridge.com/industry/perspectives-mobility/application-of-digital-twin-in-industrial-manufacturing/> (accessed May 04, 2021).
- [3] A. Rasheed, O. San, and T. Kvamsdal, “Digital Twin: Values, Challenges and Enablers From a Modeling Perspective,” *IEEE Access*, vol. 8, pp. 21980–22012, 2020, doi: 10.1109/ACCESS.2020.2970143.
- [4] “Digital Twin – Top Use Cases in Healthcare,” Mar. 05, 2021. <https://www.einfochips.com/blog/digital-twin-top-use-cases-in-healthcare/> (accessed May 04, 2021).
- [5] H. D. Young and R. A. Freedman, “Electromagnetic Induction,” in *University Physics with Modern Physics*, 13th edition., vol. 2, 2 vols., San Francisco, CA: Pearson Education Limited, 2012.
- [7] “What is Frequency Converter? How it works?” <http://www.frequencyinverter.org/what-is-frequency-converter-how-it-works-631601.html> (accessed Feb. 23, 2021).
- [8] “Induction hardening,” *EFD Induction*. <https://www.efd-induction.com/en/induction-heating-applications/induction-hardening> (accessed May 13, 2021).
- [9] “Induction brazing,” *EFD Induction*. <https://www.efd-induction.com/en/induction-heating-applications/induction-brazing> (accessed May 13, 2021).
- [10] “Induction welding,” *EFD Induction*. <https://www.efd-induction.com/en/induction-heating-applications/induction-welding> (accessed May 13, 2021).
- [11] “Inductoheat - Induction Process Development Laboratory,” *Inductoheat Inc*. <https://www.inductoheat.com/services/induction-process-development-laboratory/> (accessed Apr. 29, 2021).
- [12] “Plustherm. Your individual Induction Heating Solutions,” *Your induction solution finder*. <https://www.plustherm.com/> (accessed Apr. 29, 2021).
- [13] H. D. Young and R. A. Freedman, “Temperature and Heat,” in *University Physics with Modern Physics*, 13th edition., vol. 1, 2 vols., San Francisco, CA: Pearson Education Limited, 2012.
- [14] “Mathematical Modeling - MATLAB & Simulink Solutions.” <https://se.mathworks.com/solutions/mathematical-modeling.html> (accessed May 03, 2021).
- [15] Dr. M. Grieves, “Digital Twin: Manufacturing Excellence through Virtual Factory Replication.” 2003.

References

- [16] R. Minerva, G. M. Lee, and N. Crespi, “Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models,” *Proc. IEEE*, vol. 108, no. 10, pp. 1785–1824, Oct. 2020, doi: 10.1109/JPROC.2020.2998530.
- [17] “Digital Twin Example - How Does the Technology Works?,” *4subsea*. <https://www.4subsea.com/digital-twin-example-article/> (accessed May 12, 2021).
- [18] “Echo — Equinors digitale tvilling - En interaktiv 3D-modell i lommen - equinor.com.” <https://www.equinor.com/no/magazine/echo-equinors-digital-twin.html> (accessed May 14, 2021).
- [19] “The Future of Mobility: Tomorrow’s Ships: Born in an Ocean of Data,” *siemens.com Global Website*. <https://new.siemens.com/global/en/company/stories/research-technologies/digitaltwin/future-of-mobility-efficiency-for-shipping.html> (accessed May 12, 2021).
- [20] “Modern manufacturing’s triple play: Digital twins, analytics, IoT.” https://www.sas.com/en_us/insights/articles/big-data/modern-manufacturing-s-triple-play-digital-twins-analytics-iot.html (accessed May 04, 2021).
- [21] “5 Real World Examples of Digital Twins,” *PALAMIR ... smart resilient cities*. <https://www.palamir.com/news/5-real-world-examples-of-digital-twins> (accessed May 04, 2021).
- [22] “Digital twin cities and the future of urban planning.” <https://www.itu.int:443/en/myitu/News/2020/10/21/14/15/Digital-twin-cities-Microsoft-Bentley-Systems> (accessed May 12, 2021).
- [23] “MQTT,” *Wikipedia*. Jan. 21, 2021. Accessed: Feb. 10, 2021. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=MQTT&oldid=1001776985>
- [24] T. H. Team, “Client, Broker / Server and Connection Establishment - MQTT Essentials: Part 3.” <https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment> (accessed Feb. 16, 2021).
- [25] “MQTT Version 3.1.1.” http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718028 (accessed Mar. 09, 2021).
- [26] T. H. Team, “MQTT Publish, Subscribe & Unsubscribe - MQTT Essentials: Part 4.” <https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe> (accessed Mar. 09, 2021).
- [27] steve, “MQTT Publish and Subscribe Beginners Guide,” Jun. 04, 2018. <http://www.steves-internet-guide.com/mqtt-publish-subscribe/> (accessed Mar. 09, 2021).
- [28] T. H. Team, “Quality of Service 0,1 & 2 - MQTT Essentials: Part 6.” <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels> (accessed Feb. 23, 2021).
- [29] T. H. Team, “MQTT Topics & Best Practices - MQTT Essentials: Part 5.” <https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices> (accessed Mar. 11, 2021).
- [30] R. Light, *paho-mqtt: MQTT version 5.0/3.1.1 client class*. Accessed: May 16, 2021. [Online]. Available: <http://eclipse.org/paho>

