FH Vorarlberg
University of Applied Sciences

Data-Driven Energy Modeling of a Dynamical System

Energy Modeling with Neural Networks on the Example of an Industrial Robot

Master Thesis

Submitted in partial fulfillment of the requirements for the Degree of
Master of Science in Engineering, MSc

Handed in by:
Philipp Steurer

Supervisor:
DI Dr.techn. Ralph Hoch

Course of study:
Mechatronics

Dornbirn, September 2021

Abstract

Data-Driven Energy Modeling of a Dynamical System

The purpose of an energy model is to predict the energy consumption of a real system and to use this information to address challenges such as rising energy costs, emission reduction or variable energy availability. Industrial robots account for an important share of electrical energy consumption in production, which makes the creating of energy models for industrial robots desirable. Currently, energy modeling methods for industrial robots are often based on physical modeling methods. However, due to the increased availability of data and improved computing capabilities, data-driven modeling methods are also increasingly used in areas such as modeling and system identification of dynamic systems. This work investigates the use of current data-driven modeling methods for the creation of energy models focusing on the energy consumption of industrial robots.

For this purpose, a robotic system is excited with various trajectories to obtain meaningful data about the system behavior. This data is used to train different artificial neural network (ANN) structures, where the structures used can be categorized into (i) Long Short Term Memory Neural Network (LSTM) with manual feature engineering, where meaningful features are extracted using deeper insights into the system under consideration, and (ii) LSTM with Convolutional layers for automatic feature extraction. The results show that models with automatic feature extraction are competitive with those using manually extracted features. In addition to the performance comparison, the learned filter kernels were further investigated, whereby similarities between the manually and automatically extracted features could be observed. Finally, to determine the usefulness of the derived models, the best-performing model was selected for demonstrating its performance on a real use case.

Kurzreferat

Datengetriebene Energiemodellierung eines Dynamischen Systems

Der Zweck eines Energiemodells ist es, den Energieverbrauch eines realen Systems vorherzusagen und diese Informationen zu nutzen, um Herausforderungen wie steigende Energiekosten, Emissionsreduzierung oder variable Energieverfügbarkeit zu bewältigen. Ein signifikanter Anteil des elektrischen Energieverbrauchs in der Produktion entfällt dabei auf Industrieroboter, was die Erstellung von Energiemodellen für diese Systeme erstrebenswert macht. Derzeit basieren die Energiemodellierungsmethoden für Industrieroboter häufig auf physikalischen Modellierungsmethoden. Aufgrund der steigenden Verfügbarkeit von Daten und verbesserter Rechenkapazitäten werden jedoch zunehmend auch datengetriebene Modellierungsmethoden in Bereichen wie der Modellierung und Systemidentifikation dynamischer Systeme eingesetzt. In dieser Arbeit wird der Einsatz aktueller datengetriebener Modellierungsmethoden für die Erstellung von Energiemodellen mit Fokus auf den Energieverbrauch von Industrierobotern untersucht.
Zu diesem Zweck wird ein Robotersystem mit diversen Trajektorien angeregt, um aussagekräftige Daten über das Systemverhalten zu erhalten. Diese Daten werden dann verwendet, um verschiedene Strukturen künstlicher neuronaler Netze (ANN) zu trainieren, wobei die verwendeten Strukturen in (i) Long Short Term Memory Neural Network (LSTM) mit manuellem Feature-Engineering, wobei aussagekräftige Features unter Verwendung tiefgehender Kenntnisse des betrachteten Systems extrahiert werden, und (ii) LSTM mit Convolutional Layern zur automatischen Feature-Extraktion unterschieden werden können. Die Ergebnisse zeigen, dass Modelle mit automatischer Feature-Extraktion konkurrenzfähig mit jenen mit manuell extrahierten Features sind. Zusätzlich zum Performancevergleich wurden die gelernten Filtermasken weiter untersucht, wobei Ähnlichkeiten zwischen den manuell und automatisch extrahierten Features festgestellt werden konnten. Abschließend wurde zur Bestimmung der Anwendbarkeit der erstellten Modelle, das effektivste Modell ausgewählt, um seine Leistung in einem realen Anwendungsfall zu demonstrieren.

Acknowledgments

First of all, I would like to thank Prof.(FH) DI Dr. Robert Merz for the opportunity to join the team of the Research Center Digital Factory Vorarlberg and for providing the topic leading to this thesis.
Furthermore, a big thank you to my supervisor DI Dr.techn. Ralph Hoch for excellent support and the very valuable and profound feedback during this work. I would also like to thank my colleagues at the research center for important and inspiring discussions, here a special thanks to DI Dr.techn. Sebastian Hegenbart, who gave me important inputs with his technical expertise in machine learning.

Moreover, I would like to thank my parents, whose support made my studies possible in the first place.

Most of all, I would like to thank my girlfriend Bianca. Especially in the last few months, you gave me tremendous support and made countless sacrifices to help me get to this point. I couldn't have done it without you.

Thanks!

This thesis was carried out as part of FFG project 87599 *Digital Energy Twin - Optimized Operation and Design of Industrial Energy Systems*, funded by the Austrian *"Klima- und Energiefonds"* within program *"Energy Research (e!MISSON)"*

Contents

List of Figures

1. Introduction

There is growing interest in reducing the energy consumption of production processes as stricter guidelines for CO2 emissions are imposed and energy costs are climbing [36]. Since modern production processes are highly automated nowadays, robots are also used on a large scale. As a consequence, the share of electric energy consumed by robots in production processes is rising and becoming more impactful.

1.1. Motivation

It is therefore of great interest to reduce the energy consumption of industrial robots (IR). Common methods to implement this are energy-efficient motion planning, optimizing IR operating parameters or scheduling IR operations [31]. Most of these methods require the creation of a model of the robot on which optimization can be performed. Different methods and tools are used for model building of IRs concerning energy demand. The methods reviewed have in common, that they are mostly based on the creation of traditional physical models, which require the identification of physical parameters as well as expert knowledge to create these models [22, 38, 31]. An alternative to traditional approaches for the modeling of physical systems is to use data-driven approaches, such as artificial neural networks (ANN), and to utilize their approximation capabilities [35, 8].

1.2. Thesis Objective

The aim of this work is to develop a methodology to model the behavior of a dynamic system with respect to its energy consumption. This shall be done on the example of an industrial robot. The focus will be on data-driven modeling methods, where different techniques should be applied and their performance evaluated. To aggregate the data needed for model creation, a suitable solution for the measurement of relevant data shall be developed and implemented.

1.3. Solution Approach

The basic modeling workflow that guides this thesis is shown in Figure 1.1. The first step is to obtain data with relevant information about the system behavior. This data is then preprocessed to be suitable for the chosen model type and structure. The model is then trained and subsequently its performance is evaluated. If performance is adequate, the model can be deployed and used in production. However, modeling satisfactory models usually requires several iterations because the initial model usually does not produce the desired results.

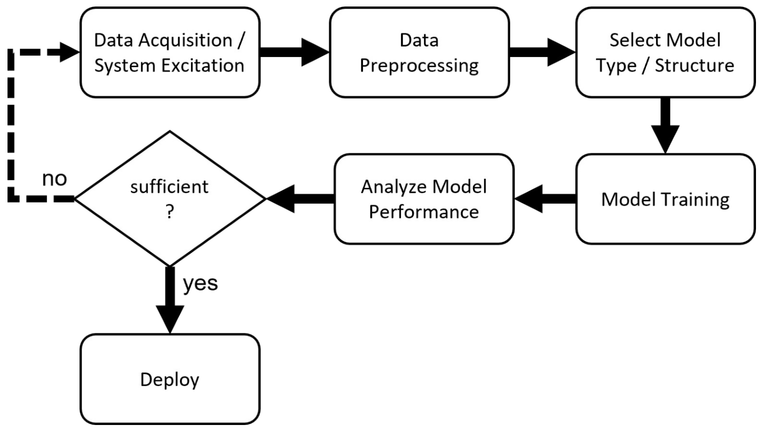

Data Acquisition / System Excitation
Data Preprocessing
Select Model Type / Structure
Model Training
Analyze Model Performance
sufficient ?
no
yes
Deploy

Figure 1.1.: Basic modeling workflow

1.4. Structure of the Thesis

In Chapter 2, the underlying boundary conditions to this thesis are outlined. In doing so, the structure of the existing production line, in which the robot is integrated, is shown. Furthermore the technical specifications of the robot are presented. Chapter 3 gives an overview of related studies and relevant state of the art techniques are revealed. Subsequently, in Chapter 4, the development and implementation of a measurement concept for the acquisition of the relevant data is described. The technical details for acquisition and storage are shown and the underlying functionality principles are described. Chapter 5 shows, how appropriate trajectories, which can be used for the excitation of the robotic system, are generated and executed. In Chapter 6 the analysis and processing of the measurements is shown and a method for compensation of invalid time information is presented. Building upon this, in Chapter 7 the techniques used for modeling are explained and the derivation of different model architectures is shown. The derived models are then evaluated in Chapter 8 with respect to their performance based on appropriate metrics. After modeling and model evaluation, a selected model is used in Chapter 9 to demonstrate the application of this model to a real use case. Here the derived model is transformed to match standardized exchange schemes and its functionality is demonstrated within a selected simulation platform. Finally in Chapters 10 and 11 the found results are reflected upon and summed up.

2. Boundary Conditions

This thesis was carried out as a research project at the Digital Factory Vorarlberg Research Center. The Digital Factory develops and tests various digital methods in cooperation with partners from industry and academia. Examples of important research topics covered are: (i) production and control, (ii) automation, (iii) IT security, (iv) methods of digital twins. The research center also has a laboratory where the developed methods can be applied and tested. This laboratory includes a complete production line in model size and consists, among other things, of a milling machine for processing raw materials, an automatic material transport unit and several collaborative robots, which are used for the assembly and handling of tools and workpieces. Figure 2.1 shows the basic structure of the production line.

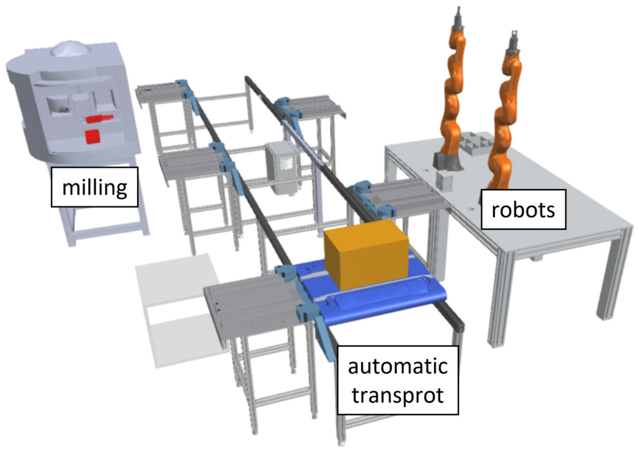


Figure 2.1.: Structure of Laboratory Setup of Digital Factory

A research topic currently being addressed at the research center is the creation of models that can predict the energy consumption of production processes - so-called digital energy twins. This thesis is intended to develop the methodology for creating such digital energy twins and to evaluate it on the real example of a industrial robot.

2.1. Robot System

The system of which the energy consumption should be modeled is the seven axis collaborative industrial robot LBR iiwa R800 from the company KUKA Germany GmbH. The main components of the robotic system are shown in Figure 2.2.

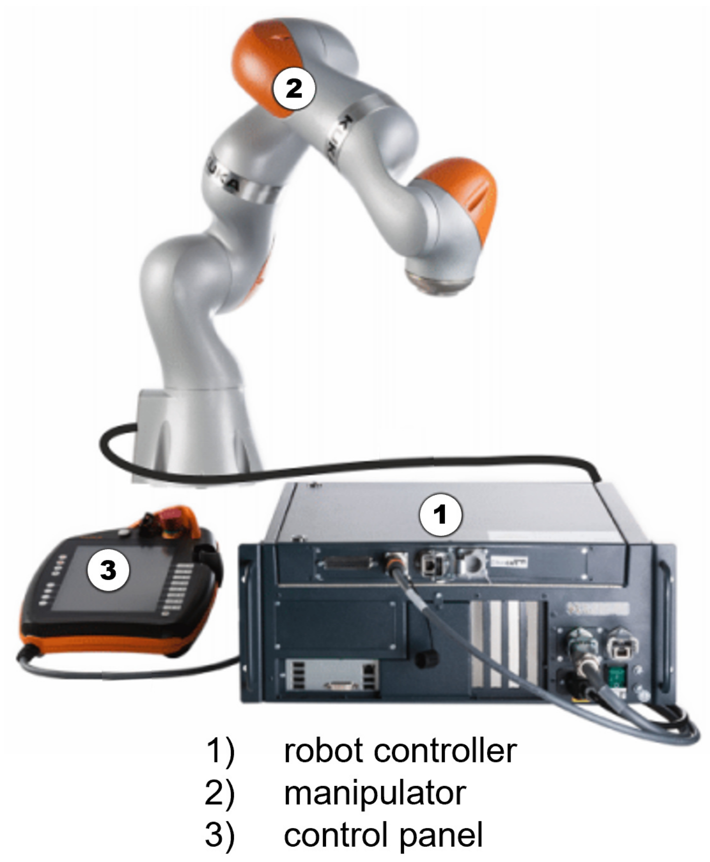


Figure 2.2.: Robotic system (source: [20], adapted by author)

The specifications relevant for this thesis are shown in Table 2.1.

manipulator	
number of axes	7
weight	approx. 23.9 kg
rated payload	7 kg
maximum reach	800 mm
axis data	
motion range A1	±170°
motion range A2	±120°
motion range A3	±170°
motion range A4	±120°
motion range A5	±170°
motion range A6	±120°
motion range A7	±175°
speed with rated payload A1	98 °/s
speed with rated payload A2	98 °/s
speed with rated payload A3	100 °/s
speed with rated payload A4	130 °/s
speed with rated payload A5	140 °/s
speed with rated payload A6	180 °/s
speed with rated payload A7	180 °/s
robot controller	
rated supply voltage	110V / 230V AC, single-phase
mains frequency	60 / 50 Hz
rated power input	1 kVA
thermal power dissipation	370 W

Table 2.1.: Specifications of KUKA iiwa R800 robot (source: [21, 20])

For the creation of robot applications and motion programming, the development environment KUKA *Sunrise.Workbench* is used. Functionalities offered by the software include the following:

- programming robot applications in Java
- managing projects and programs
- editing and managing runtime data
- project synchronization
- remote debugging
- creating I/O configurations

For programming robot applications, the KUKA *RoboticsAPI* is available within the development environment. This API is an object-oriented Java interface for controlling robots and provides the functionalities for motion planning, execution and also for data recording.

3. Related Work

Some efforts have been made to model the energy requirements of industrial processes. The modeling approaches can thereby be categorized in: (i) Analytical modeling where energy demand is simply modeled by analyzing different operation modes (e.g. on, off, idle ...) and taking the average energy consumption over each mode, (ii) Empirical modeling that commonly uses multiple linear regression to fit a predefined mathematical expression, which is linear in the models parameters, (iii) physical modeling where fundamental physical relationships are expressed in mathematical equations (iv) machine learning methods, which utilizes data-driven algorithms to identify relationships between input and output data [36].
According to Walther and Weigold [36], the most common methods for modeling energy demand of manufacturing processes are Analytical and Empirical modeling. For Analytical modeling, oversimplification can be considered as a possible drawback, while for empirical modeling it can be difficult to find the right mathematical expression that fits the collected data. Physics-based models can be very accurate, but are often difficult to implement and require a high level of insight and expertise about the problem to be modeled. Methods based on machine learning have recently become more important due to advances in automation and sensor technology. One advantage of machine learning methods is that they can learn relationships of inputs and outputs purely from data [36].
Concerning energy consumption modeling for IRs, the methods used in the reviewed works are mostly based on the development of a physical model, which requires the identification of parameters for robot dynamics and parameters describing power losses due to, e.g. friction, copper losses in the motors or losses in the control system. For dynamic parameter identification usually torque measurements must be obtained [34, 37, 1]. However, for some industrial robots, it is not trivial to determine the torque of the robot axis, which is addressed in [22], where the dynamic robot parameters are identified based only on the total electrical power consumption. In [31], modeling is performed using the Modelica language, while dynamic parameters are obtained from documentation and related studies.
However, limited research has been done so far concerning data-driven energy modeling for industrial robots. Zhang and Yang [40] use a feedforward neural

network to determine the operating parameters of maximum velocity and acceleration with a focus on minimizing total energy consumption on a given path and type of motion. Yin [39] et al. used a deep neural network for modeling and optimizing the energy consumption of a robot on a simplified case study. In general, energy modeling using machine learning techniques has provided promising results and is increasingly applied in industrial manufacturing processes. However, regarding energy modeling for IRs, more research needs to be done in in terms of case studies on real applications and by applying different machine learning algorithms in order to compare their performance. This work is intended to address this problem and to contribute with the derivation of an energy model for an IR.

4. Data Acquisition

To be able to derive data-driven models, it is necessary to first acquire meaningful data that holds relevant information of our System behavior. Relevant data concerning the problem statement of this thesis is:

- the robot axis angles as a function of time $\theta(t)$
- the consumed electrical active power $p(t)$

The developed measurement concept for obtaining data from both the electrical and mechanical domains is shown in Figure 4.1. The illustration shows the general structure and working principle.

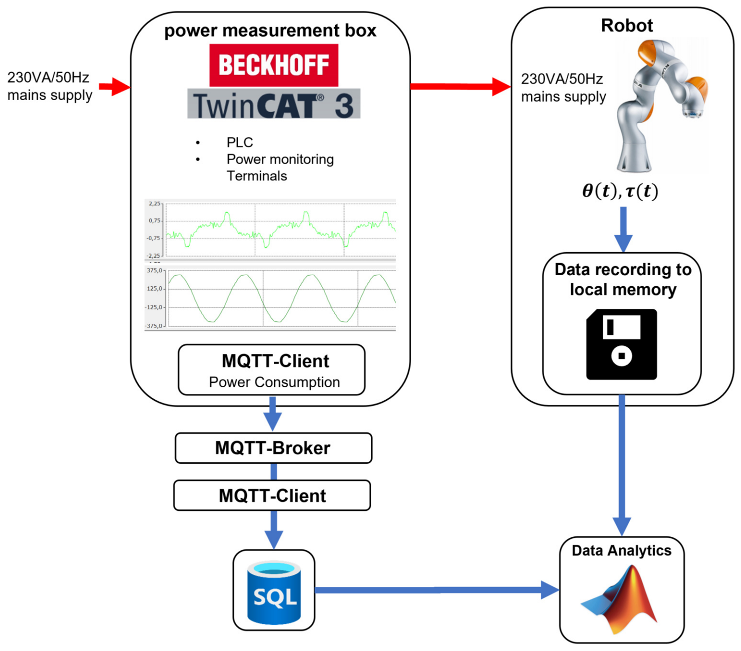


Figure 4.1.: General structure of the measurement concept

The flow of electrical energy is indicated by red arrows from left to right. It can be seen that the electrical supply to the robot is routed through an additional *power measurement box*, which measures the energy consumed by the robot.

For the measurement of mechanical data, functions of the KUKA *RoboticsAPI* are used. With these functions it is possible to save motion data directly on the robot controller in the form of a log-file. The data stream is marked by blue arrows. The collected electrical data is distributed using a suitable network protocol (MQTT) and written to a central storage (SQL database). The collected data from the database and the controller are afterwards available for further analysis. In the following sections a more detailed description of the technical implementation for the measurement of electrical and mechanical data is given.