References

- [31] “Paho Python MQTT Client - Working with Connections,” Aug. 20, 2016. <http://www.steves-internet-guide.com/client-connections-python-mqtt/> (accessed Feb. 16, 2021).
- [32] H. P. Langtangen and A. Logg, “Solving PDEs in Python – The FEniCS Tutorial Volume I,” p. 153.
- [33] H. D. Young and R. A. Freedman, “Sources of Magnetic Field,” in *University Physics with Modern Physics*, 13th edition., vol. 2, 2 vols., San Francisco, CA: Pearson Education Limited, 2012.
- [34] H. D. Young and R. A. Freedman, “Alternating Current,” in *University Physics with Modern Physics*, 13th edition., vol. 2, 2 vols., San Francisco, CA: Pearson Education Limited, 2012.
- [35] *eclipse/paho.mqtt.m2mqtt*. Eclipse Foundation, 2021. Accessed: May 16, 2021. [Online]. Available: <https://github.com/eclipse/paho.mqtt.m2mqtt>
- [36] “A Guide to Logging MQTT Sensor Data,” Mar. 19, 2018. <http://www.steves-internet-guide.com/logging-mqtt-sensor-data/> (accessed Apr. 11, 2021).
- [37] J. Coe, “SQL Basics — Hands-On Beginner SQL Tutorial Analyzing Bike-Sharing,” *Dataquest*, Feb. 01, 2021. <https://www.dataquest.io/blog/sql-basics/> (accessed Apr. 14, 2021).
- [38] “SQL vs NoSQL: What’s the Difference Between SQL and NoSQL.” <https://www.guru99.com/sql-vs-nosql.html> (accessed Apr. 14, 2021).
- [39] “Comma-separated values,” *Wikipedia*. Mar. 23, 2021. Accessed: Apr. 14, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Comma-separated_values&oldid=1013831687
- [40] *pimoroni/automation-hat*. Pimoroni Ltd, 2021. Accessed: May 16, 2021. [Online]. Available: <https://github.com/pimoroni/automation-hat>
- [41] “Ovako 42CrMo4 EN10083-3 Steel.” <http://www.matweb.com/search/datasheet.aspx?matguid=6dc42d4c5aa647e28ebac62f1bc34336> (accessed Apr. 28, 2021).
- [42] “AISI Type 316L Stainless Steel, annealed sheet.” http://www.matweb.com/search/datasheet_print.aspx?matguid=1336be6d0c594b55afb5ca8bf1f3e042 (accessed Apr. 28, 2021).
- [43] “Stork Cooperheat | Thermocouple Attachment Unit (230V a.c.) - Stork.” <https://www.stork.com/en/cooperheat-equipment-shop/thermocouple-attachment-unit-battery-recharge-supply-230v-ac> (accessed Apr. 27, 2021).
- [44] “MODBUS – protokoll for industriell kommunikasjon | WAGO NO.” <https://www.wago.com/no/modbus> (accessed Apr. 19, 2021).
- [45] L. Lefebvre, *pyModbusTCP: A simple Modbus/TCP library for Python*. Accessed: May 13, 2021. [Online]. Available: <https://github.com/sourceperl/pyModbusTCP>
- [46] “Welcome to pyModbusTCP’s documentation — pyModbusTCP 0.1.10 documentation.” <https://pymodbustcp.readthedocs.io/en/latest/> (accessed Apr. 27, 2021).

References

- [47] “Control Raspberry Pi GPIO Pins from Python,” *ICS - Integrated Computer Solutions*. <https://www.ics.com/blog/control-raspberry-pi-gpio-pins-python> (accessed Apr. 25, 2021).
- [48] “mosquitto.conf man page,” *Eclipse Mosquitto*, Apr. 06, 2021. <https://mosquitto.org/man/mosquitto-conf-5.html> (accessed Apr. 14, 2021).
- [49] “Setting up a Raspberry Pi headless - Raspberry Pi Documentation.” <https://www.raspberrypi.org/documentation/configuration/wireless/headless.md> (accessed Mar. 24, 2021).
- [50] “IP Address - Raspberry Pi Documentation.” <https://www.raspberrypi.org/documentation/remote-access/ip-address.md> (accessed Mar. 24, 2021).
- [51] “Equip Raspberry Pi with a static IP address,” *IONOS Digitalguide*. <https://www.ionos.com/digitalguide/server/configuration/provide-raspberry-pi-with-a-static-ip-address/> (accessed Mar. 24, 2021).

6 Appendices

Appendix A Task description of the master's thesis



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

FMH606 Master's Thesis

Title: Digital Twin for Industrial Induction Heating Equipment

USN supervisor: Kjetil Svendsen, with co-supervisor Nils-Olav Skeie

External partner: EFD Induction

Task background:

EFD Induction has a comprehensive range of induction heating products and predicting operation of these products in actual applications may be a challenge. EFD wants to extend the testing and evaluation of these products and applications with better simulators. These simulators should also use the setup from the physical devices and compare the results with the results from the physical devices, and QA requirements for approval. A good tool is a "Digital Twin" that can support this functionality as a digital twin can be a simulator interacting with a physical device. It is also important that the Digital Twin supports the principles of Interconnection, Information Transparency, Technical Assistance, and Decentralized Decisions that are important principles in modern industrial ecosystems.

Task description:

A digital twin for testing and evaluating induction heating is wanted, Proposal of the tasks are:

- Make a literature survey of available tools for testing and evaluating induction heating applications.
- Make a literature survey on how a digital twin can be modelled and what programs that can be used
- Make an overview of the building blocks, the information flow and the usage of mathematical models in digital twins.
- Investigate how digital twins can be used at EFD Induction and by their customers, with focus on induction heating equipment.
- Develop model(s) describing the usage of induction heating products.
- Implement the model(s) as a digital twin with interconnection to an induction heating product.
- Make test plan for testing the digital twin.

Student category: EPE

The task is suitable for online students (not present at the campus): Yes


Practical arrangements:

Product interface and data will be available from EFD Induction.

Supervision:

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

Signatures:

Supervisor (date and signature): 1/2 2021 

Student (write clearly in all capitalized letters): RAGNTHILD MOLDESÆTHER

Student (date and signature): 01.02.21, Ragnhild Moldesæther

Appendix B Set up of MQTT Broker

Mosquitto is installed to function as the MQTT Broker in the connection between the models of the Digital Twin. Mosquitto is part of the Eclipse foundation and can be installed through their homepage. The software is open-source and can be installed on a wide range of platforms, including Windows and Linux distributions.

When installing Mosquitto, there is an option to automatically start up the broker when starting the machine. If this is not chosen, the Mosquitto broker can be started by entering in the command line as administrator, shown in Figure 6.1. If the Mosquitto broker service is to stop, 'net stop mosquitto' is entered.

```
C:\WINDOWS\system32>net start mosquitto
Tjenesten Mosquitto Broker starter.
Tjenesten Mosquitto Broker ble startet.

C:\WINDOWS\system32>
```

Figure 6.1: Start up MQTT Mosquitto Broker

With the install of Mosquitto, a configuration file is included in the Mosquitto folder. There is no need to edit this file for the Mosquitto broker to start and the complete file is commented out. Depending on what changes that need to be made, the respective line can be uncommented and edited. This applies if there is need for authentication with the use of username and password. 'allow_anonymous' is a Boolean value that can set to true or false [48].

```
512 allow_anonymous false
```

Figure 6.2: Allow anonymous set to false in the configuration file

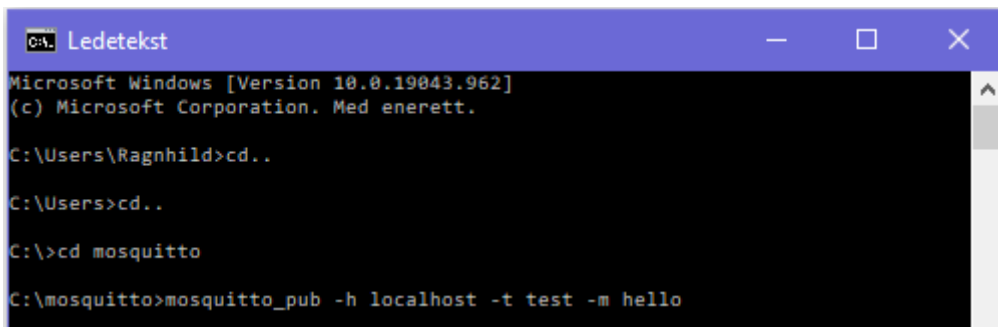
If set to false, no other clients can connect to the broker without providing username and password. A username/password file can be created in a text file with the format username:password, and located in the Mosquitto folder.

```
530 password_file C:/mosquitto/passwd
```

Figure 6.3: Enable password file and path in the configuration file

When another client is connecting to the broker, the client must specify username and password for access.