4.1. Electrical Data Acquisition

For the purpose of power measurement, a device in form of a box was built. This measurement box can determine the electrical power consumption of any arbitrary device that is powered over a CEE 7/3 socket with mains voltage of 230VAC/50Hz and a maximum input current of $I_{rms} = 5A$. In the following subsection, the technologies used as well as the hardware and software components utilized are briefly described and relevant specifications are listed.
After that, the technical implementation is described.

4.1.1. Overview of Used Technologies

Technologies:

- **MQTT**
  The Message Queuing Telemetry Transport protocol (MQTT) is a lightweight publish/subscribe transport protocol suitable for Machine to Machine (M2M) and Internet of Things (IoT) applications. The protocol runs over TCP/IP, or over other network protocols. It is an open standard maintained by the organization OASIS. The specifications of MQTT v3.1.1 were also adopted by joint Technical Committee ISO/IEC JTC1 into the Standard ISO/IEC 20922:2016. [18, 29, 28]

- **EtherCAT**
  EtherCAT is a real-time Industrial Ethernet technology. The EtherCAT protocol is based on the Ethernet technology and is suitable for hard and soft real-time requirements in automation technologys. The protocol is standardized in IEC 61158. [11]

- **SQL**
  SQL is a programming language designed for manipulate data in relational databases. The language is standardized in ISO/IEC 9075. [33]

- **JSON**
  The JavaScript Object Notation (JSON) is a language-independent data format to express data objects as human readable lists of properties (name/value pairs). [12]

Hardware components:

- **EtherCAT Bus Coupler**
  Is a product by beckhoff automation to connect EtherCat Terminals to EtherCat networks.
  Specs:

Connectors:	2 x RJ45
Data transfer rate:	100 Mbit/s
power supply:	24VDC

  Further technical specifications can be found in [2]

- **power monitoring terminal**
  Is a product by beckhoff automation for power monitoring of a 3-phase AC voltage system. It is capable to measure six channels (voltage and current) simultaneously with a temporal resolution of $50\mu s$.
  Specs:

conversion time:	$50\mu s$
measured values:	3 x current, 3 x voltage
Measuring voltage:	ULx-N: max. 400VAC
Measuring current:	max. 5 AAC
resolution:	16 bit
measuring error:	0.2% relative to full scale value

  Further technical specifications can be found in [3]

- **Desktop PC**
  Standard desktop PC suitable to run required software components, e.g. a Soft-PLC.
  Specs:

Processor:	Intel(R) Core(TM) i7-6700T CPU @ 2.8GHz
memory:	16GB
storage:	500GB
operating system:	Windows 10 Education
Network Connection	RJ45
Network Controller	Intel Ethernet Controller suitable for EtherCAT (see [4])

  Further technical specifications can be found in [17]

A list of the used Software Components can be found in Appendix A.1

4.1.2. Technical Implementation

In Figure 4.2 the simplified circuit diagram of the measurement box is shown. The main electrical components are placed inside an enclosure to protect them

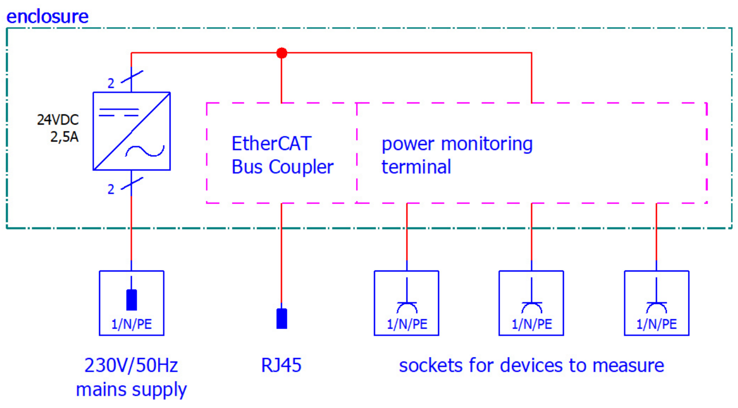


Figure 4.2.: Simplified circuit diagram of measurement box
(A detailed electrical schematic can be found in A.2)

from external influences and to shield potentially dangerous voltage-carrying components from outside. The power supply with 230V/50Hz is provided by a CEE 7/4 plug. Inside the box there is an 24VDC/2.5A power supply for powering an EtherCAT bus-coupler and a special power monitoring terminal. To the power monitoring terminal three CEE 7/3 sockets are connected, which can be used to supply a device whose energy consumption should be monitored. For the bus connection a CAT6 patch cable with RJ45 connectors is attached to the EtherCAT bus-coupler.
The tasks for data-acquisition, communication and data-preparation are handled by a Soft-PLC which is running on a Desktop-PC. On the Soft-PLC a MQTT-Client is set up that publishes the acquired data to a MQTT-Broker for distributing the acquired data to various applications. Figure 4.3 shows the framework consisting of measurement box and Desktop-PC that can acquire and distribute the desired data. Furthermore, two use cases are shown:

- **general use case:**
  The energy consumption of up to three devices is measured simultaneously and the data is distributed via a MQTT-Broker to any client that is subscribed to the topic under which the data was published.

- **robotic use case:**
  Only the industrial robot under investigation is connected for energy measurement and the collected measurement data is saved directly in an SQL database via a client application directly connected to the MQTT-Broker.

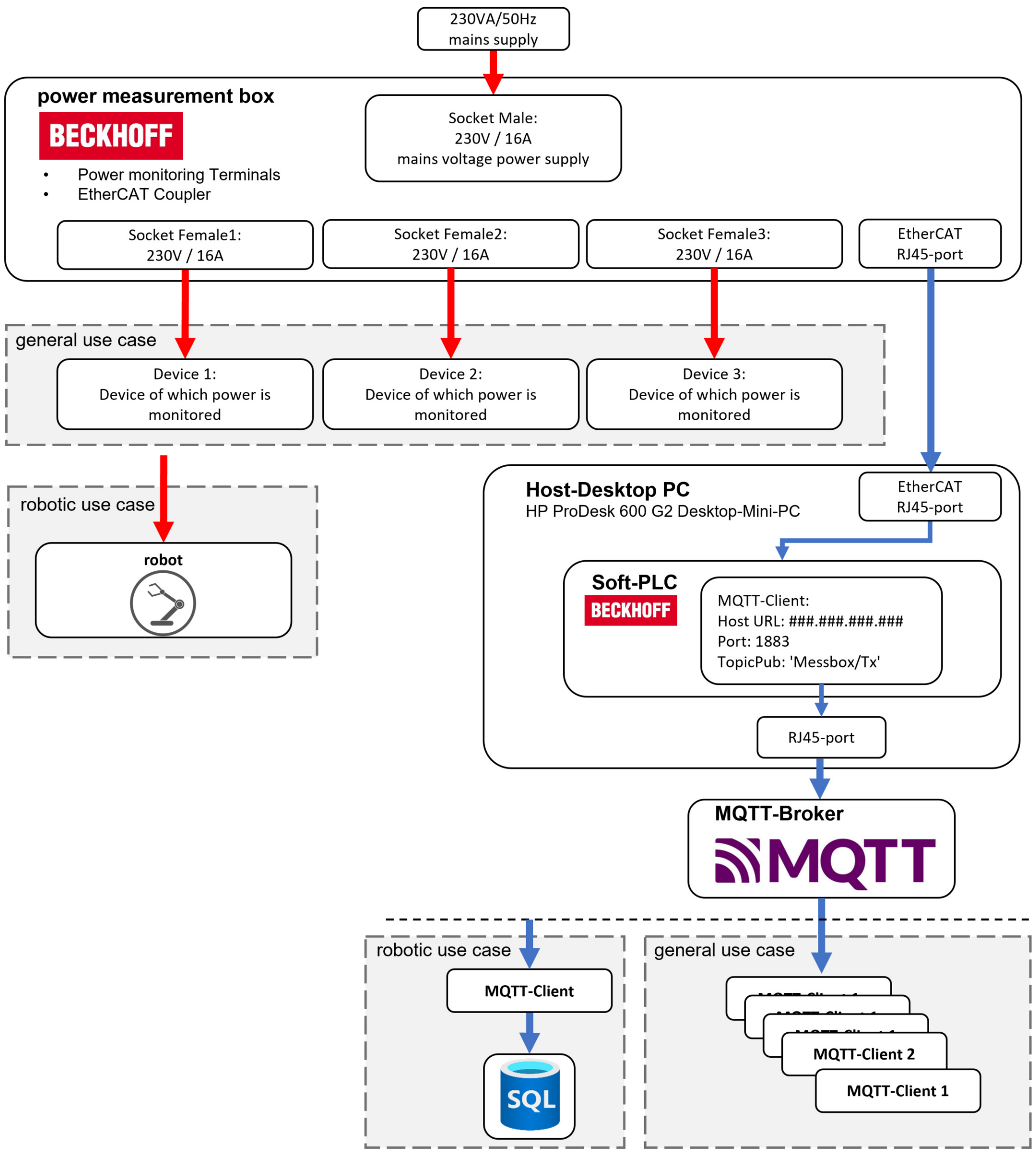

230VA/50Hz mains supply
power measurement box
BECKHOFF
Power monitoring Terminals
EtherCAT Coupler
Socket Male: 230V / 16A mains voltage power supply
Socket Female1: 230V / 16A
Socket Female2: 230V / 16A
Socket Female3: 230V / 16A
EtherCAT RJ45-port
general use case
Device 1: Device of which power is monitored
Device 2: Device of which power is monitored
Device 3: Device of which power is monitored
robotic use case
robot
Host-Desktop PC
HP ProDesk 600 G2 Desktop-Mini-PC
EtherCAT RJ45-port
Soft-PLC
BECKHOFF
MQTT-Client:
Host URL: ###.###.###.###
Port: 1883
TopicPub: 'Messbox/Tx'
RJ45-port
MQTT-Broker
MQTT
robotic use case
MQTT-Client
SQL
general use case
MQTT-Client 2
MQTT-Client 1

Figure 4.3.: Framework measurement box and PC

The used power monitoring terminal measures voltage $v(t)$ and current $i(t)$ with a sampling frequency of $20000\frac{samples}{second}$. This data is acquired and handle by two PLC-tasks. A fast task "MAIN_FAST", which operates at an interval of $1ms$ and thereby takes the samples from the power monitoring terminal, scales the collected data to represent physical units and saves the data to a buffer. The cycle time of the second task "MAIN_SLOW" is chosen to be twice the period of the mains power supply voltage which leads to $\Delta T = 40ms$. In this task the buffer values are taken to calculate the metrics listed in Table 4.1. The metrics are calculated out of 800 measurement values, which is a result of the $40ms$ cycle time and $20000\frac{samples}{second}$ sampling frequency. At the Start of every "MAIN_SLOW" cycle, a timestamp is generated according to the NTFS file system time format [10]. At the end of every "MAIN_SLOW" cycle, the program "MQTT_CONNECTOR" is called. This program first initializes a connection with an MQTT-Broker and, if connected, formats the calculated values to a JSON message which is then published over MQTT and subsequently saved to into a SQL-Database to be accessible for further use.

Description	**Symbol**	**Unit**
root-mean-square Voltage	V_{rms}	V
root-mean-square Current	I_{rms}	A
active power	P	W
apparent power	S	VA
power factor	PF	$-$
frequency	f	Hz
consumed energy	W	kWh

Table 4.1.: Calculated electrical quantities

Figure 4.4 illustrates the program operation, the complete program structure in form of a class diagram can be found in Appendix A.3.

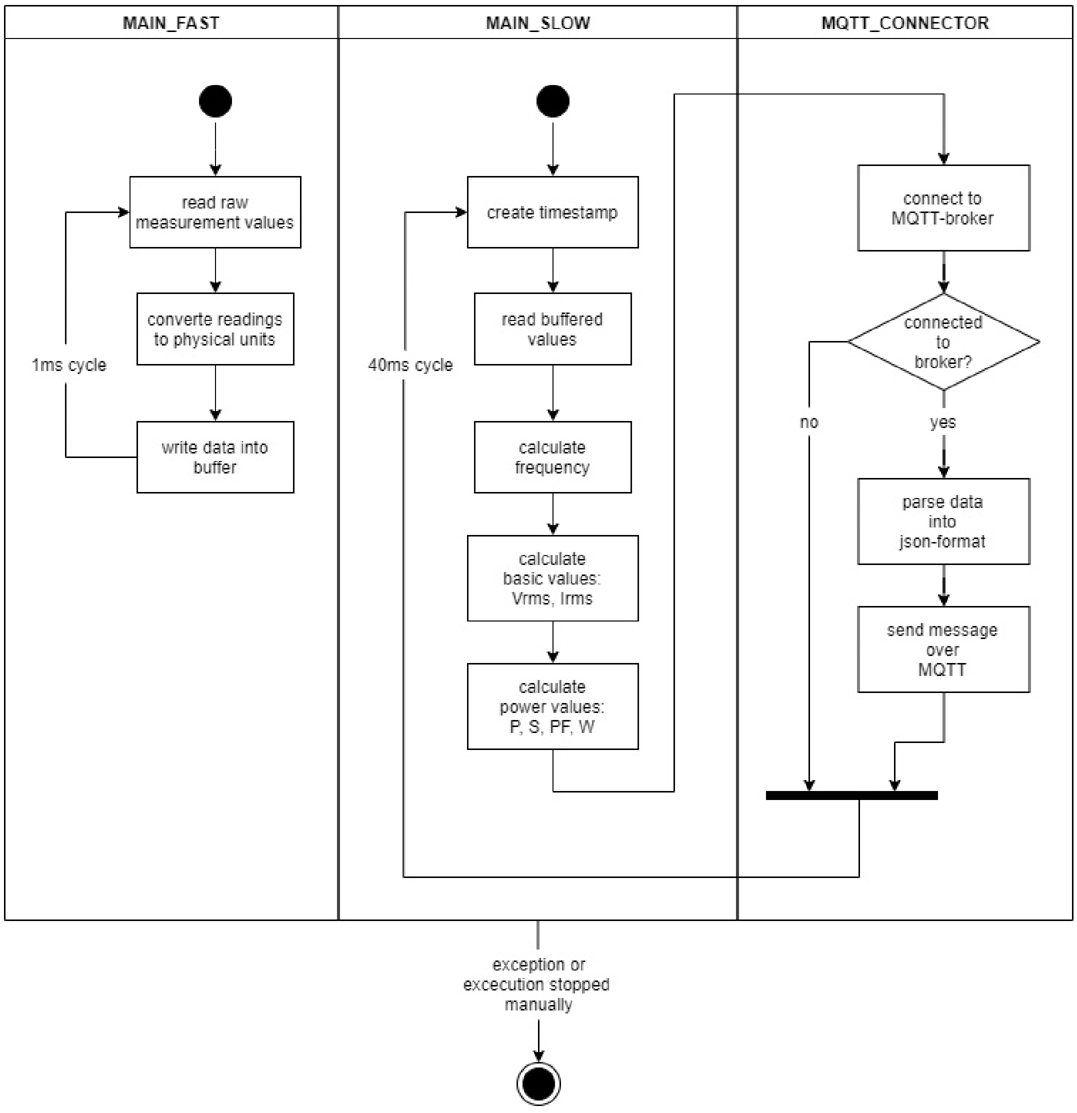

MAIN_FAST
MAIN_SLOW
MQTT_CONNECTOR
read raw measurement values
converte readings to physical units
write data into buffer
1ms cycle
create timestamp
read buffered values
calculate frequency
calculate basic values: Vrms, Irms
calculate power values: P, S, PF, W
40ms cycle
connect to MQTT-broker
connected to broker?
no
yes
parse data into json-format
send message over MQTT
exception or excecution stopped manually

Figure 4.4.: State chart PLC-program

4.2. Mechanical Data Acquisition

For mechanical data acquisition, the *DataRecorder* class form the API "com.kuka.roboticsAPI.sensorModel" is used. This class enables the recording of specific data at a given sampling interval. The recorded data is saved in a file and stored on the robot controller. For data recording, first an object of type DataRecorder must be created and parameterized. Parameters which can be set are:

fileName	File name under which the recorded data is saved
timeout	max. Recording duration
timeUnit	Time unit for the recording duration
sampleRate	Recording rate (unit: ms)

A Unix timestamp with milliseconds resolution is generated for each data sample. To add the data which should be recorded, several methods of the DataRecorder class are available. Since we are interested in the joint angles and joint torques, the following methods are used:

addInternalJointTorque(...)	Recording of the measured axis torques
addCurrentJoinPosition(...)	Recording of the axis actual positions

First, the DataRecorder object must be activated via the enable() method. Recording is then started via the startRecording() method.
Listing 4.1 shows the code for setting up the data recording. For the measurement setup the recording duration is restricted to 400sec and a sampleRate of 40ms is chosen.

Listing 4.1: Setup DataRecorder object

```
/*
instantiate data recorder with max recording time of 400s
and sampling time 40ms
*/
DataRecorder rec_1 = new DataRecorder(
                       "13_07_21_randSigCollCheck01.csv",
                       400, TimeUnit.SECONDS,
                       40);
// log joint torques
rec_1.addInternalJointTorque( lbr_iiwa_7_R800_1);
// log joint angles
rec_1.addCurrentJointPosition( lbr_iiwa_7_R800_1,
                               AngleUnit.Radian);
rec_1.enable(); // enable recording
rec_1.startRecording(); // start recording
```

5. Trajectory Generation and Execution

Sufficient excitation is necessary to obtain an accurate model. In this chapter, two distinct methods are presented to generate trajectories suitable for excitation purposes, and it is shown how these trajectories can be executed on the real system. Methods chosen for trajectory generation are:

- Limited Space Random
- All Space Random

For limited space trajectories, the generated positions on which the robot is supposed to move are restricted to a subspace of all reachable manipulator poses. Advantages on using just a subspace of all possible poses are (i) that it is easier to realize robot movements without the concern of possible collisions and (ii) the reduced complexity of the data leads to easier realization of models representing the system behavior. But models obtained from limited space data are likely to not perform well outside the provided training data by extrapolation. To develop models that generalize well over all possible robot movements and to test the limitations of the models obtained by limited space data, the All Space Data is generated as well. For generation of this data it is important to consider physical constrains of the robot and its environment to prevent demolitions. The following sections show the generation of different trajectories and how they can be executed on the real system.

5.1. Limited Space Random

For the Limited Space Random trajectories, a volume in the manipulators workspace is defined. The chosen volume has cubic shape and is parameterized with the center point $cP = (cP_x = 0.6m, cP_y = 0m, cP_z = 0.34m)$ and the edge length $L = 0.3m$. Figure 5.1 schematically shows the defined volume within the robots workspace.

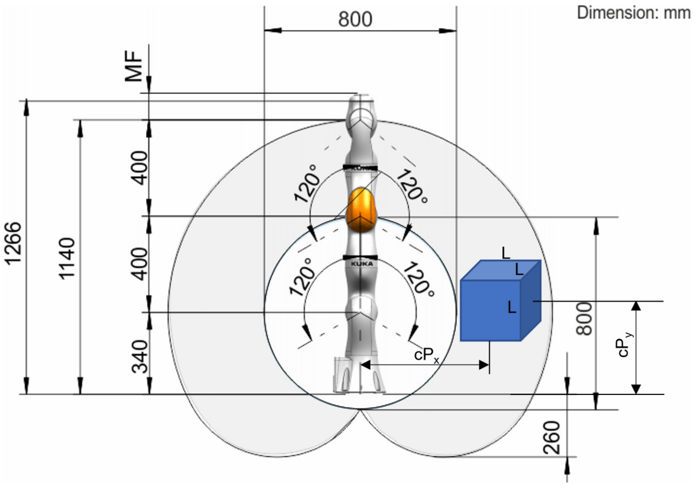


Figure 5.1.: Limited Space Random: Cubic Space

Within the cubic volume, N random and uniform distributed positions (P_x, P_y, P_z) are generated. Where the number of points N is chosen as uniform distributed integer within the interval $[minN = 40, maxN = 60]$. The orientation of the end-effector is chosen to stay constant with the Z-Axis pointing down which can be represented with the rotation-matrix $Ree = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$. The desired robot configurations, which are the seven axis angles of the robot $q = (q1, \ldots, q7)$, can be obtained by solving the inverse kinematics problem. This is done by utilizing the inverse kinematic solver within the MATLAB extension *Robotics System Toolbox*. The process of generating one trajectory is shown in Algorithm 1.

Algorithm 1 generate random poses in limited space

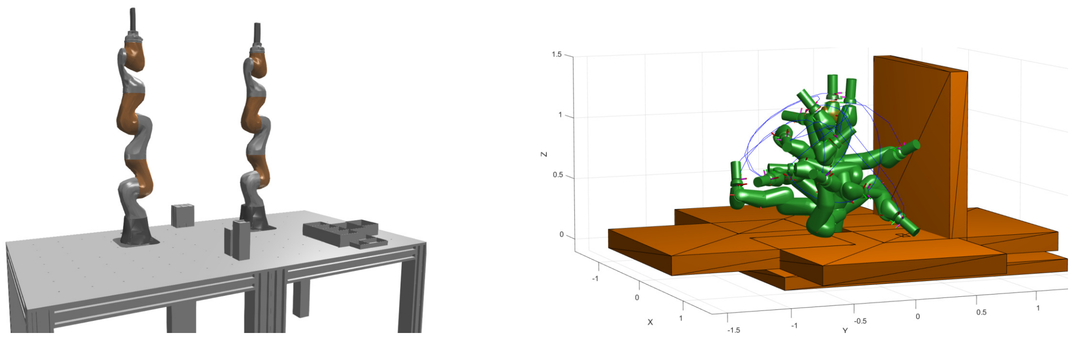
```
 1: $minN \leftarrow 40$    : min. No. of points
 2: $maxN \leftarrow 60$    : max. No. of points
 3: $[cP_x, cP_y, cP_z] \leftarrow [0.6m, 0m, 0.34m]$    : cube center point
 4: $L \leftarrow 0.3m$    : cube edge length
 5: $Ree \leftarrow \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$    : end-effector orientation (z down)
 6: $N \leftarrow$ random int with $N \in [minN, maxN]$
 7: $P_x \leftarrow N$ random double values with $P_x \in [cP_x - L/2, cP_x + L/2]$
 8: $P_y \leftarrow N$ random double values with $P_y \in [cP_y - L/2, cP_y + L/2]$
 9: $P_z \leftarrow N$ random double values with $P_z \in [cP_z - L/2, cP_z + L/2]$
10: for $i \leftarrow 1$ to $N$ do
11:     $q[i] \leftarrow invKin(Ree, P_x[i], P_y[i], P_z[i])$    : solve inverse kinematics
12: end for
```

Using this method, 46 different trajectories were generated and saved as csv-files, which serve for execution of the robot motion.

5.2. All Space Random

The method chosen to generate robot configurations in *All Space* is to generate random points in the robot's configuration space represented by the seven axis angles $q = (q_1, \ldots, q_7)$. To obtain positions that the robot can reach without damaging itself or surrounding objects, several constraints must be considered. First, all generated axis angles must lie within the limits specified by the robot manufacturer. In addition to this simple condition, it must be checked if there are collisions at the generated positions or on the path between them. Collisions must be checked between the robot and the environment as well as with the robot itself. For collision checking, the functionalities of MATLAB *Robotics System Toolbox* are used, which allows checking for collisions between predefined geometry meshes. It supports defining simple collision objects such as boxes and cylinders or to create convex mesh geometries from a list of 3D vertices. A model of the robot LBR iiwa R800 used in this thesis, including collision meshes, is already included in the used toolbox and can be applied for collision avoidance. To model the regions where collisions can potentially occur, simple collision boxes are created. The real areas that are delimited by the virtual collision boxes are: the assembly table, various installations mounted on the table and a neighboring robot.

In Figure 5.2 the real installation is shown on the left and on the right the generated collision boxes as well as the collision meshes of the used robot shown in some random poses.



(a) assembly line setup

(b) assembly line setup with virtual collision boudaries

Figure 5.2.: Assembly line setup with- and without collision boundaries

Algorithm 2 shows the procedure for generating the axis positions in All Space. The number of points to be generated is chosen as uniformly distributed random integer in the interval $[minN = 10, maxN = 20]$. The smaller number of points is due to the longer execution time of one movement, since the distances between two points in All Space are wider. A starting position q_{Start} is chosen with all axis angles set to zero. All points are now created one after the other. First, a new axis position is randomly generated, taking into account the specified limitations. A temporary trajectory is now created with N_{temp} positions between the newly generated point and the previous point. The generation of the temporary trajectory is simply done by a linear interpolation between the axis angles of the two positions. This results in a so-called PTP (point-to-point) movement. Now, it is checked if there are collisions at any point on the temporary trajectory. If a collision occurs, the point is discarded and a new one is generated. If the movement between the two points is collision-free, the next point can be generated.

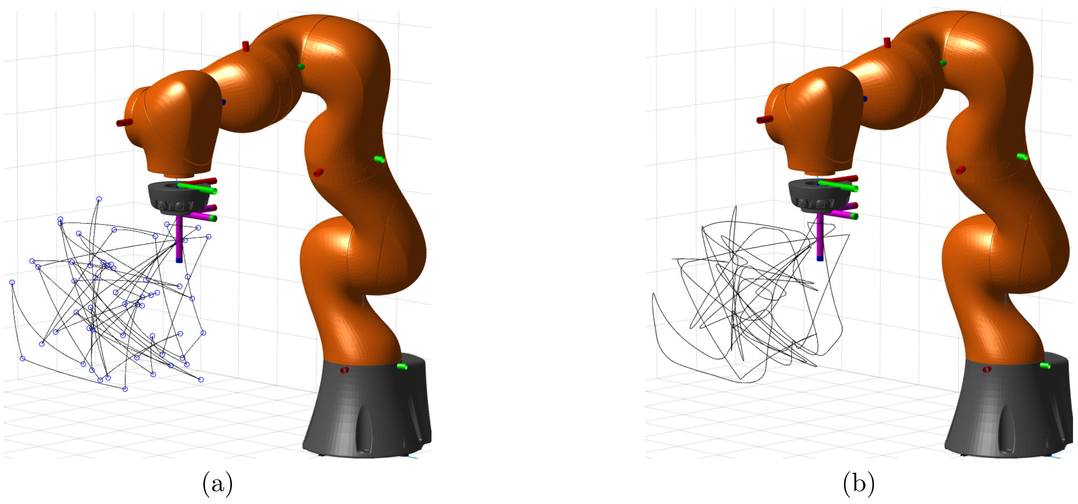
Algorithm 2 generate random poses in all space

```
1: $minN \leftarrow 10$    : min. No. of points
2: $maxN \leftarrow 20$    : max. No. of points
3: $q_{min} \leftarrow (q_{1min}, \ldots, q_{7min})$    : axis min. limits
4: $q_{max} \leftarrow (q_{1max}, \ldots, q_{7max})$    : axis max. limits
5: $q_{Start} \leftarrow (0°, 0°, 0°, 0°, 0°, 0°, 0°)$    : start position
6: $N \leftarrow$ random int with $N \in [minN, maxN]$
7: $N_{temp} \leftarrow 10$    : No. of temporary points
8: $q[1] \leftarrow q_{Start}$
9: for $i \leftarrow 1$ to $N$ do
10:     $hasCollision \leftarrow True$    : flag indicating collision
11:     while hasCollision do
12:         $q[i+1] \leftarrow$ random double values with $q \in [q_{min}, q_{max}]$
13:         $q_{temp} \leftarrow N_{temp}$ evenly spaced points from $q[i]$ to $q[i+1]$
14:         $hasCollision \leftarrow$ check for any collisions at positions $q_{temp}$
15:     end while
16: end for
```

With this method, 30 different trajectories were generated and saved as csv-files, which serve for execution of the robot motion.

5.3. Trajectory Execution

For executing the created trajectories on real hardware, a robot application is written that can be executed by the robot controller. The application loads one predefined trajectory form a csv-file into the controller and executes a ptp-motion between the given positions. To get additional variability in the data, each motion is executed with random blending factor and random velocity. Blending is set as a relative value in "%". A Relative Blending factor of 20% means that blending starts when 80% of the distance in cartesian space between two positions is reached. A Relative Blending factor of 0% has no effect. Relative Blending is chosen to be uniformly distributed between 0% and 50%. In Figure 5.3 the effect of blending is shown.



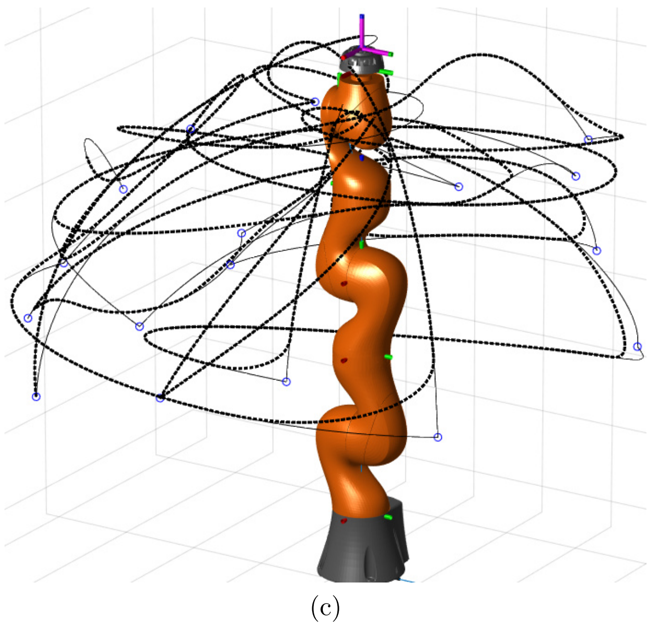


Figure 5.3.: Effect of relative blending factor: (a): example of Limited Space Random points indicated with circles and the ptp-motion between points as solid line; (b): recordings when trajectory (a) is executed with blending; (c): example of All Space Random points indicated as circles, ptp-motion as solid line, recorded motion as dotted fat line

Relative Velocity is set to be between 20% and 50% of the maximum Angular velocities of each axis. Velocity is also randomly chosen for every motion. Axis acceleration is chosen to be constant at 20% of the maximum possible acceleration. A code sample from the robot application for executing a ptp-motion with random motion parameters is shown in Listing 5.1.

Listing 5.1: Code sample for random motion

```
// instantiate random number generator
Random randomNum = new Random();
double minBlend = 0;  // lower limit of random blending
double maxBlend = 0.5;  // upper limit of random blending
double minVel = 0.2;  // lower limit of random rel velocity
double maxVel = 0.5;  // upper limit of random rel velocity
double relAcc = 0.2;  // set maximum rel acceleration

// instantiate Joint Positions for async movement
JointPosition[] J = new JointPosition[3];
J[0] = new JointPosition(7);
J[1] = new JointPosition(7);
J[2] = new JointPosition(7);

// ptp-movement
// get random blending factor
randRelBlend =
  minBlend + (maxBlend-minBlend)*randomNum.nextDouble();
// get random relative velocity
randRelVel =
  minVel + (maxVel-minVel)*randomNum.nextDouble();
// set Joint Positions to desired values
J[0].set( Math.toRadians(jointPositions.get(pointIdx)[0]),
      Math.toRadians(jointPositions.get(pointIdx)[1]),
      Math.toRadians(jointPositions.get(pointIdx)[2]),
      Math.toRadians(jointPositions.get(pointIdx)[3]),
      Math.toRadians(jointPositions.get(pointIdx)[4]),
      Math.toRadians(jointPositions.get(pointIdx)[5]),
      Math.toRadians(jointPositions.get(pointIdx)[6]));
// move to position
lbr_iiwa_7_R800_1.moveAsync(
    ptp(
      lbr_iiwa_7_R800_1.getForwardKinematic(J[0])
      ).setBlendingRel(randRelBlend)
      .setJointVelocityRel(randRelVel)
      .setJointAccelerationRel(relAcc)
      );
// get random blending factor
// get random relative velocity
// set Joint Positions to desired values
// move to position
// .....
```

The effect of random blending factor and velocity are visualized in Figure 5.4, which shows two recordings of the same trajectory. It can be seen that the underlying shape of the curves appears similar but with noticeable variations.

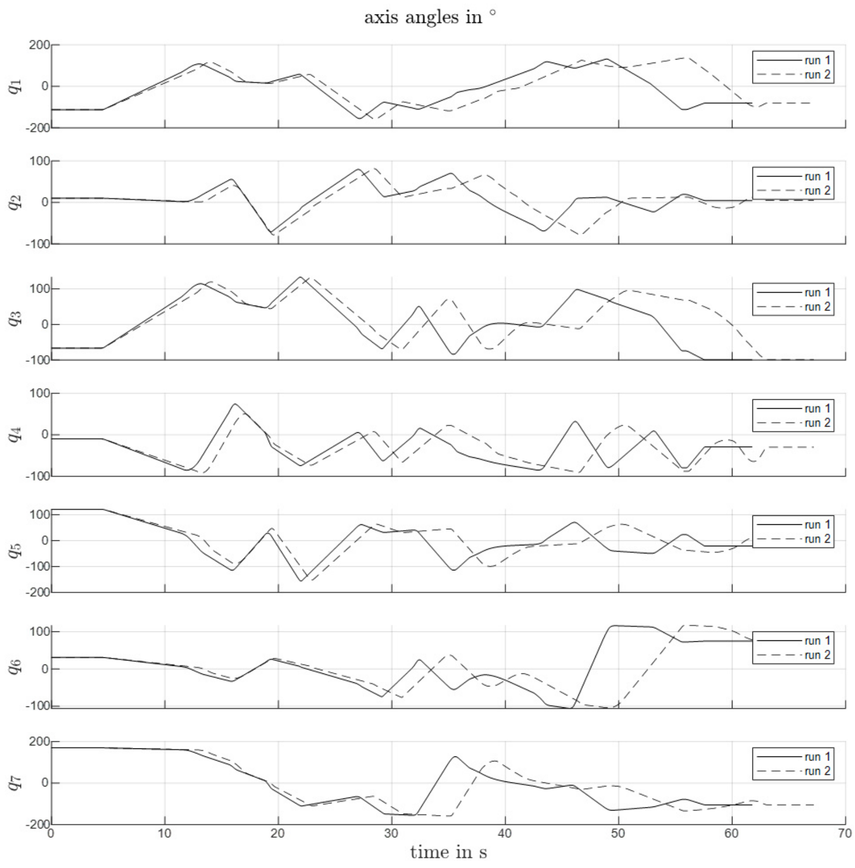


Figure 5.4.: Effect of random blending and velocity

The variation of execution parameters makes it possible to produce more data out of the same source trajectories. With this method, 202 recordings were made with Limited Space Data, which corresponds to a total recording time of $2.8 hours$. For All Space, 135 recordings were made with a total recording time of $2.7 hours$. In parallel to the execution of trajectories, the power consumption was recorded as well. The obtained data is the basis for deriving models through further analysis and processing.

6. Data Preparation and Analysis

This chapter describes the procedures for data preparation for the purpose of making the collected measurements accessible for model building. The data was recorded by two different systems. The electrical data is measured by the measurement box (Section 4.1) and the mechanical data is aggregated by the robot controller. Each of these systems operates independently, resulting in both systems generating their own time stamps for a recorded data point. It is therefore very unlikely that the time information of the measurements of both systems will match, so that an electrical measurement value can be precisely assigned to one motion measurement at a given instance. For this reason, a method is required for merging the measured values of both systems to a common time vector. Furthermore, it is to be investigated whether and to what extent the temporal information of the measured values deviates from each other and how a possible deviation can be corrected.
In the next sections, first the merging of the measurement series will be shown, then the processed data will be further analyzed to determine possible deviations in the temporal information between the electrical and mechanical measurement series and, if necessary, how to compensate them.

6.1. Data Merging

For each executed trajectory, one log file is stored on the robot controller, while power measurements of the measurement box are taken in parallel and spanning multiple executions. In order to merge the measurement series, the time stamps of the two systems are first converted to a common format. Afterwards, the start and end time is extracted from each recording form the robot controller. With this information, the measurement values of the measurement box can be split into measurement series, which are located within the time span of one recording from the controller.
The resulting data sets are merged using the Matlab function *synchronize()*, which allows synchronizing time tables to a common time vector and resample the data in the given data sets. With this function, the collected data is sampled

to a new time vector with a sampling interval of $40ms$. For times in the time vector that do not match row times from the input data, the data is linearly interpolated. In Figure 6.1 the effect of data merging is illustrated.

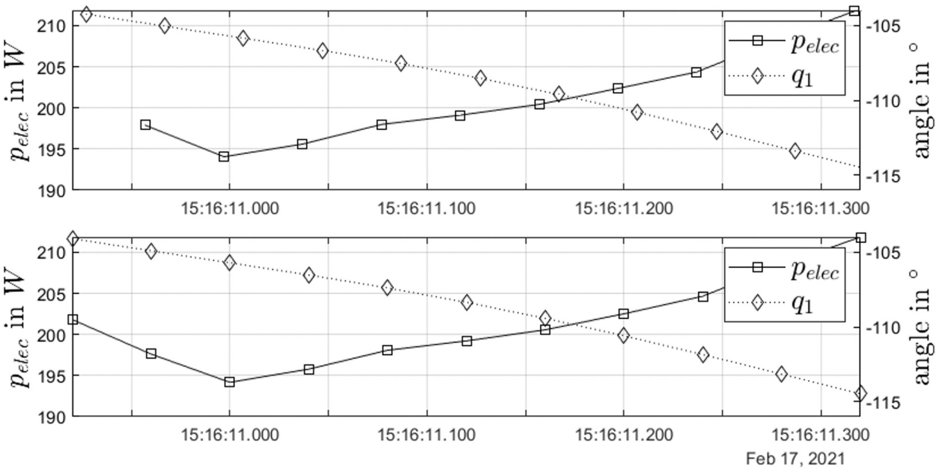


Figure 6.1.: Resampling of data: upper plot shows electrical and mechanical data recordings, timestamps are indicated by diamonds and squares; lower plot shows the same data resampled to a common time vector

The processed data is the basis for the time shift investigation presented in the next section.

6.2. Time Shift Investigation

In this section, we investigate how the time series fit together by observing distinctive patterns in the data. The aim is to find out whether the data series are shifted in time to each other or differ in their duration. A time offset could be caused by unsynchronized clocks whereas a different duration could be caused by varying clock frequencies.
A successful approach for comparing the timing of electrical and mechanical data is to calculate the mechanical power and compare it with the electrical power. The mechanical power p_{mech} is calculated by multiplying the angular velocity $\dot{\theta}$ of the axes with the torque τ acting on the motor shaft of the axes Equation 6.1.

$$p_{mech} = \tau \cdot \dot{\theta} \tag{6.1}$$

The torque needed to calculate p_{mech} is measured directly during execution, this is possible as the robot used has integrated torque sensors. Angular velocity can be calculated out of the measured axis angles by calculating the first order centered finite difference Equation 6.2.

$$\dot{\theta}(t) \approx \frac{\theta(t+\Delta T) - \theta(t-\Delta T)}{2\Delta T} \tag{6.2}$$

The total mechanical power p_{mech}^{tot} is calculated by determining the mechanical power of each axis and summing up over all results.

$$p_{mech}^{tot}(t) = \sum_{i=1}^{7} \tau_i(t) \cdot \dot{\theta}_i(t) \tag{6.3}$$

In Figure 6.2 the calculated mechanical power of a selected measurement is compared with the corresponding measured electrical power. It can be seen, that in the electrical power measurement there is a high initial peak. This can be attributed to the inrush currents of the axes motors when the breaks of the axes are released. Furthermore, it can be seen that apart from the initial peak, the mechanical power is indeed similar in amplitude and course. However, it also indicates a considerable shift in time between mechanical and electrical power.