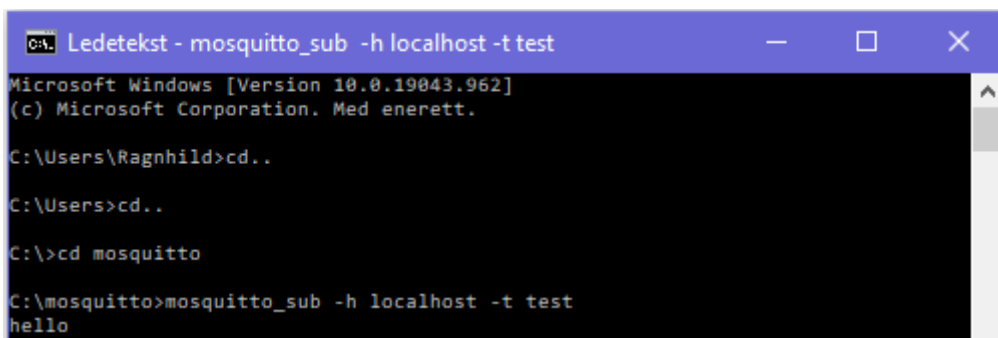
With the Mosquitto software, there is also included MQTT client tools. 'mosquitto_pub' is a function for publishing message to a topic. An example is shown in Figure 6.4. In this function, there are different flag options. The flags that are the most basic and necessary for publishing a message is -h Host address, -t Topic and -m Message Payload. Host address will be the address of the broker. Since Mosquitto Broker is installed on the computer, localhost can be used as host address. By default, localhost will always be chosen as host address if nothing else is specified. Topic is defined as 'test' for this purpose, while the message of payload to be sent is 'hello'.



```
C:\Users\Ragnhild>cd..
C:\Users>cd..
C:\>cd mosquitto
C:\mosquitto>mosquitto_pub -h localhost -t test -m hello
```

Figure 6.4: Test of Mosquitto client function 'mosquitto_pub' in Windows command line

After entering, the message will be published to the topic. To receive the message that have been sent, the 'mosquitto_sub' function can be used for subscribing to the topic. Figure 6.5 show the function with the necessary flags, -h and -t, to receive the message 'hello'. It is important that the same host address and topic is defined as in the 'mosquitto_pub' function to receive the message, since it is the broker that distributes the messages that are being published and is the connection between the clients.



```
C:\Users\Ragnhild>cd..
C:\Users>cd..
C:\>cd mosquitto
C:\mosquitto>mosquitto_sub -h localhost -t test
hello
```

Figure 6.5: Test of Mosquitto client function 'mosquitto_sub' in Windows command line

There are multiple public brokers available that are open for everyone to use. Some examples are: 'test.mosquitto.org', 'broker.hivemq.com' and 'iot.eclipse.org'. If the purpose of the MQTT communication is for testing and exploring the functions, the public brokers are fine to use. But for publishing and subscribing on important and sensitive information, a localhost with MQTT broker installed should be used.

Appendix C Installing FEniCS in Ubuntu

To get access to the FEniCS software library, the program must be run in a Linux operating system. Instead of having a virtual machine, Windows Subsystem for Linux (WSL) gives access to Linux software programs in the Windows environment through a Linux distributor. Ubuntu is chosen as a Linux distributor since it is user-friendly and suitable for beginners.

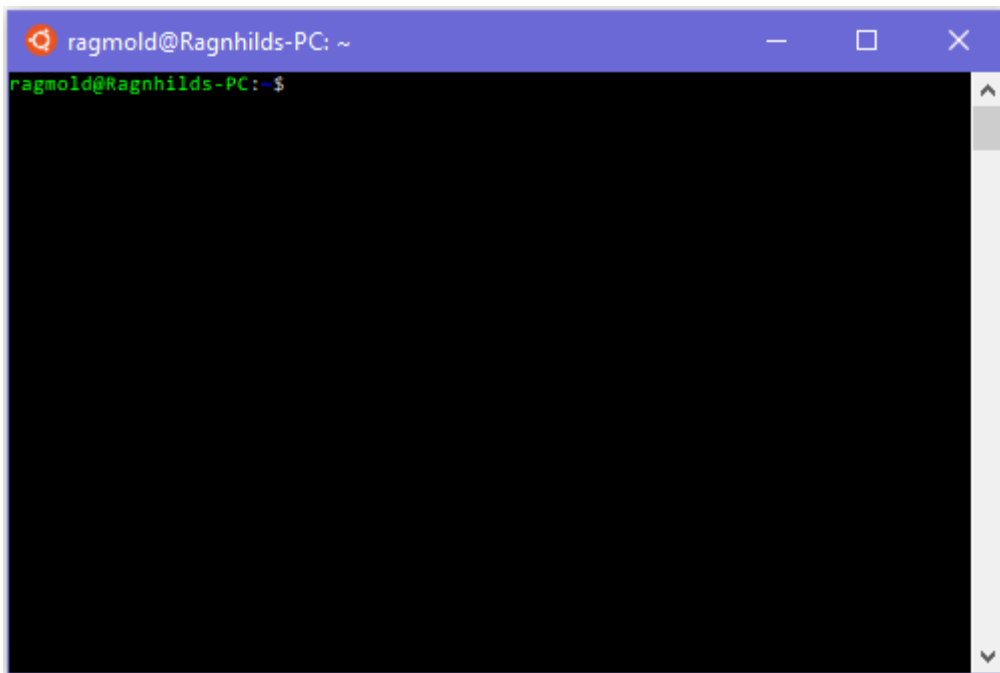


Figure 6.6: Ubuntu terminal as an interface to the Linux operating system

Figure 6.6 show the terminal for Ubuntu. In this shell, commands can be written and executed. The FEniCS library is installed through Ubuntu by the command lines:

```
sudo add-apt-repository ppa:fenics-packages/fenics
```

```
sudo apt-get update
```

```
sudo apt get install fenics
```

```
sudo apt-get dist-upgrade
```

FEniCS is then added to Ubuntu's list of software sources and is added to Python by the command line:

```
python3 -c 'import fenics'
```

The application model in Python with the imported FEniCS library can now be run in the Ubuntu terminal by changing directory to the file location in Windows.

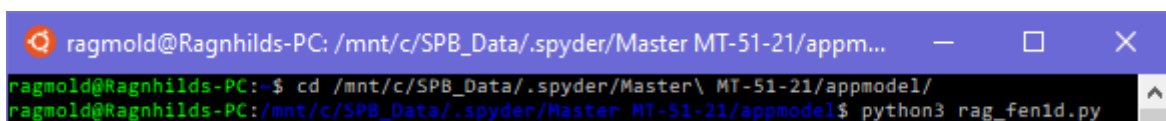


Figure 6.7: Run Python script with FEniCS from Ubuntu terminal

Appendix D Complete set up of Raspberry Pie 3

Raspberry Pi 3 is a small computer that can be programmed to execute tasks specified by the user. Figure 6.8 show the Raspberry Pi 3 on a model B+ board. The board has a 64-bit quad-core processor, 1 GB RAM, micro SD port, 40 pin GPIO header, four USB ports, one HDMI port and Ethernet connection. This board also comes with the possibility to connect through wireless network. For power supply, a micro USB port is accessible.



Figure 6.8: Raspberry Pie 3 Model B+ board

A full first set up of the Raspberry Pi 3 is shown in Figure 6.9. This includes computer mouse, keyboard, Ethernet cable and HDMI cable to a monitor.



Figure 6.9: Full set-up of the Raspberry Pi 3

For the Raspberry Pi to function, the SD card must have the operating system installed. There are several Linux-based operating systems that are available, such as Raspbian, Ubuntu and Windows IoT Core. Raspbian is the operating system that are recommended for the Raspberry Pi since it is especially designed for this hardware.

To install Raspbian operating system, the SD card must first be inserted in a SD card port on a computer. When the SD card is inserted, a folder named 'boot' is shown. This is a folder containing files that are formatted with the FAT file system. This means that the files are visible on both Windows and Linux machines. Boot is another word for start up, and this folder will store the necessary files for the operating system to start up. The operating system is an image that must be downloaded and installed on the SD card. Raspberry Pi Imager is a program that can be downloaded in Windows and is a card reader that writes the image to the SD card, shown in Figure 6.10.

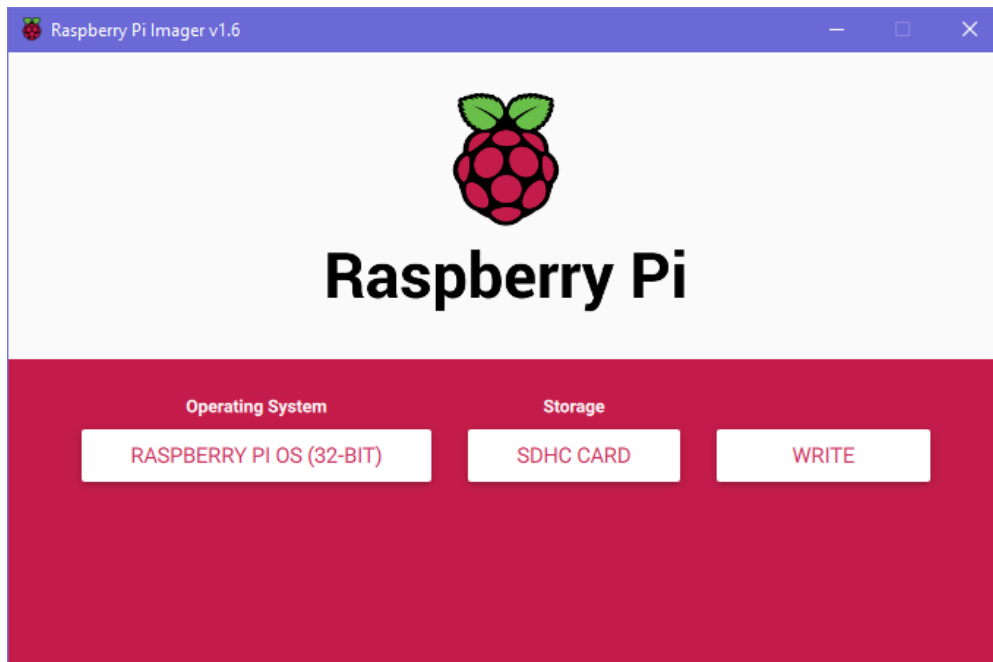


Figure 6.10: Raspberry Pi Imager is used for writing the operating system over to a SD card

The operating system is chosen from a list, then the storage, which will be the inserted SD card. When this is chosen, the image can be written to the SD card by clicking 'Write'.