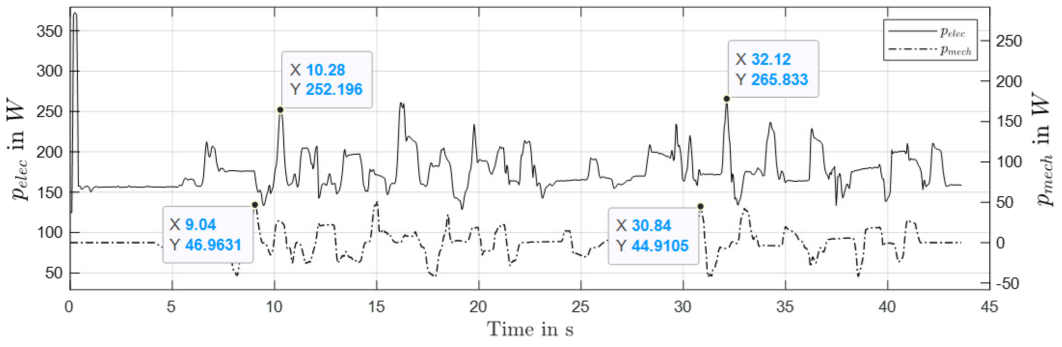


Figure 6.2.: Comparison of mechanical and electrical power

To estimate how much the data is shifted and if the data is also stretched, two time points are extracted from each time series at prominent points that seem to correspond to each other. The calculated time spans between the measured points are:
$32.12s - 10.28s = 21.84s$
$30.84s - 9.04s = 21.80s$
which shows a difference by only $40ms$, which corresponds to one tick at the sampling rate used. This difference suggests that time stretching is unlikely

to be a concern. In contrast, the difference between corresponding data points leads to:
$10.28 - 9.04 = 1.24$
$32.12 - 30.84 = 1.28$
That is, the two sequences seem to be shifted to each other by over one second. Some delay in the course of the sequences can be expected due to the internal dynamics, but the shift observed here is to large in magnitude. Hence, the time difference in the acquired data is likely to be due to the non-synchronized clocks of the systems. A proper procedure would be to synchronize the clocks using a suitable method (e.g. precision time protocol ptp). Actually, there are plans to perform a synchronization by connecting the measuring box and the robot via EtherCat and synchronizing the clocks via the ptp protocol. Unfortunately, we were not able to perform this synchronization within the scope of this work.
All in all, sampling seems reliable and equidistant for both systems, but time information has overall offset. This overall offset makes a time shift compensation (see Section 6.3) necessary for fully aligning the two time series.

6.3. Time Shift Compensation

To determine the time shift, also referred to as *lag*, the cross-correlation of the two time series was calculated within a suitable range using the Matlab function *xcorr()*. The result of this function can be interpreted as measure of similarity between a vector x and shifted (lagged) copies of a vector y as a function of the lag. Cross-Correlation at lag m is calculated with [26]:

$$\hat{R}_{xy}(m) = \begin{cases} \sum_{n=0}^{N-m-1} x_{n+m} y_n^*, & m \geq 0 \\ \hat{R}_{xy}^*(-m), & m < 0 \end{cases} \tag{6.4}$$

The time difference to compensate for was determined as the value at which the cross-correlation becomes maximal. Figure 6.3 shows exemplary a cross correlation plot. The range chosen to shift one time series over the other is $\pm 2.4s$. With a sampling interval of $\Delta T = 40ms$, this corresponds to ± 60 time samples, as indicated on the x-axis in the figure.
The obtained shift is used to adjust the data so that the sequences match in their temporal domain. Figure 6.4 shows an example of the original and the shifted data. The sequences are shown in the time domain as well as a scatter plot for visualizing the correlation.

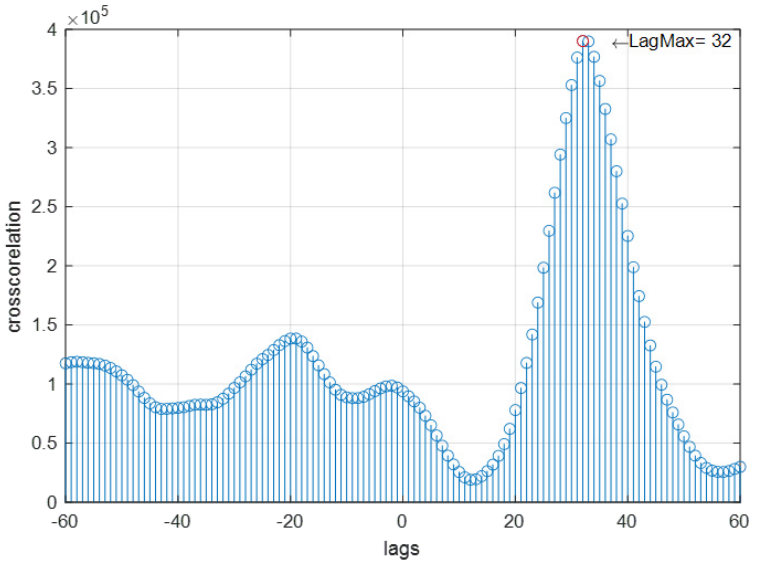


Figure 6.3.: Crosscorrelation p_{mech} and p_{elec}

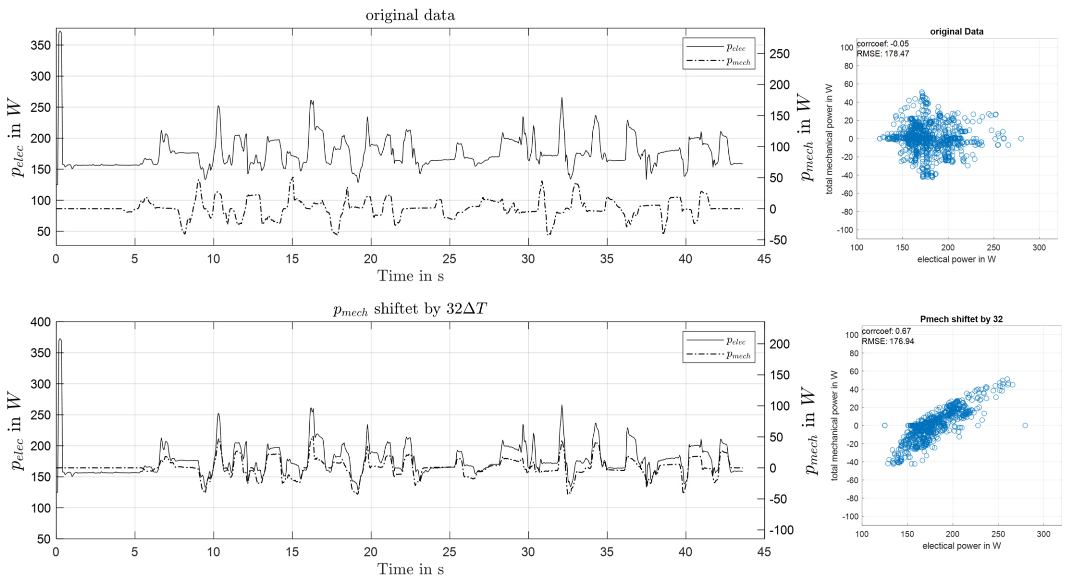


Figure 6.4.: Comparison of shifted and non-shifted sequences

This method allows for compensating the observed timeshift. However, synchronizing clocks would eliminate the need for this procedure. Moreover, this method is only feasible since torque measurements are available.

7. Data-Driven Modeling

In this chapter, we present the development of different data-driven models based on the collected and preprocessed data. The used modeling approaches are based on machine learning techniques as well as neural networks and we specifically investigated model structures which can be categorized in:

- Long Short Term Memory Neural Network (LSTM) with manual feature engineering
- LSTM with Convolutional-Layers for feature extraction

Several variations are trained and tested for each of these structures, where the implementation of the models is realized using the MATLAB *Deep Learning Toolbox*. In the next sections, first a brief introduction to the machine learning techniques used is given. After that, the preprocessing and feature engineering of the data is described. This is followed by a description of the creation of the selected network structures and finally the chapter concludes with the necessary procedure for the training of the models.

7.1. Basics of used Technologies

This section presents the basic principles of the methods used for data-driven model building.

7.1.1. Feedforward Neural Network

The feedforward (FF) neural network (NN) is a so-called static network, which means that the output depends only on the current inputs and not on previous ones. Although FF-Networks are not able to represent temporal behavior, they can still be used to model simple dynamical systems [35, 5]. The basic building block of a neural network is a neuron that takes an input vector x, multiplies it by weights w, and passes the result to a summer which is also fed by a bias b. This weighting and summing of the inputs can also be written as a matrix multiplication of the input vector x with the weight matrix W. The output of the summer z goes into a nonlinear activation function f, which generates the neuron output y. Without the activation function the network could only learn affine transformations, but by introducing this non-linearity it is possible to learn more complex behavior [9]. Common activation functions are, the *sigmoid function*, $\sigma(x) = \frac{1}{1+e^{-x}}$, and the *hyperbolic tangent function*, $tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$ [32]. Figure 7.1 shows a single neuron with an R dimensional input vector. In general, a single neuron is not sufficient to mimic complex

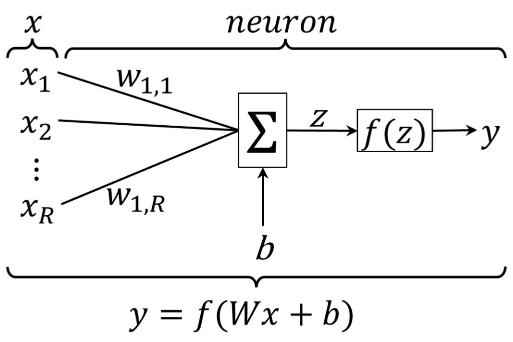


Figure 7.1.: Single neuron (adapted from [15] and [32])

behavior. Hence, in a typical feedforward NN, multiple neurons are stacked in parallel to form a *layer*, and multiple layers may be connected in such a way that information is passed from one layer to another. Figure 7.2 shows a feedforward NN with three layers. Each layer consists of S neurons having their own weights, biases, and activation functions. To distinguish between layers, the variables are labeled with superscripts indicating the layer number. Different layers can have different numbers of neurons. In the example shown,

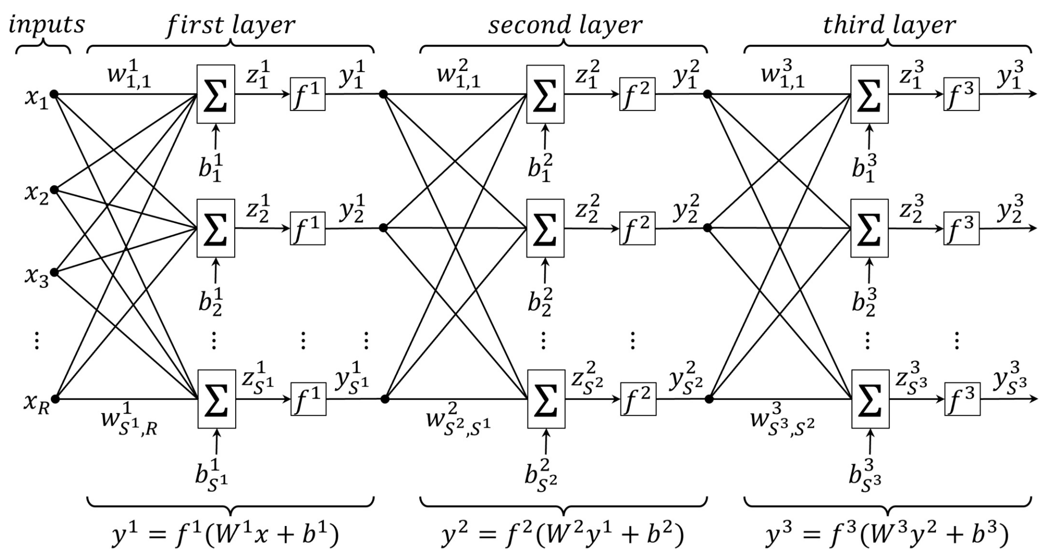


Figure 7.2.: Multi layer network (adapted from [15])

there are S^1 neurons in the first layer, S^2 neurons in the second layer, and so on. The last layer, whose output is the output of the network, is called *output layer*, and the layers between the input and output layer are called *hidden layers*.

7.1.2. LSTM

The long-short term memory (LSTM) neural network is a special type of recurrent neural network (RNN). RNNs have recurrent states, which makes it possible to learn time-dependent dynamical behavior. RNNs have widely been used in areas of dynamic system identification and control of nonlinear systems [30, 35, 5]. The advantage of LSTMs over conventional RNNs is that they avoid the problem of vanishing or exploding gradients during the training process by using gated units. Figure 7.3 shows the structure of one LSTM layer. The recurrent states are the *hidden states* h and the *cell states* c. The output y of the LSTM layer at a certain time step t is equal to the hidden state at this time step ($y_t = h_t$). The states contain information from previous time steps. In every step, information is added or removed from the states which is controlled by so-called *gates*[24].

The learnable parameters of the LSTM are the input wights W, the recurrent weights R and the biases b. The computation of the states is given by Equations

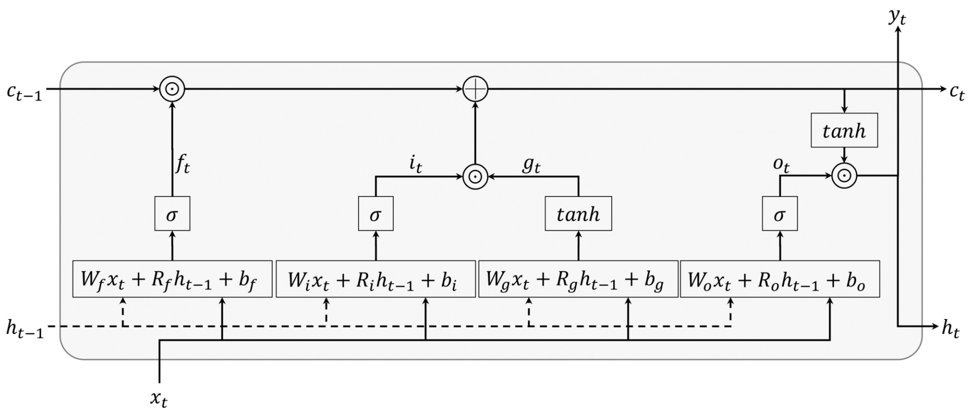

c_{t-1}
c_t
y_t
h_{t-1}
h_t
x_t
f_t
i_t
g_t
o_t
σ
$tanh$
$W_f x_t + R_f h_{t-1} + b_f$
$W_i x_t + R_i h_{t-1} + b_i$
$W_g x_t + R_g h_{t-1} + b_g$
$W_o x_t + R_o h_{t-1} + b_o$

Figure 7.3.: LSTM layer (adapted from [32] and [24])

7.1 to 7.7 where $\odot$ denotes the Hadamard product, which is the element-wise multiplication of vectors.

$$i_t = \sigma(W_i x_t + R_i h_{t-1} + b_i) \qquad \text{input gate} \qquad (7.1)$$
$$f_t = \sigma(W_f x_t + R_f h_{t-1} + b_f) \qquad \text{forget gate} \qquad (7.2)$$
$$g_t = tanh(W_g x_t + R_g h_{t-1} + b_g) \qquad \text{cell candidate} \qquad (7.3)$$
$$o_t = \sigma(W_o x_t + R_o h_{t-1} + b_o) \qquad \text{output gate} \qquad (7.4)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \qquad \text{cell state} \qquad (7.5)$$
$$h_t = o_t \odot tanh(c_t) \qquad \text{hidden state} \qquad (7.6)$$
$$y_t = h_t \qquad \text{output} \qquad (7.7)$$

7.1.3. Convolutional Layer

The convolutional layer is the core layer of Convolutional Neural Networks (CNNs), which are widely used for image-related tasks, such as object detection or image recognition [32], but they are also highly relevant for sequence processing tasks were they can be competitive with RNNs on certain sequence-processing problems [9].
Convolutional layers are based on the mathematical operation of convolution, which can be viewed as the spatial filtering of an input matrix f with a filter matrix w, also referred to as *kernel*. The computation with a one-dimensional signal is given by Equation 7.8, where the kernel has size $1 \times n$, with n assumed to be odd. Variable b is a non-negative integer defined by $n = 2b + 1$, the input f has size $1 \times N$, and t denotes the index of the input elements. The

convolution is usually denoted by the asterisk operator $*$. The filtered output g is also called *feature map*.

$$g(t) = \sum_{s=-b}^{b} w(s) f(t-s) = w * f \tag{7.8}$$

In machine learning, correlation is often used instead of convolution, but it is also referred to as convolution [14]. This is also the case with the MATLAB machine learning toolbox in this work. However, in this thesis, convolution and correlation are treated separately, whereas a correlation is denoted with the operator $\circledast$. The calculation of the correlation is very similar to convolution where its one-dimensional calculation is given by Equation 7.9. The only difference in computation is a plus instead of a minus-sign, which can also be interpreted as rotating a convolutional filter kernel by 180° about its center.

$$g(t) = \sum_{s=-b}^{b} w(s) f(t+s) = w \circledast f \tag{7.9}$$

Figure 7.4 shows the operating principle of applying a kernel of size 1×5 to a one-dimensional input by correlation, thereby the kernel slides over the input and at each position an element-wise multiplication of the kernel elements, or weights, is performed and the results are summed to produce the output g. The size of the kernel is also called *receptive field* and the number of spatial increments by which a receptive field is moved is called *stride* [13]. The output

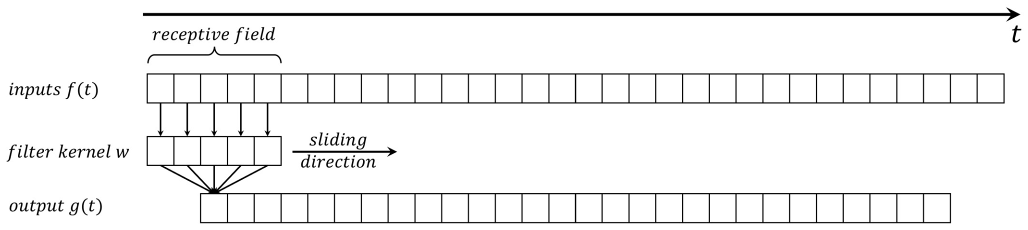


Figure 7.4.: Correlation 1-D (adapted from [9])

shown in Figure 7.4 is $4 = n - 1$-steps smaller than the input. This is because filtering cannot be performed on the edges of the input, since the receptive field would exceed the input length. This can be overcome by applying *zero padding*, where zeros are appended to the ends of the input to match the kernel size.
In convolutional layers for image processing, the two-dimensional correlation given by Equation 7.10 is applied. Here, (i, t) are the indices of the input

matrix of size $M \times N$, the kernel size is $m \times n$, and r is a non-negative integer with $m = 2r + 1$.

$$g(i,t) = \sum_{r=-a}^{a} \sum_{s=-b}^{b} w(r,s) f(i+r, t+s) = w \circledast f \tag{7.10}$$

Within a convolutional layer often more than one filter kernel is applied to the input, which also results in multiple feature maps as output. Figure 7.5 schematically shows the operation of a convolutional layer with a two-dimensional input and multiple filter kernels.