It can be a benefit to have an empty SD card when installing the operating system to avoid any issues. SD Memory Card Formatter is a program that can be downloaded in Windows for formatting the SD card and erase all previous files.

On the underside of the Raspberry Pi there is a microSD port where the SD card with the installed Raspbian will be inserted. Since there is no power switch, the Raspberry Pi will turn on as soon as the power supply is connected. It is therefore important to do all other connections before the power supply is connected to ensure no damage to other components. When starting the Raspberry Pi, a rainbow is first displayed at the monitor before the desktop of the Raspbian operating system appears, shown in Figure 6.11.

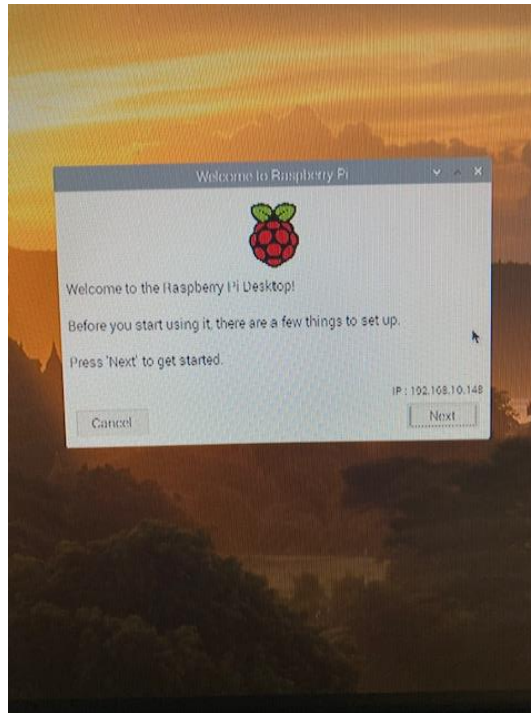


Figure 6.11: Desktop of the Raspbian operating system

The Raspberry Pi is now ready for the initial setup. The setup includes choosing country, language, and time-zone. A new password is recommended for the Raspberry Pi which is necessary when there is a remote access. Then the Raspberry Pi needs to be connected to a network. For the Model B+ board it is possible to connect to a wireless network, which is chosen here. To know the IP address of the Raspberry Pi, enter 'hostname -I' in the command line and the IP address will be revealed.

At last, there is a check for update and install of software, and a reboot of the Raspberry Pi. The Raspberry Pi is now finished with the setup and ready for use.

Appendix E Headless setup of the Raspberry Pi 3

To have remote access to the Raspberry Pi, the Raspberry Pi is also configured to have a headless setup. A headless setup means a setup of the Raspberry Pi without monitor, mouse, and keyboard. The purpose is to control and monitor the Raspberry Pi from another computer.

For the computer to get access to the Raspberry Pi, there must be defined a file in the boot folder with information about the wireless network. To edit the boot folder, The SD card must be inserted back in the computer. Inside this folder, a new text file named 'wpa_supplicant.conf' must be defined.

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=<Insert 2 letter ISO 3166-1 country code here>

network={
    ssid="<Name of your wireless LAN>"
    psk="<Password for your wireless LAN>"
}
```

Figure 6.12: wpa_supplicant.conf file to define wireless network[49]

In the 'wpa_supplicant.conf' file shown in Figure 6.12, the lines that needs to be edited are 'country' with country code, and 'ssid' and 'psk' under 'network' with name and password of the wireless network. When this is edited, the file must be saved as a text file and the SD card can be removed. Next time the Raspberry Pi starts up with the SD card, this file will be copied to the operating system and use the defined settings to connect to the network [49].

To remotely access the command line of the Raspberry Pi, SSH (Secure Shell) must be enabled. Inside the boot folder, a file named 'ssh' must be created. The content of the file is not of importance as the Raspberry Pi only detects the file name and then deletes the file. From the desktop this can be edited in the Raspberry Pi Configuration Tool. This is done by entering 'sudo raspi-config' in the command line and the Raspberry Pi configuration is launched, shown in Figure 6.13.

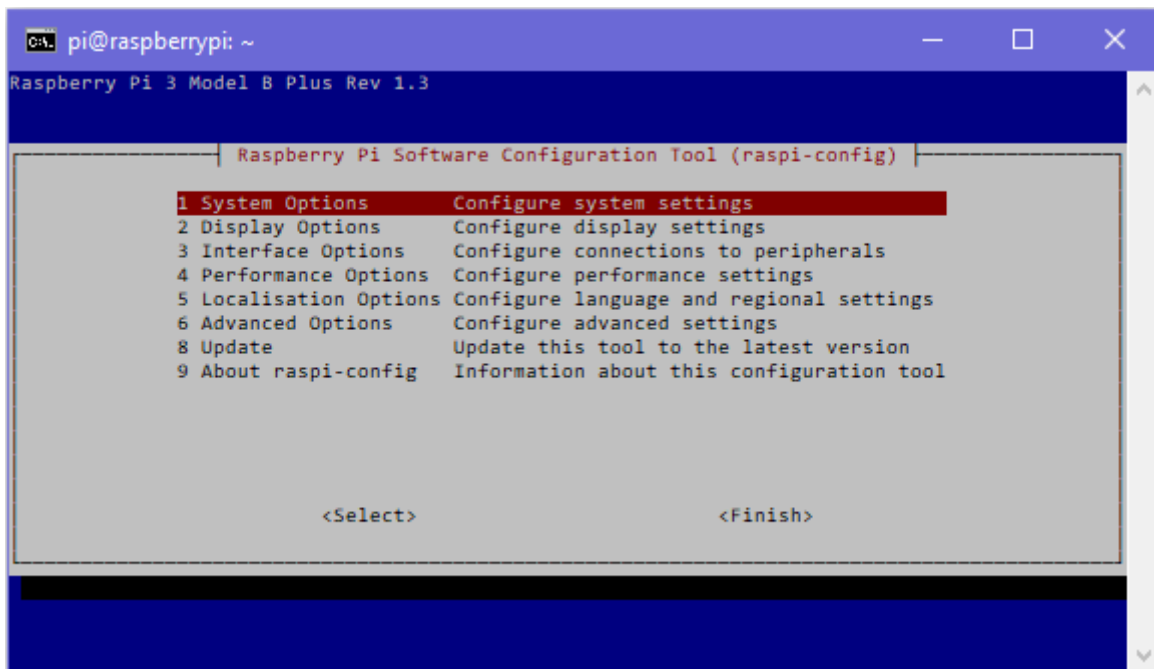


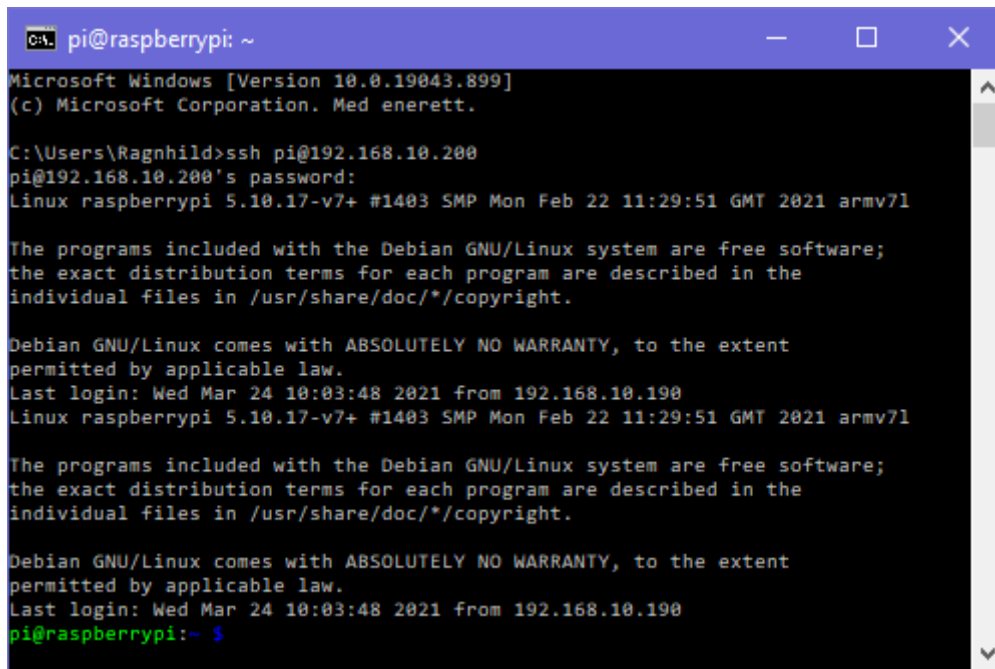
Figure 6.13: Raspberry Pi Configuration from the command line

By entering '3 Interface Options' and then navigate to SSH, SSH can be disabled/enabled.

It is important to note the IP address of Raspberry Pi when the access is done remotely for the first time. In this case, the first set up was with the monitor included where the IP address could be revealed from the command line. With a headless setup, this will not be possible. The IP address can then be found in the router's list of connected devices by login on the computer [50].

SSH is included in Windows 10 by default, but this can be checked by going to Settings > Apps > Apps and Features > Optional Features and search for OpenSSH Client and install if not included.

In the Windows command line, type 'ssh pi@pi-address', press enter and then type in the password for the Raspberry Pi. If there is a successful connection, the Raspberry Pi will be remotely accessed as shown in Figure 6.14.