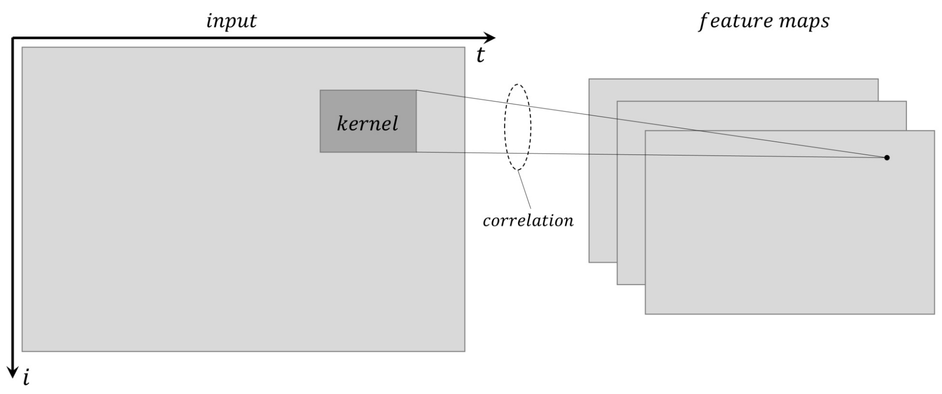


Figure 7.5.: 2-D convolutional layer with several filter kernels (adapted from [13])

The learnable parameters of a convolutional layer are the filter kernel weights.

7.1.4. Training Algorithm

The goal of network training is to minimize a loss function $E(\theta)$ that provides information about how well the network is performing. θ is a vector of network parameters containing weights and biases. A widely used loss function is the mean squared error (MSE) calculated between the actual output $\hat{y}$ and the predicted output y [32]. Equation 7.11 shows the calculation of the MSE for a one dimensional network output over N training samples.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{7.11}$$

The loss function is minimized by updating the network parameters θ in an appropriate way. One of the most popular algorithms to accomplish this are *gradient descent* algorithms, e.g. the standard gradient descent algorithm updates the network parameters by taking small steps in the direction of the negative gradient of the loss function (Equation 7.12).

$$\theta_{t+1} = \theta_t - \alpha \nabla E(\theta_t) \tag{7.12}$$

where t is the iteration number and $\alpha > 0$ the learning rate. For standard gradient descent, all training samples are used to compute the gradient of the loss function, which can lead to slow training progress on a large dataset [32]. Therefore, a commonly used variant is the *stochastic gradient descent* (SGD) algorithm, where gradient evaluation and parameter updating is performed on a subset of the training data referred to as a *mini-batch*. A complete run over the entire training data using mini-batches is called one *epoch*.
A more advanced variant of the SGD algorithm is the adaptive moment estimation (Adam) introduced in [19]. The computation of one update step is given in Equation 7.13, here m_t is the exponential moving average of the gradients and v_t is the exponential moving average of the element-wise squares of the gradients. Division by $\sqrt{v_t}$ is also performed element-wise and ϵ is a small constant to prevent division by zero. Parameters $\beta_1, \beta_2 \in [0, 1)$ are the gradient- and squared gradient decay factors.

$$\begin{aligned}
\theta_{t+1} &= \theta_t - \frac{\alpha m_t}{\sqrt{v_t} + \epsilon} \\
m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla E(\theta_t) \\
v_t &= \beta_2 v_{t-1} + (1 - \beta_2) \left[\nabla E(\theta_t)\right]^2
\end{aligned} \tag{7.13}$$

The element-wise division by $\sqrt{v_t}$ normalizes the update step for each parameter individually, which means that learning rates of parameters with large gradients are decreased and those with small gradients are increased. By using the moving average m_t, the gradients are accelerated in the direction that leads to convergence [32].

7.2. Data Preprocessing and Feature Engineering

The purpose of preprocessing is to make the existing raw data more usable for machine learning algorithms. For example, one important aspect is the normalization of the data. Machine learning algorithms should not be fed with large numbers, also non homogeneous data should be avoided. The reason being that this can prevent the chosen algorithm from converging. In general, the data should have the following characteristics [9]:

- take small values (e.g. range 0-1, mean of 0, standard deviation of 1)
- be homogeneous (all the data should take values in the same range)

Feature engineering is the process of utilizing the understanding of the problem to be solved and the machine learning algorithm to be used to improve the performance of the algorithm by applying transformations to the data before it is fed into the model [9].

7.2.1. Feature Engineering

To select suitable features, we will first take a closer look at the system shown in Figure 7.6. Our input to the system is the desired trajectory $\theta_{ref}(t)$, and the system response we are interested in is the electrical power supplied $p_{elec}(t)$. The desired trajectory goes as reference input to the robot controller, which ensures that the angles $\theta(t)$ of the manipulator axis follow the reference as closely as possible.

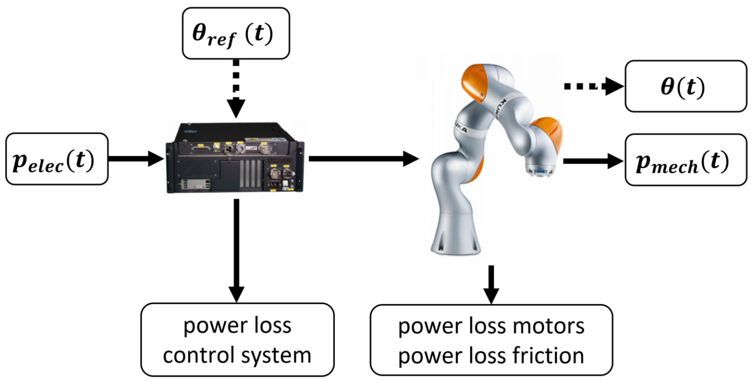


Figure 7.6.: Energy flow through the robot system

Portions of the total power supplied are lost due to losses in the control system, losses in the motors as well as through friction. What remains is the mechanical

power that can be used for performing various tasks.
As already shown in Section 6.2, the mechanical power is closely related to the electrical power and can be calculated by ($p_{mech} = \tau \cdot \dot{\theta}$). In robotics, the torque acting on a robot axis can be calculated by solving the so called inverse dynamics problem, which is given as a set of nonlinear differential equations in the variables angle θ, angular velocity $\dot{\theta}$ and angular acceleration $\ddot{\theta}$ [23].The calculation is shown in Equation 7.14

$$\tau = M(\theta)\ddot{\theta} + h(\theta, \dot{\theta}) \tag{7.14}$$

where $M(\theta)$ is a symmetric, positively defined mass matrix, and $h(\theta, \dot{\theta})$ are forces that combine centripetal, coriolis, gravitational, and frictional terms. By multiplying Equation 7.14 with $\dot{\theta}$, the mechanical power of the system under consideration is given by:

$$p_{mech} = \Big(M(\theta)\ddot{\theta} + h(\theta, \dot{\theta})\Big)\,\dot{\theta} \tag{7.15}$$

Hence, the mechanical power corresponds to a nonlinear function with the variables angle θ, angular velocity $\dot{\theta}$ and angular acceleration $\ddot{\theta}$. Since the electrical power is closely related to the mechanical power, and the mechanical power depends substantially on the three variables θ, $\dot{\theta}$, $\ddot{\theta}$, these variables can be considered as suitable features to model the system behavior via machine learning algorithms. It should be noted that θ, $\dot{\theta}$, $\ddot{\theta}$ refer to the actual motion of the manipulator and thus are characterized by the output trajectory of the system. However, the goal here is to predict the power consumption by means of the input trajectory. But the active control ensures that the output follows the input with small deviations. Therefore, the first and second derivatives of the input trajectory are also assumed to be suitable features. The following sections describe how to extract the desired features both manually and by using learned convolutional filter kernels.

7.3. LSTM with Manual Feature Extraction

In this section, first the necessary steps to extract the engineered features are described followed by the introduction of the selected network structures with the manually extracted features as input.

7.3.1. Feature Extraction and Normalization

To extract the features obtained in Subsection 7.2.1, the first and second order finite difference method is applied to the recorded data. Equations 7.16 and 7.17 show the calculation of the first respectively second order finite difference.

$$\dot{\theta}(t) \approx \frac{\theta(t+\Delta T) - \theta(t-\Delta T)}{2\Delta T} \tag{7.16}$$

$$\ddot{\theta}(t) \approx \frac{\theta(t+\Delta T) - 2\theta(t) + \theta(t-\Delta T)}{(\Delta T)^2} \tag{7.17}$$

Before the collected data is handed over to the selected modeling algorithms, normalization is performed. For the electrical power, normalization is chosen with $p_{norm} = (p_{elec} - p_{offset})/p_{scale}$. Where power normalization parameters are chosen with $p_{offset} = 155$ and $p_{scale} = 166$. Angle, angular velocity and angular acceleration are just scaled by the factors: $\theta_{scale} = 3$, $\dot{\theta}_{scale} = 1.6$, $\ddot{\theta}_{scale} = 13$. Figure 7.7 shows the effect of normalization, where the original data is shown on the left and the normalized values on the right.

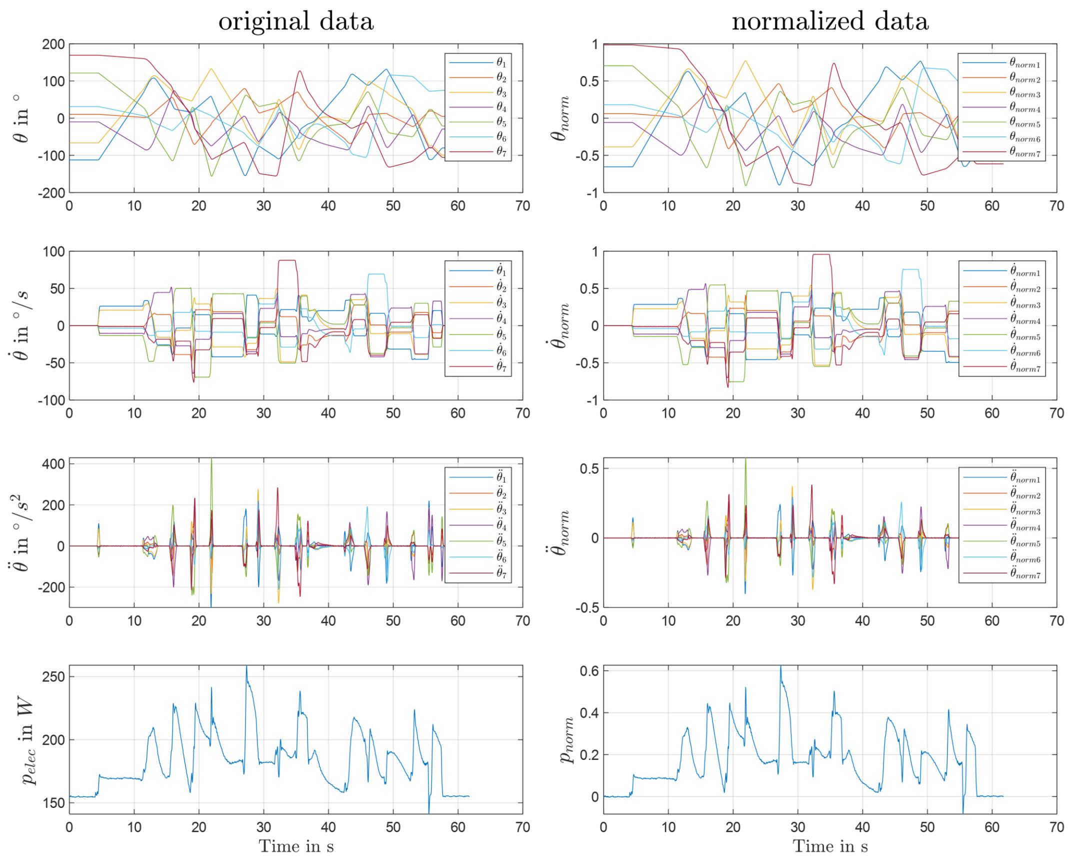


Figure 7.7.: Manual feature extraction and normalization: left recorded axis angles, calculated derivatives and measured power consumption, right the corresponding values

7.3.2. Network Structure Setup

The chosen base structure consists of a sequence input layer, followed by an LSTM layer with 10 hidden units, succeeded by a 4 layered FF-Network without activation functions. The last layer is a one-dimensional output layer. With this base structure, three networks were set up that differ only in their input dimension. One network has only the normalized seven axis angles as input, the second has additionally the normalized angular velocities as input, which leads to a 14-dimensional input, and the third network has all 21 constructed features as input. The base structure with its variations of input dimensions is shown in Figure 7.8, a more detailed description of the network structures can be found in Appendices B.1, B.2, B.3.

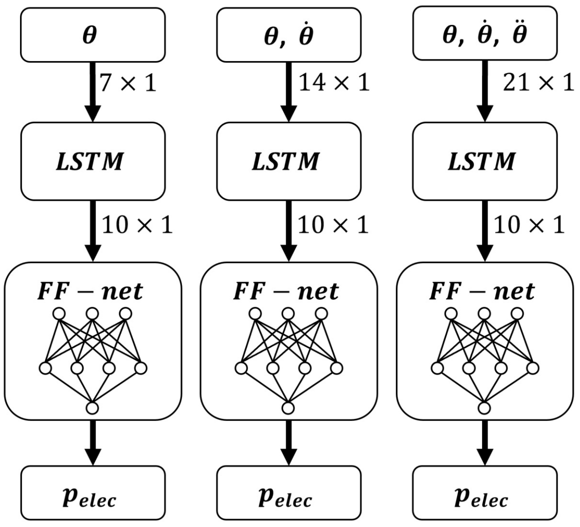


Figure 7.8.: Network structures with manually extracted features of varying input dimensions; from left to right: 7x1dim, 14x1dim, 21x1dim

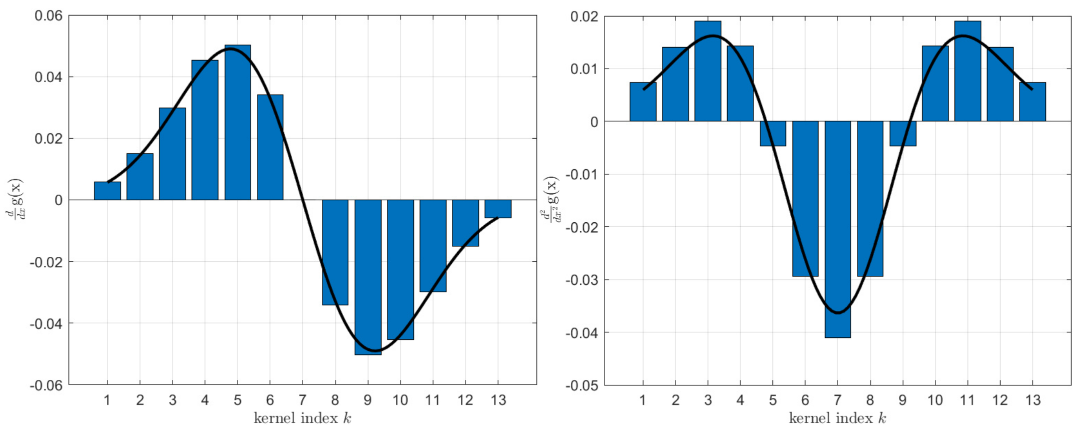
7.4. LSTM with Convolutional-Layer for Feature Extraction

For the networks in the previous section, the input features were extracted manually, which requires a deeper insight into the system we want to model. In this section, we will build a model structure that is capable of automatically extracting features from the data without requiring any prior knowledge of the system. A commonly used layer type for feature extraction in image processing is the convolutional layer, which can also be applied for the networks built in this section.
It is interesting to note, that the derivative of a signal can be computed by convolution of the signal with an appropriate filter kernel. Common filter kernels in the field of image processing are thereby the *Difference of Gaussian* (DoG) for the computation of the first derivative (Equation 7.18) and the *Laplacian of Gaussian* (LoG) for the second derivative (Equation 7.19) [6].

$$\frac{d}{dt}f(t) \approx DoG * f \tag{7.18}$$

$$\frac{d^2}{dt^2}f(t) \approx LoG * f \tag{7.19}$$

Figure 7.9 shows an example of one dimensional DoG and LoG kernels with a receptive field of $L = 13$.



(a) DoG 1-D (b) LoG 1-D

Figure 7.9.: Filter kernels for Difference of Gaussian (a) and Laplacian of Gaussian (b) with receptive field of $L = 13$

When the receptive field of the kernels is reduced to L=3 (Figure 7.10), then the convolution of the resulting kernels with a signal is the same as the calculation of the first and second finite differences given by Equation 7.16 and Equation 7.17 when $\Delta T = 1$.

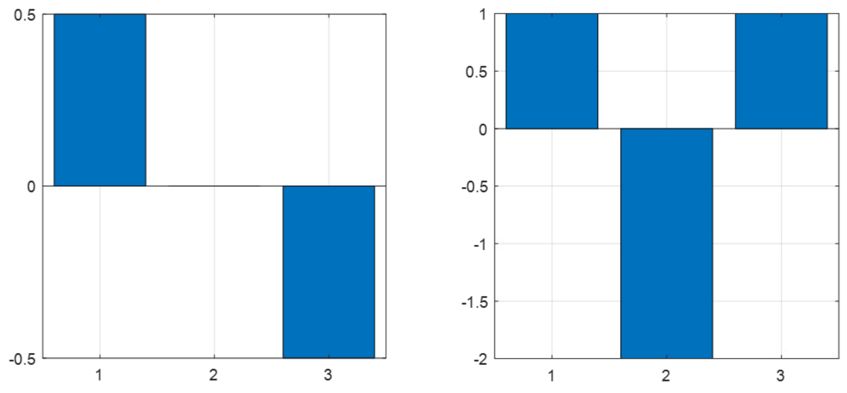


Figure 7.10.: Filter kernels equivalent to centered finite difference when $\Delta T = 1$: left DoG; right LoG

Hence, a convolutional layer with multiple filter kernels of length 3 prior to the LSTM layer can in principle provide the same performance as the networks with manually extracted features.

7.4.1. Sequence Transformation

Before using a convolutional layer in combination with an LSTM layer, the sequential input data must first be converted into a sequence of frames suitable for the receptive field of the applied filter kernels. This must be done so that the convolutional layer can take a frame at each time step to compute the correlation and form feature maps. These feature maps are then flattened into a vector so that the LSTM can process them.

Figure 7.11 shows this process on the example of a sequence of seven axes angles. Here, the receptive field is chosen to be $1 \times L$, where L is an odd integer and also the width of the frames. When applying correlation with Q filter kernels to one frame, with no zero padding, Q feature maps are extracted with size 7×1. After flattening the feature maps this leads to a $7Q \times 1$ feature vector which can then be fed into the LSTM layer.

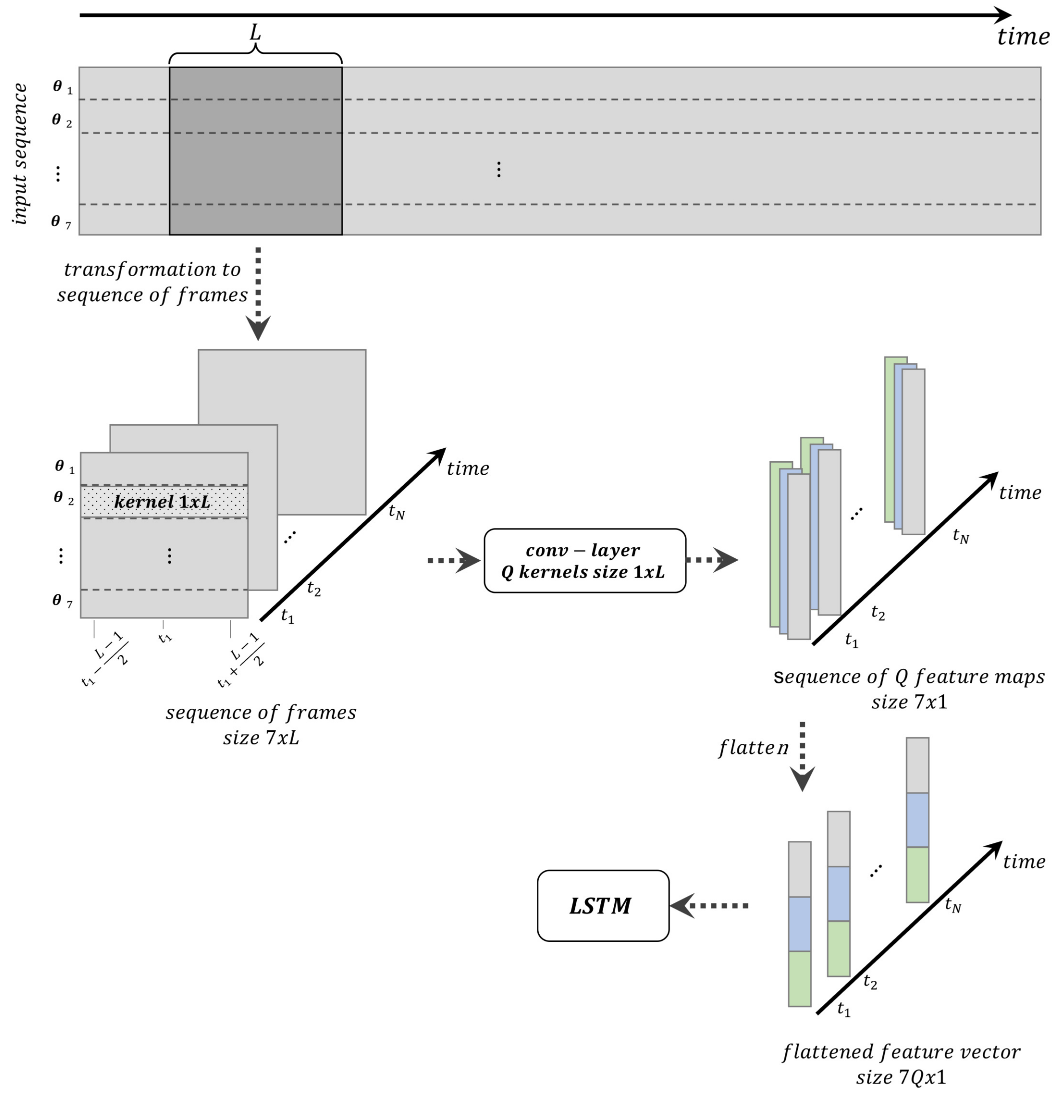

time
L
input sequence
θ 1
θ 2
θ 7
transformation to sequence of frames
kernel 1xL
time
tN
t2
t1
t1 − (L−1)/2
t1
t1 + (L−1)/2
sequence of frames size 7xL
conv − layer Q kernels size 1xL
time
tN
t2
t1
sequence of Q feature maps size 7x1
flatten
LSTM
time
tN
t2
t1
flattened feature vector size 7Qx1

Figure 7.11.: Sequence transformation and processing with convolutional layer

7.4.2. Network Structures Setup

The first structure chosen is a convolution layer with filters of size 1x3. The number of filters is chosen as 2x and 3x. The defined base structures with its variation is shown in Figure 7.12. For a more detailed description of the network structures, see B.4, B.5. In contrast to the networks with manual

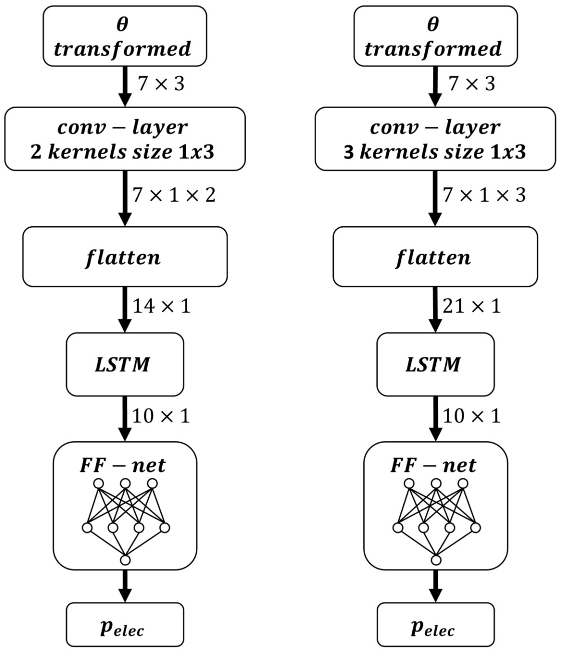


Figure 7.12.: Network structures 2conv1x3, 3conv1x3

feature extraction, the inputs for these networks are only the 7 axis angles. With the number of filters chosen to be 2 and 3, it is intended that the results are comparable to the models with manually extracted features.

In addition to the 1x3 networks, 3 further networks were built with a wider receptive field. One with 3x 1x5 kernels, one with 3x 1x11 and one with 5x 1x21. The setups are shown in Figure 7.13, a more detailed description of the network structures can be found in Appendices B.6, B.7, B.8.

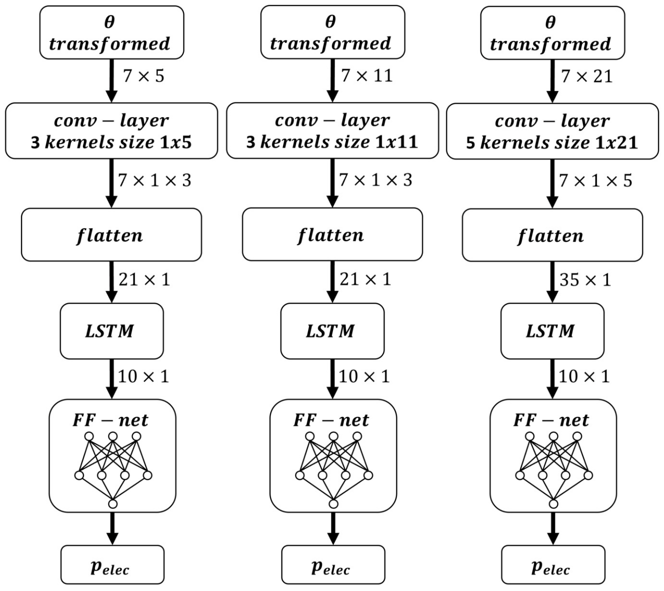


Figure 7.13.: Network structures 3conv1x5, 3conv1x11, 5conv1x21

7.5. Model Training

In the Sections 7.3 and 7.4, 8 different network structures were defined. Furthermore, data from two different types of trajectories were recorded, namely the Limited Space and the All Space data defined in Chapter 5. This enables comparing the generalization capabilities of the networks trained with those data sets. In addition, another training dataset is produced by combining the recordings of the two trajectory types mentioned above. This dataset is used to train models that are provided with as much information as possible to achieve the best overall performance.
Hence, there are in total 24 combinations of different training datasets with different model structures. In order to reduce the evaluation effort, not all of these combinations are obtained. In doing so, only the network types with manually extracted features are trained on each combination of the available datasets to investigate the generalization behavior. The rest of the set up network structures are trained using only the combination of data from all space and the

limited space. Thus, the performance comparison between models with manual feature extraction and those using convolutional layers is done with models trained on all available data. Table 7.1 shows the used combinations of network architectures and training datasets. For model training and evaluation,

Network Structure	Train Dataset		
	Lim. Sp.	All Sp.	Lim.+All Sp.
7x1dim	X	X	X
14x1dim	X	X	X
21x1dim	X	X	X
7x3dim 2conv1x3			X
7x3dim 3conv1x3			X
7x5dim 3conv1x5			X
7x11dim 3conv1x11			X
7x21dim 5conv1x21			X

Table 7.1.: Considered combinations of network structures and data sets

the collected datasets are split into training, testing, and validation datasets. The splitting ratios used are shown in Table 7.2. When splitting, we ensured that no recordings based on the same source trajectory were found in the same split. All models were trained using Adam training algorithm. The used train-

	total		training				validation				testing			
	No. Rec.	dur. [min]	No. Rec.	dur. [min]	No. ratio	dur. ratio	No. Rec.	dur. [min]	No. ratio	dur. ratio	No. Rec.	dur. [min]	No. ratio	dur. ratio
Lim. Sp.	202	170	149	126	74%	74%	24	18	12%	11%	29	25	14%	15%
All Sp.	135	162	99	120	73%	74%	18	21	13%	13%	18	21	13%	13%
Lim. + All Sp.	337	332	248	246	74%	74%	42	39	12%	12%	47	46	14%	14%

Table 7.2.: Split training testing and validation data

ing parameters are listed in Table 7.3. The parameters chosen are the same for all networks, except for the gradient and quadratic gradient decay factors β_1 and β_2, which were adjusted for the 7x1dim networks in order to achieve stable training results. Further information about used training parameters can be found in [25]. During training, the gradient for updating the network parameters is calculated from one mini-batch, which holds several sequences (in this case 11). Within one mini-Batch the sequences must have the same

	Network Structure	
	14x1dim, 21x1dim, conv-networks	7x1dim
training algorithm	Adam	Adam
max. epochs	2000	2000
mini-batch size	11	11
learn rate schedule	piecewise	piecewise
initial learn rate α	0.02	0.02
learn rate drop factor	0.9	0.9
learn rate drop period	100	100
validation frequency	100	100
sequence length	shrotest	shrotest
gradient decay factor β_1	0.9	0.99
square gradient decay factor β_2	0.999	0.9
Epsilon ε	10^{-8}	10^{-8}

Table 7.3.: Used training parameters

length. The chosen method to achieve same sequence length is to truncate all sequences within a mini-Batch to the shortest sequence within it. To minimize the truncation, it is desirable to sort the sequences by their length before handing them over to the training algorithm. Figure 7.14 shows the effect of truncation within one mini-batch and the advantage of sorting the data can be seen. The result shows that sorting the data significantly reduces the loss of data due to truncation.

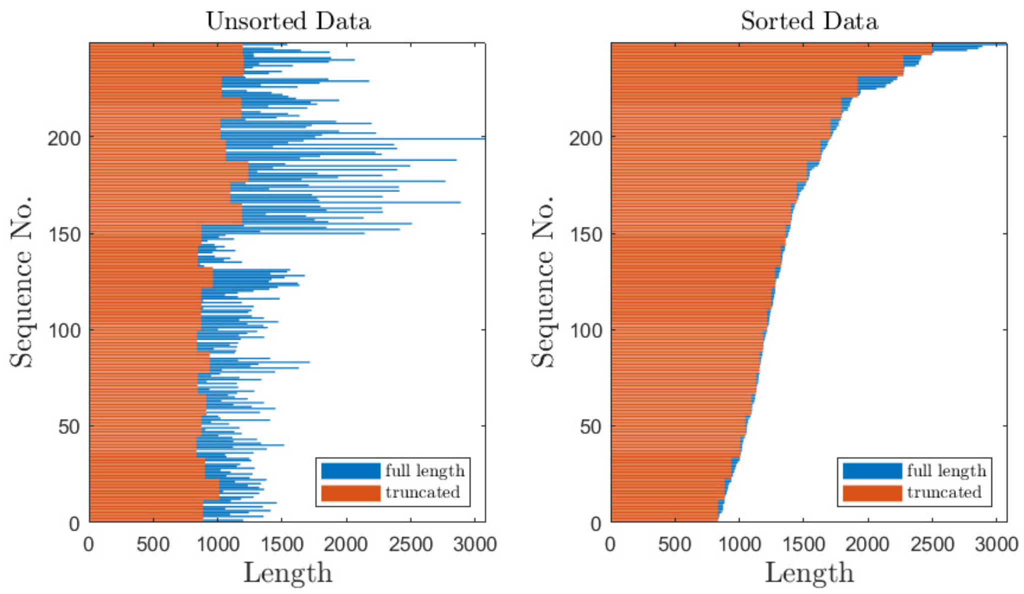


Figure 7.14.: Effect of truncation to shortest sequence within mini-batch

The sorted training data is then passed to the training algorithm, where a total of 14 networks defined with Table 7.1 were trained.

8. Model Comparison

In this chapter, the trained models are evaluated in terms of their generalization capabilities as well as prediction accuracy and a comparison between the model structures is performed using appropriate metrics. In the following sections, first the selected performance metrics are described, followed by a comparison of the models trained on different datasets. Then follows an investigation of the quality of the manually extracted features. Subsequently, the capabilities of models with convolutional layers for feature extraction are explored. Finally, a summary of the results is given, and the best performing model is selected by presenting an overall comparison.

8.1. Performance Metrics

A performance metric can be defined as a mathematical construct that measures how close the actual results are to the predictions. According to Botchkarev [7], the most common performance metrics in machine learning regression are Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). For both metrics, the difference err between the predicted values y and the measured values $\hat{y}$ is calculated for each sample in the test dataset with N samples (Equation 8.1). The MAE is then calculated by taking the mean of the absolute values of the errors (Equation 8.2). The RMSE is given by Equation 8.3 and, as the name implies, is the square root of the MSE defined by Equation 7.11.

$$err = y - \hat{y} \tag{8.1}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |err_i| \tag{8.2}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} err_i^2} \tag{8.3}$$

In this thesis, both MAE and RMSE are computed for model evaluation, with focus on MAE since this measure is more commonly used in machine learning regression [7].

8.2. Investigation of Generalization Capabilities

This section investigates how the used training dataset affects the model performance. To this end, the models 7x1dim, 14x1dim and 21x1dim trained on different datasets are compared to each other. The heatmaps depicted in Figure 8.1 show the MAE of the various networks for predictions made on different test datasets. The x-axis indicates the test dataset on which the predictions were made and the y-axis denotes the train dataset that was applied for the network architecture indicated above each plot.

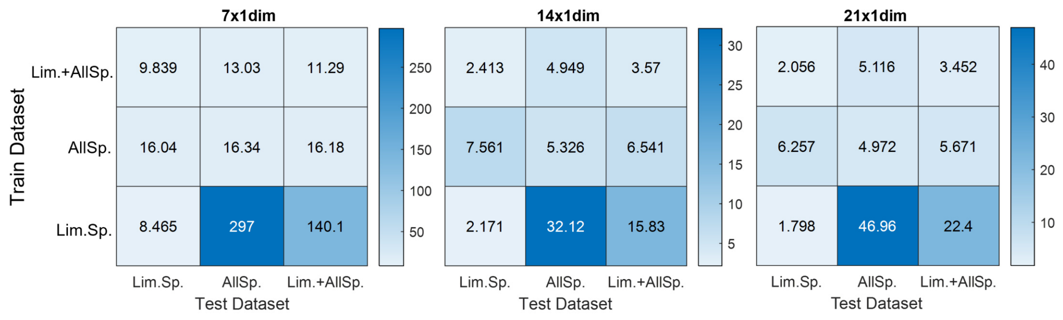


Figure 8.1.: MAE in Watt for 7x1dim 14x1dim 21x1dim models trained on different datasets

The models trained on the limited space data (bottom row) are only able to make predictions with little deviation when the predictions are also made in the limited space. When attempting to make predictions for the data of the entire space, these models perform poorly. The networks in the second row were trained on the all space dataset, and these models are able to make predictions with small errors on the limited space data, as well as on the All Space dataset on which they were trained. The models trained on both the limited and all space datasets show the best performance.
For better interpretation of these results we show some predictions of the models with input feature dimension 21x1 in Figure 8.2. The figure shows predicted and measured values in time domain and as a scatter plot, where the correlation coefficient, MAE, and RMSE are also given. The upper three plots show the predictions made with the same limited space trajectory as input whereas the lower plots show the predictions made on a selected all space trajectory. The selected trajectories are those where the model trained on the Lim.+AllSp. dataset showed the largest MAE. It indicates that the predictions for the limited space data generally agree well with the measured values. However, for all space data, the performance can still be improved.

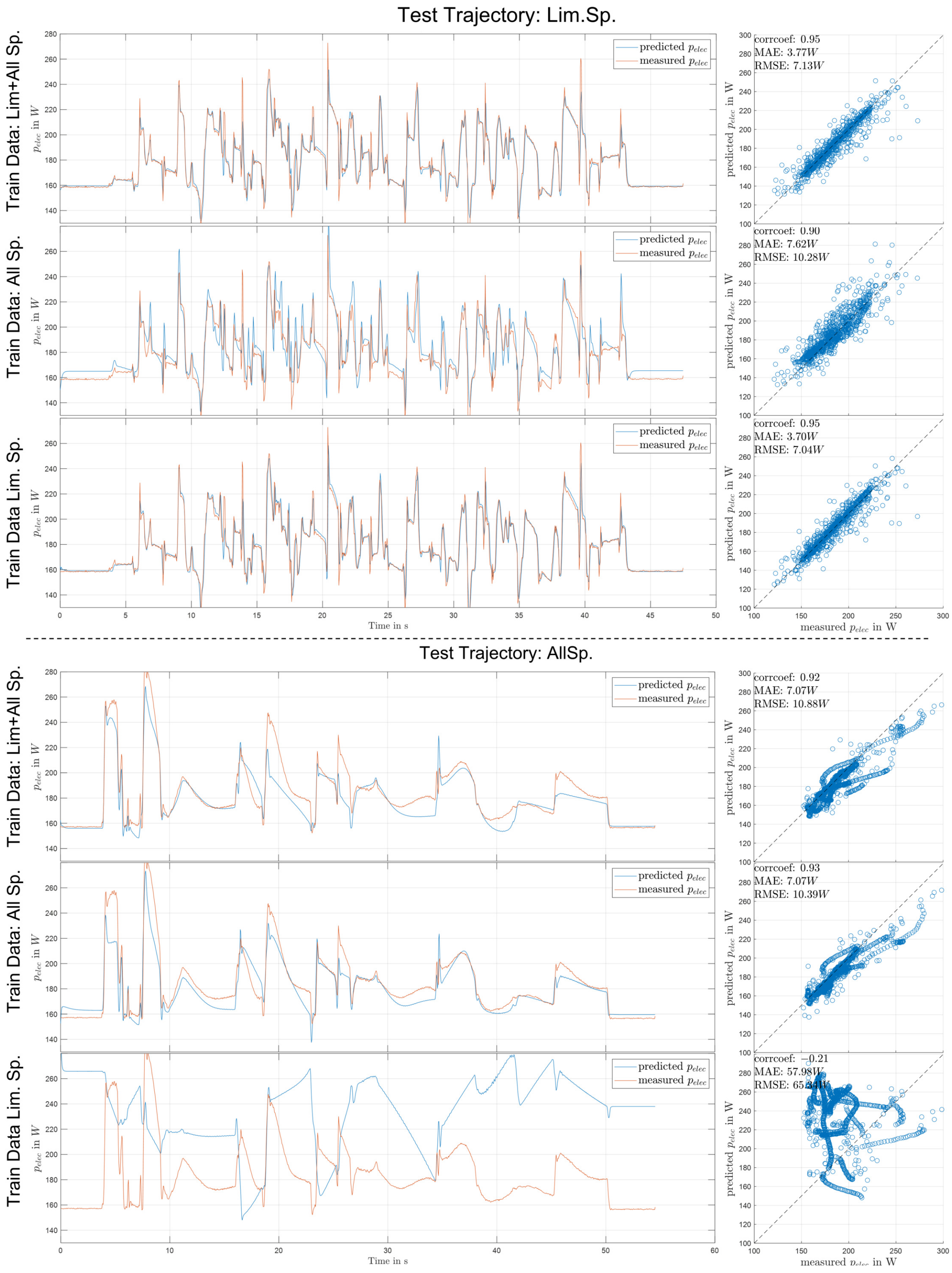


Figure 8.2.: Sample predictions of 21x1dim networks trained on different datasets

8.3. Quality of Manually Engineered Features

In this section, we evaluate the quality of the manually engineered and extracted features by comparing the models that were trained with different features as inputs. Since the models trained with all data performed the best, we selected them for further investigation and for presenting our results. Figure 8.3 shows the MAE of models trained on all data and with different features as input. It appears that the selected features are of good design as the MAE decreases with the inclusion of additional features. The performance of 14x1 and 21x1 is similar, but the 21x1 network is slightly better with regard to all data.