```

C:\Users\Ragnhild> ssh pi@192.168.10.200
pi@192.168.10.200's password:
Linux raspberrypi 5.10.17-v7+ #1403 SMP Mon Feb 22 11:29:51 GMT 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Mar 24 10:03:48 2021 from 192.168.10.190
Linux raspberrypi 5.10.17-v7+ #1403 SMP Mon Feb 22 11:29:51 GMT 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

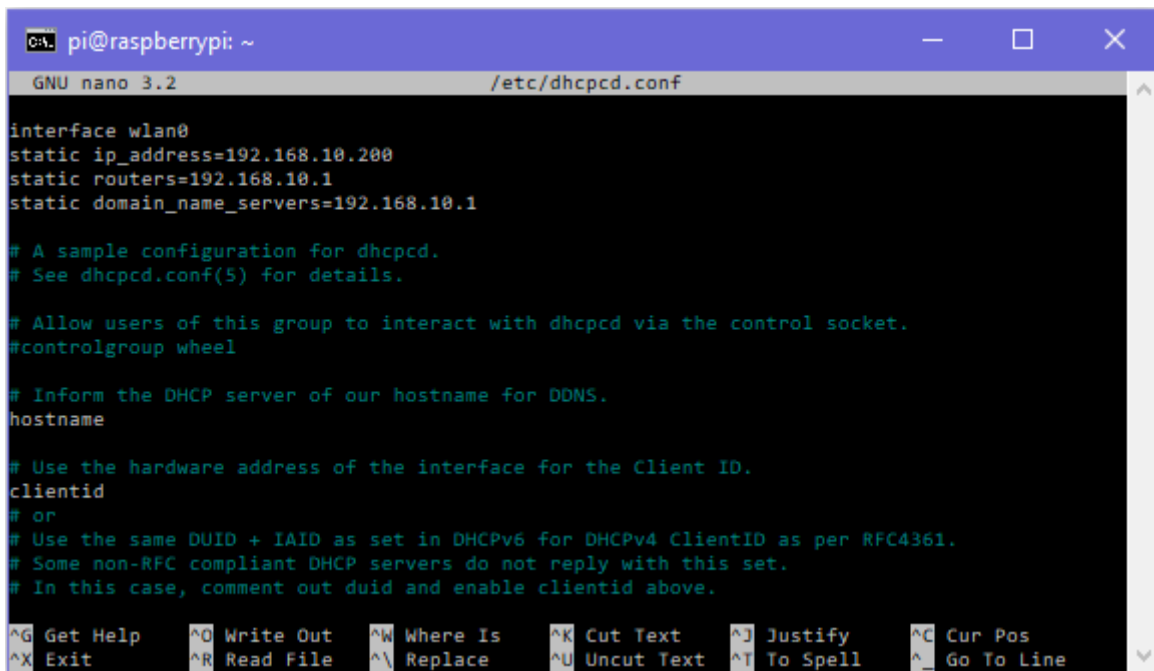
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Mar 24 10:03:48 2021 from 192.168.10.190
pi@raspberrypi:~ $

```

Figure 6.14: Remote access to the Raspberry Pi using Windows SSH Client

Each time the Raspberry Pi is connected to the network, it is given an IP address by the router. To be able to access the Raspberry Pi headless and ensure connection, the IP address must then be static. By entering 'ifconfig' in the command line of the Raspberry Pi, the IP address is shown under 'wlan0', which stands for wireless LAN. This is the IP address given by the router for connecting to the network through the DHCP server. This server has a range of IP addresses it gives to clients when connecting to the network and to avoid that other clients will cause conflict with the Raspberry Pi, an IP address outside the server's range should be chosen. A login to the router at the web browser will give information about the DHCP server and its IP address range. The Raspberry Pi also have a range of IP addresses. The static IP address must then be outside the DHCP server range but still inside the Raspberry PI range.

By entering 'sudo nano /etc/dhcpd.conf' in the command line in the Raspberry Pi it is possible to configurate the static IP address by editing the dhcpd.conf file shown in Figure 6.15.



```
pi@raspberrypi: ~
GNU nano 3.2 /etc/dhcpd.conf
interface wlan0
static ip_address=192.168.10.200
static routers=192.168.10.1
static domain_name_servers=192.168.10.1

# A sample configuration for dhcpd.
# See dhcpd.conf(5) for details.

# Allow users of this group to interact with dhcpd via the control socket.
#controlgroup wheel

# Inform the DHCP server of our hostname for DDNS.
hostname

# Use the hardware address of the interface for the Client ID.
clientid
# or
# Use the same DUID + IAID as set in DHCPv6 for DHCPv4 ClientID as per RFC4361.
# Some non-RFC compliant DHCP servers do not reply with this set.
# In this case, comment out duid and enable clientid above.

^G Get Help      ^O Write Out    ^W Where Is    ^K Cut Text     ^J Justify     ^C Cur Pos
^X Exit          ^R Read File    ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line
```

Figure 6.15: Configuration for DHCP

On top of the file, the static IP address is assigned to the Raspberry Pi. The address of the router and domain name server (DNS) are both the same in this case. The file can now be saved by ctrl + X and entering ‘Yes’ for the changes to be saved. The Raspberry Pi also needs to be rebooted [51].

Since there is no power switch on the Raspberry Pi, the commands ‘sudo poweroff’ and ‘sudo reboot’ can be entered in the command line. It is not recommended to pull out the power supply while the Raspberry Pi is on, as it can cause damage to components and the SD card.