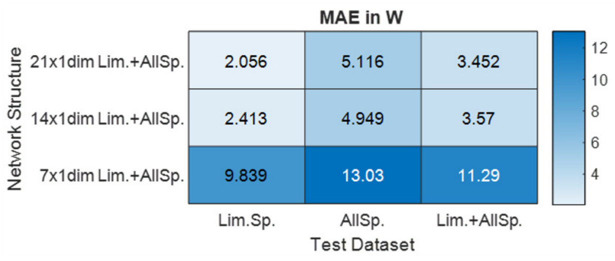


Figure 8.3.: MAE for 7x1dim 14x1dim 21x1dim networks trained combined Lim.- and All Space dataset

Figure 8.4 visualizes the absolute errors for Lim.+AllSp.-dataset as violin plots represented with blue circles. A violin plot adds additional information to the structure of boxplots by graphically representing the distribution characteristics of data batches [16]. In the figure this distribution characteristics are visualized by the symmetric scattering of the data points along the x-dimension. Also indicated are median (MED), MAE, and RMSE. The black box indicates the interquartile range. On the right side the complete data is shown, on the left side an enlarged view.

The plot shows that the network fed with only the 7 axis angles has the poorest performance. Both, 14x1 and 21x1 seem to perform similarly well, with 21x1 being slightly better. In addition to the comparison of the metrics and the distribution of the absolute residuals, the actual predicted sequences were also inspected. This revealed instability issues for the 14x1dim network, which is shown in Figure 8.5.

This undesirable oscillatory behavior was also observed to a lower extent in the predictions of the 7x1dim network, but not with the 21x1dim model.

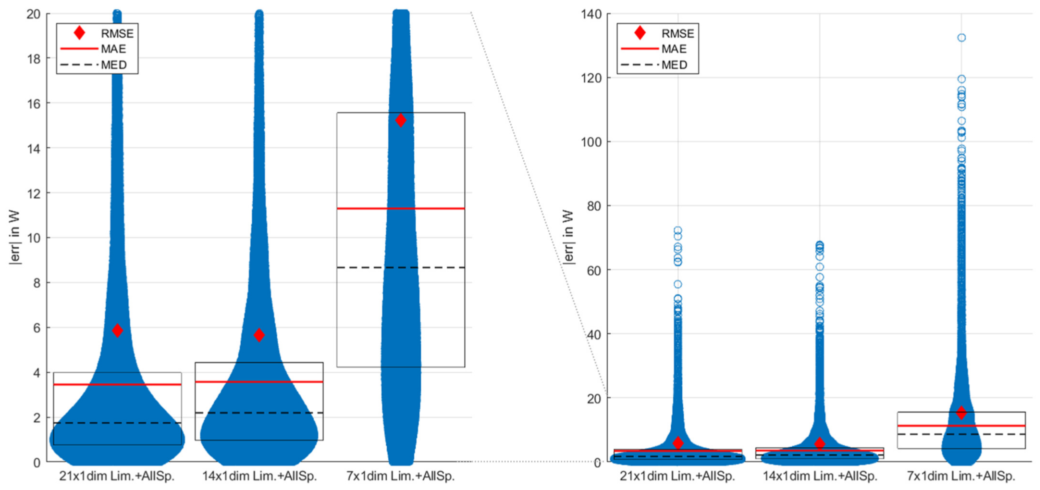


Figure 8.4.: Visualization of $|err|$ for models 7x1dim 14x1dim 21x1dim nets trained on combined Lim.- and All Space dataset

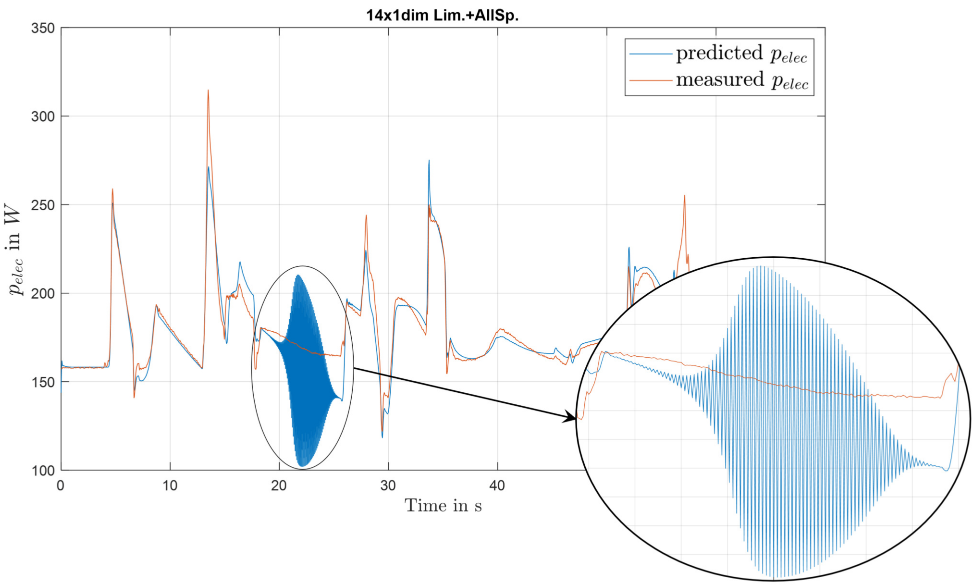


Figure 8.5.: Instability observation of 14x1dim network

8.4. Capabilities of Convolutional Layer for Feature Extraction

In this section, the models with convolutional layers as feature extractors are examined. First, models with two and three filters with receptive field of 1x3 are compared with the models using manually extracted features. Figure 8.6 shows the MAE on different datasets. The MAE of the convolutional models is higher than the models with manually extracted features, but values are in a comparable range. Furthermore, the network with three filter kernels performs better than the one with just two. The poorer performance of the 2conv1x3

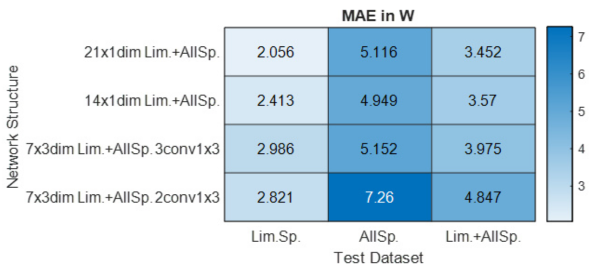


Figure 8.6.: MAE comparison of 14x1dim 21x1dim with 2conv1x3 3conv1x3

network compared to the 3conv1x3 network is also evident in the stretched distribution of absolute error values for the 2conv1x3 network, as shown in Figure 8.7.

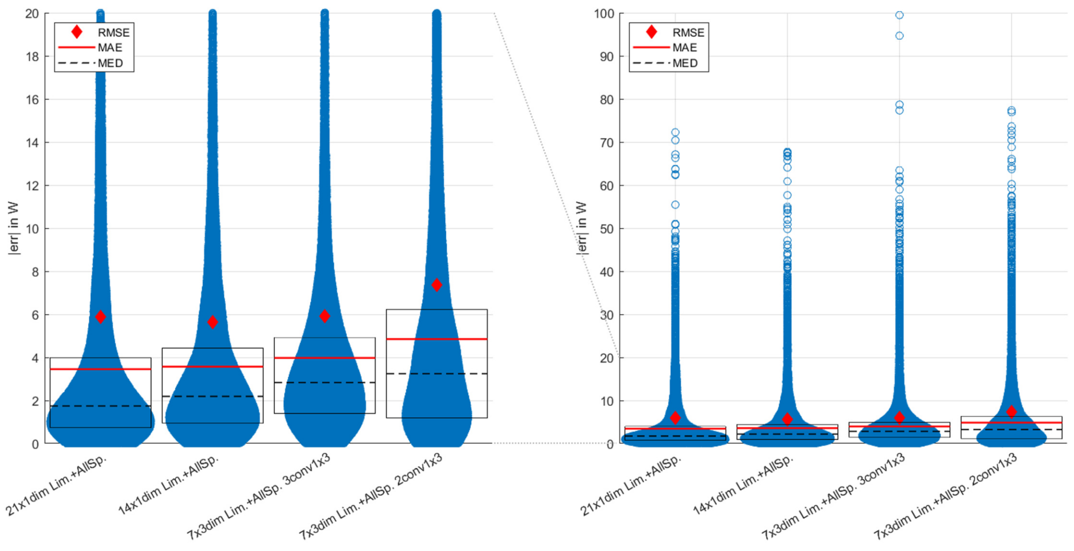


Figure 8.7.: Visualization of $|err|$ for models 14x1dim 21x1dim and 2conv1x3 3conv1x3

The predicted sequences for 2conv1x3 and 3conv1x3 were also checked for possible instability problems. However, no unusual deviations similar to those in Figure 8.5 were found.

Next, we will investigate what filter kernel weights were learned in the convolutional layers and explore the effect of convolution of these filters on a selected input signal For this purpose, a 1-D sequence is selected, representing the normalized angle of one axis as an exemplary input signal. For comparison purposes, the first and second derivatives are also formed, which exemplarily represent the manually extracted features. Figure 8.8 shows the normalized sequence along with the manually extracted first and second derivatives.

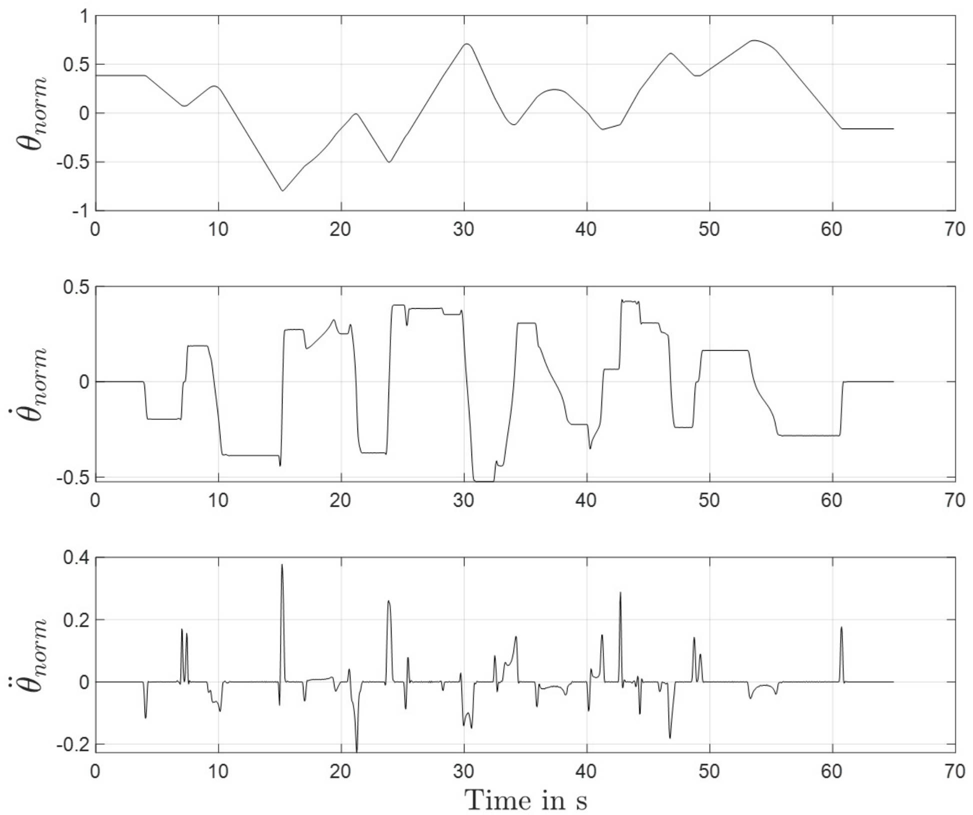


Figure 8.8.: Sample 1-D sequence input and manually extracted features

Figure 8.9 shows the two filter kernels of the 2conv1x3 network, as well as the result of convolving the normalized input signal (θ_{norm}) with these filters. It turns out that by applying filter $w1$, the original shape of the selected input signal is well preserved, but negated and scaled in value. Applying filter $w2$ to the input signal reveals that the resulting sequence compares very well with the manually extracted first derivative. Also notable is, that the kernel $w2$ has considerable similarity to the DoG-kernel with receptive field of $L = 3$, as shown in Figure 7.10.

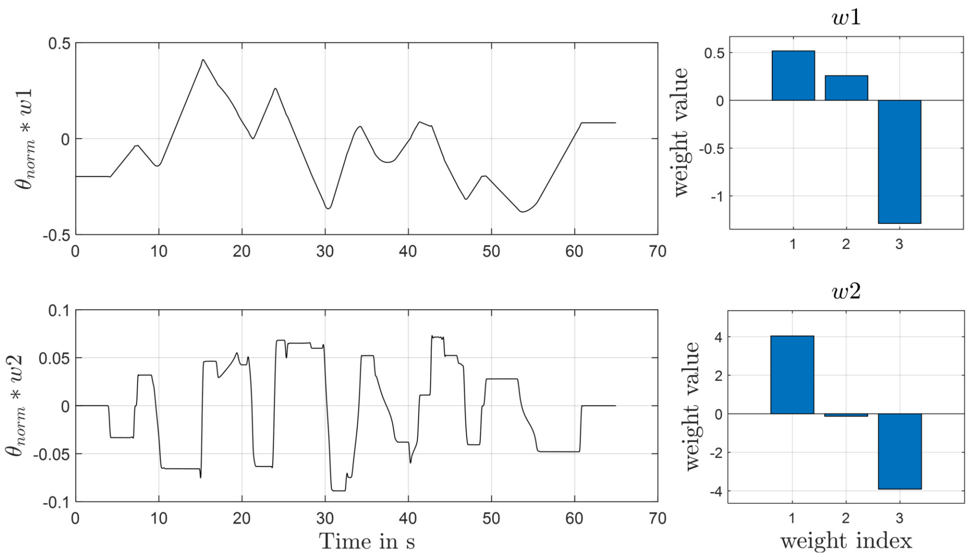


Figure 8.9.: 1-D features extracted from 2conv1x3 CNN

Let us also inspect the result when convolving the example signal with the filter kernels of the 3conv1x3 network. The kernels along with their convolution results are shown in Figure 8.10. When looking at the weights, similarities with the DoG kernel are noticeable as well. However, when these filters are applied to the selected input signal, no clear match can be found with the manually extracted first or second derivative. The result of $w1$ seems to be, to some extent, similar to the first derivative, while the results of $w2$ and $w3$ are similar to each other, but cannot be unambiguously assigned to any of the manually extracted features.

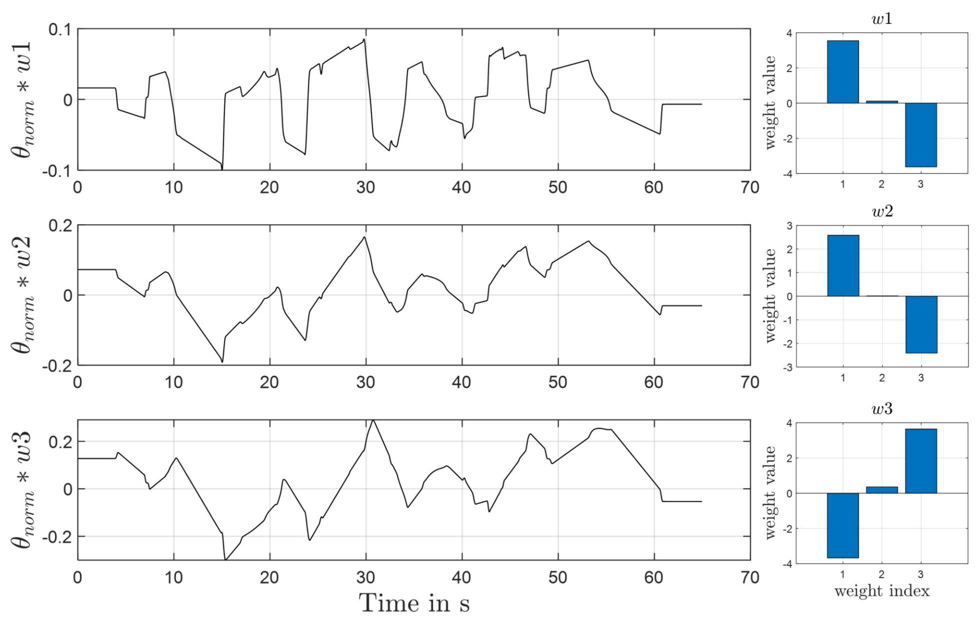


Figure 8.10.: 1-D features extracted with 3conv1x3 kernels

In addition to the 2conv1x3 and 3conv1x3 networks, also experiments with wider and more filter kernels were carried out. For this purpose the networks 3conv1x5, 3conv1x11 and 5conv1x21 were trained. The MAE and error plots for the three additional networks are shown in Figure 8.11 and Figure 8.12. The 21x1dim network is included for comparison.

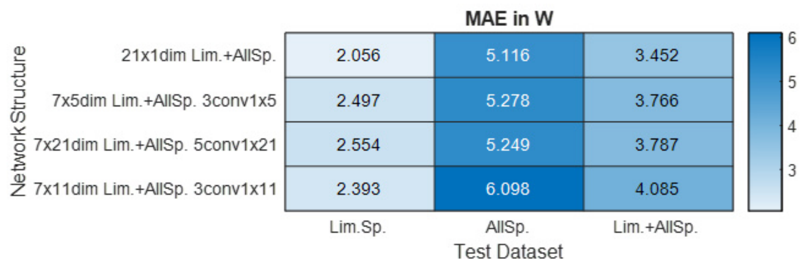


Figure 8.11.: MAE comparison of 21x1dim with 3conv1x5 5conv1x21 3conv1x11

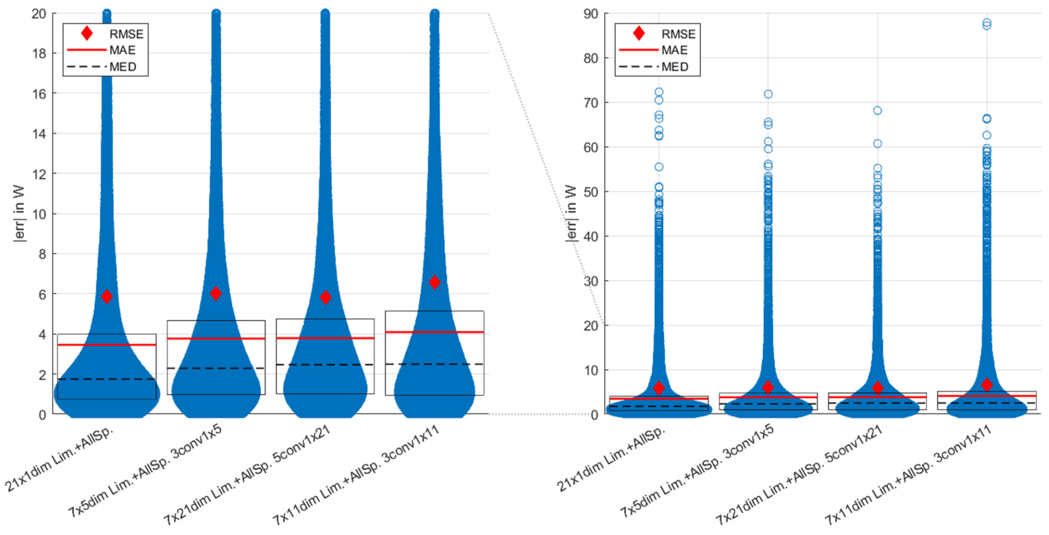


Figure 8.12.: Visualization of $|err|$ for models 21x1dim 3conv1x5 5conv1x21 3conv1x11

The performances of the networks 3conv1x5, 5conv1x21, 3conv1x11 are quite similar with no model standing out. It is also evident that the MAE of the convolutional models is higher than that of the 21x1dim network, but the values are of little difference.
The learned kernels of network 3conv1x5 along with the convolution results are depicted in Figure 8.13. The effect of applying $w1$ is similar to that of kernel $w1$ from 2conv1x3 (Figure 8.9). Although the appearance of those filter weights is very different, both kernels have the effect of scaling and negating the example input signal. The convolution results of kernels $w2$ and $w3$ of network 3conv1x5 appear similar to each other, with the resulting sequences having characteristics of the first derivative. The results from 3conv1x11 kernels are shown in Figure 8.14. Again, the effect of negating the input signal can be observed with kernel $w2$. Interestingly, the kernels $w1$ and $w3$ appear similar to the LoG and DoG in Figure 7.9, where the convolution result of $w1$ has the properties of a superposition of the first and second derivatives, and the result of $w3$ has the properties of the second derivative.

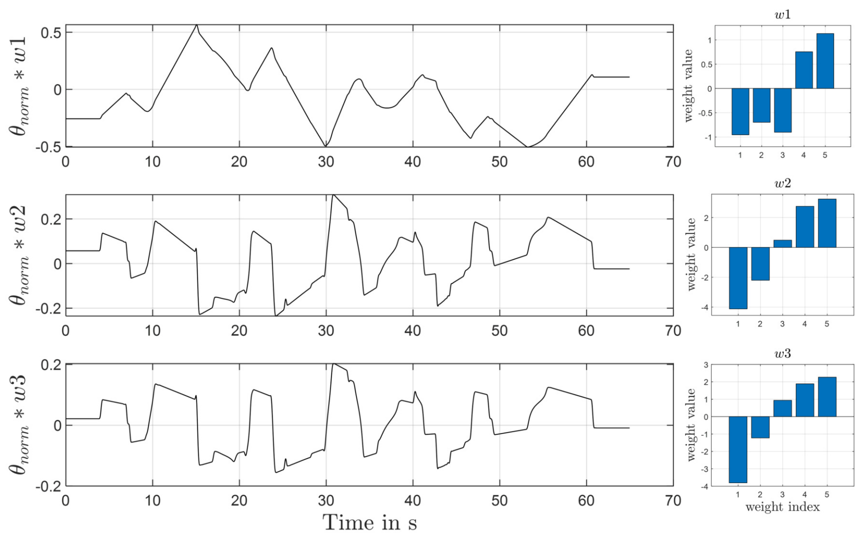


Figure 8.13.: 1-D features extracted from 3conv1x5 kernels

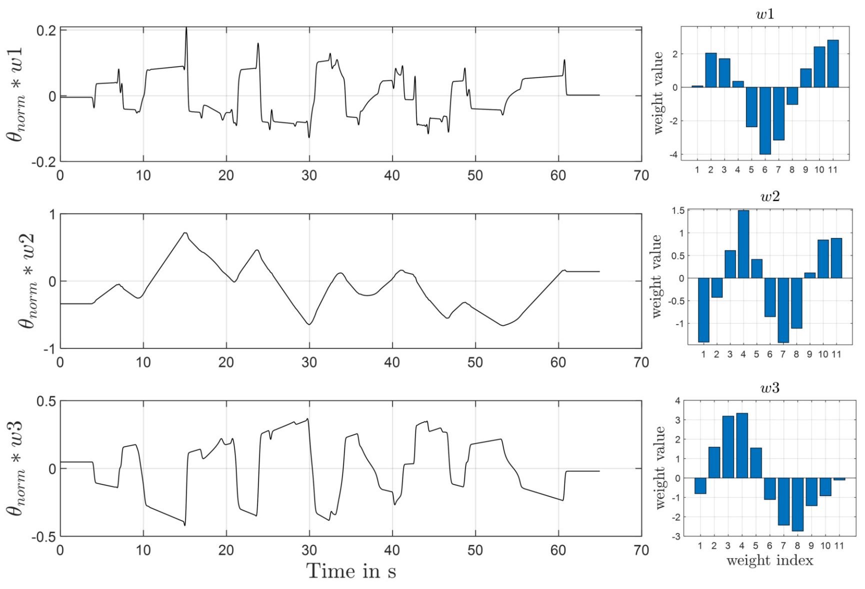


Figure 8.14.: 1-D features extracted from 3conv1x11 kernels

Finally, the kernels of the 5conv1x21 network are shown in Figure 8.15 along with the convolution results. The results of filtering with kernels $w3$ and $w5$ show similarities with the second derivative, while the result of kernel $w4$ again indicates a kind of superposition of the first and second derivative. With the learned weights of the kernel $w1$, the extracted sequences again seem to resemble the first derivative to some extent, whereas the result form $w2$ is not unambiguously interpretable.

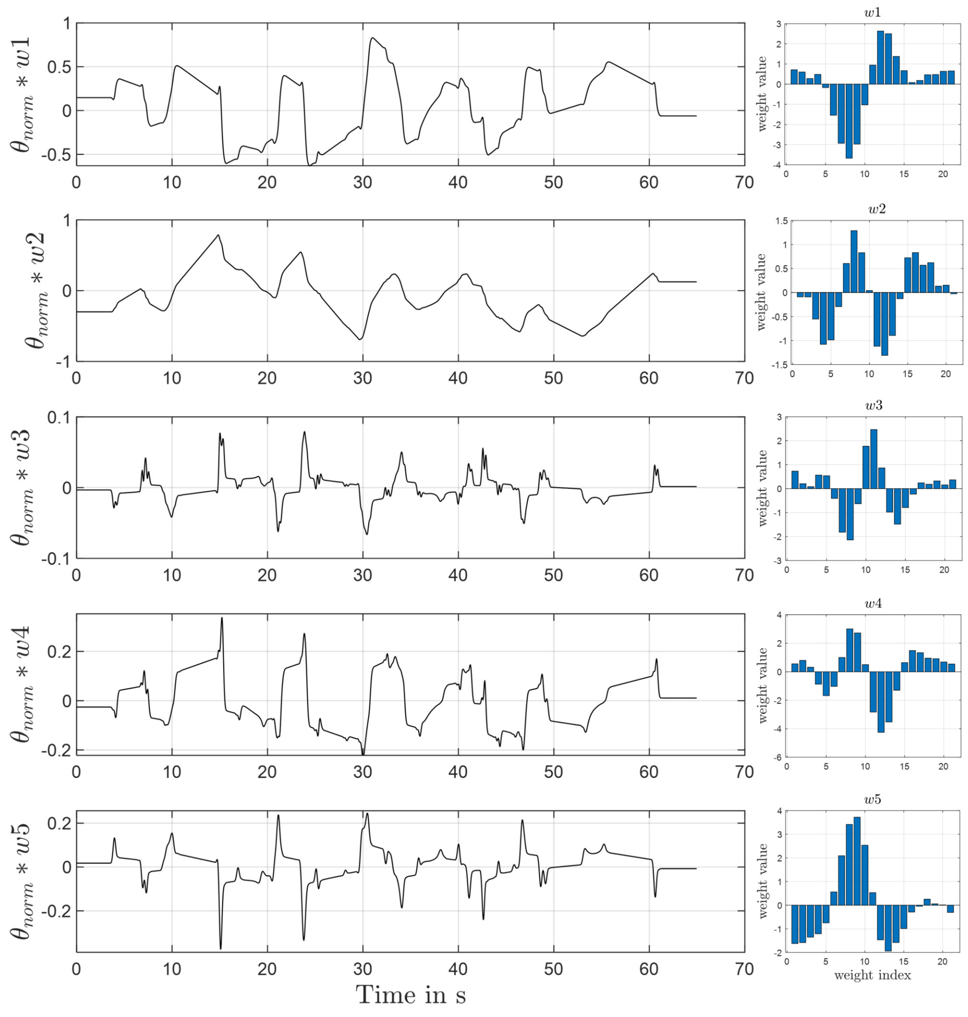
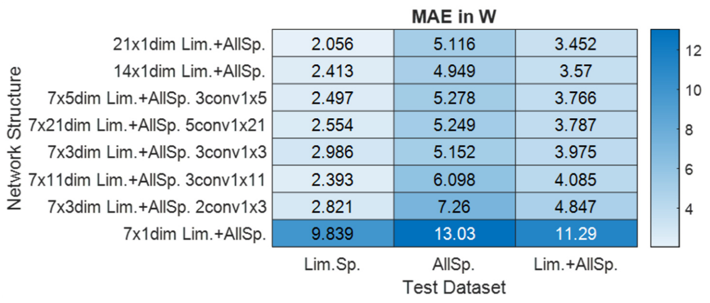


Figure 8.15.: 1-D features extracted from 5conv1x21 kernels

8.5. Overall Comparison and Model Selection

Based on the results described in the previous sections, let us now present an overall comparison of the evaluated models. Figure 8.16 shows the MAE in tabular form and Figure 8.17 shows the violin plots of the absolute errors. The performance of the networks with manually extracted features and those with learned convolutional feature extractors are within a narrow range. However, the best overall performance in terms of MAE along with a narrow interquartile range of the absolute errors is shown by the 21x1dim network.



MAE in W

Network Structure	Lim.Sp.	AllSp.	Lim.+AllSp.
21x1dim Lim.+AllSp.	2.056	5.116	3.452
14x1dim Lim.+AllSp.	2.413	4.949	3.57
7x5dim Lim.+AllSp. 3conv1x5	2.497	5.278	3.766
7x21dim Lim.+AllSp. 5conv1x21	2.554	5.249	3.787
7x3dim Lim.+AllSp. 3conv1x3	2.986	5.152	3.975
7x11dim Lim.+AllSp. 3conv1x11	2.393	6.098	4.085
7x3dim Lim.+AllSp. 2conv1x3	2.821	7.26	4.847
7x1dim Lim.+AllSp.	9.839	13.03	11.29

Test Dataset

Figure 8.16.: MAE comparison all models

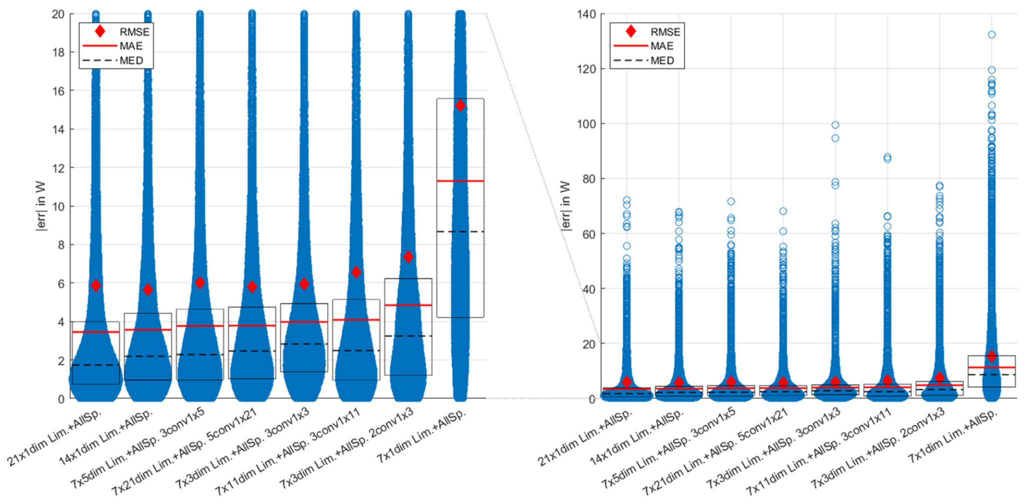


Figure 8.17.: Visualization of $|err|$ for all models

To further illustrate the performance of the networks, Figure 8.18 shows an example of the predictions from three selected models. As input trajectory an All Space trajectory was chosen for which the model 21x1dim has the lowest

MAE.
Since the 21x1dim network appears to have the best overall performance, this model is chosen to conduct additional experiments in the course of this thesis.

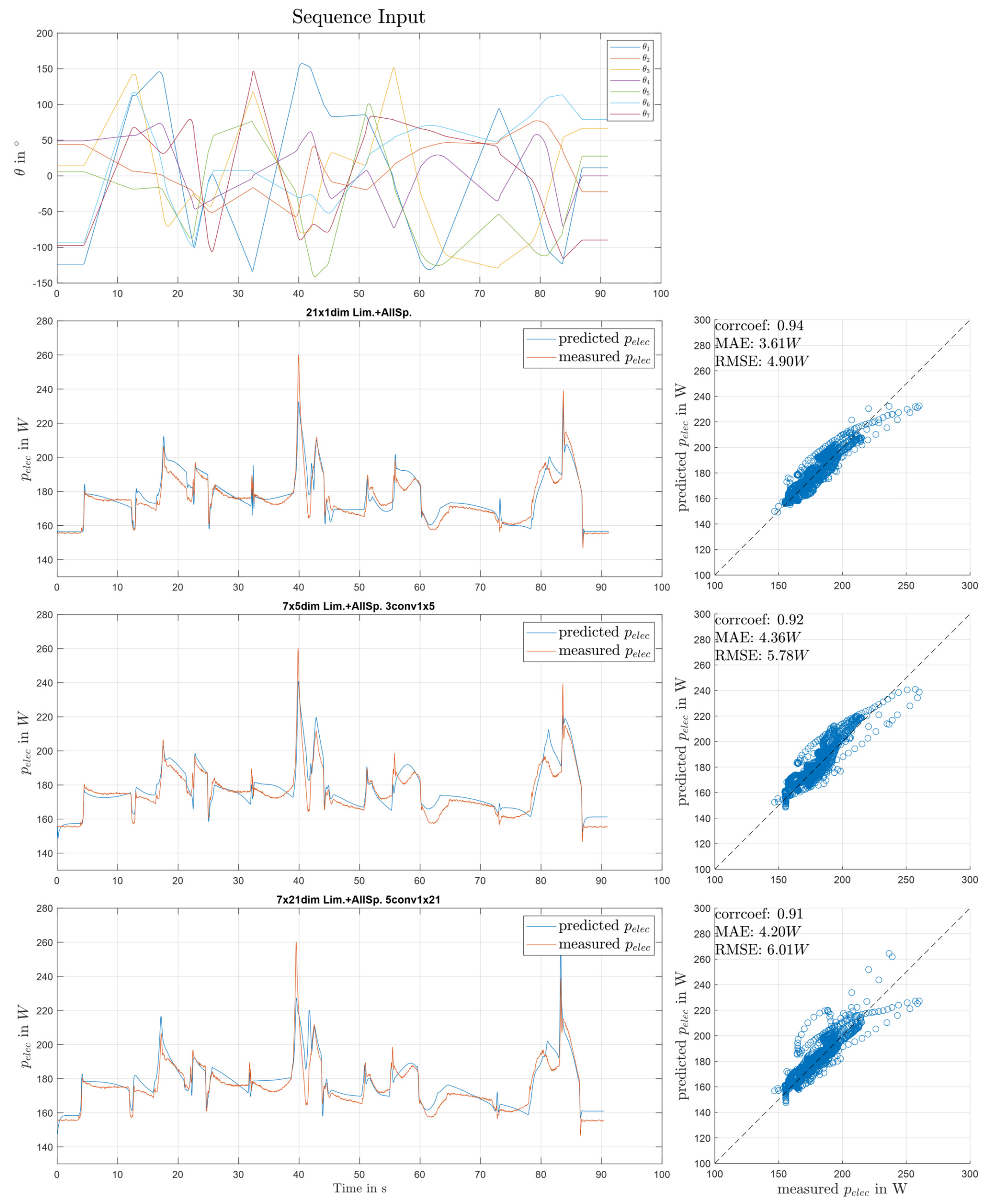


Figure 8.18.: Predictions of 21x1 3conv1x5 5conv1x21 nets on selected All Space trajectory

9. Test on Real Use-Case

This chapter presents an evaluation of the previously selected model on the basis of a real use case. Since predictive models are often used in simulation environments, the evaluation will also include the typical steps required to utilize the created model within such an environment. In doing so, several aspects will be addressed, namely:

- convert the derived model to a platform independent exchange format
- model the real use-case in an simulation environment
- utilize the energy model within the simulation environment
- evaluating simulation results

The use case is the assembly of a Fidget Spinner, which is part of a overall manufacturing process in the laboratory of the Digital Factory and consists of the steps shown in Figure 9.1.

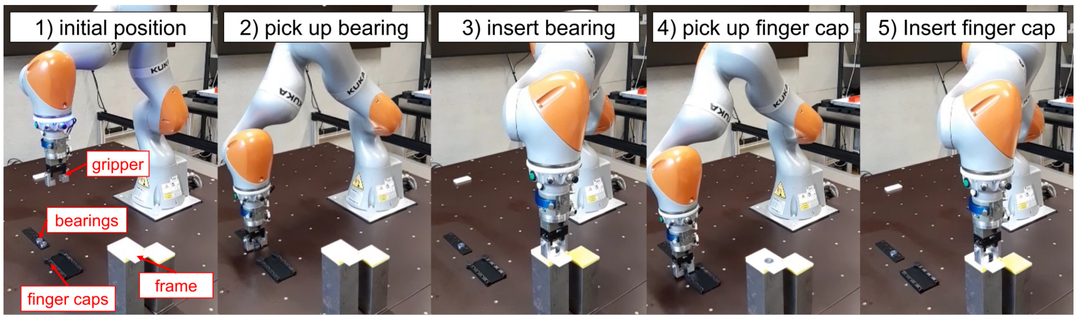


Figure 9.1.: Fidget spinner assembly

The simulation software used for testing the energy model is called *twin* and is developed and maintained by the company *digifai*[1]. Within the twin environment it is possible to simulate and control the kinematic motions of a robotic system. twin also offers the possibility to integrate 3rd party models by means of so-called Functional Mockup Units (FMUs).

[1]www.digifai.com

The next section briefly explains the process of converting the model into an FMU. This is followed by the description of implementing the assembly process, including the energy model, within the simulation environment. Subsequently, the simulation results are compared with the real system behavior.

9.1. FMU Generation

An FMU is an executable file that implements the Functional Mock-up Interface (FMI). The FMU is essentially a ZIP-file whose main components are an XML-formatted model description and C-code source files including the required runtime libraries used in the model. A FMU can be formatted for model exchange or co-simulation [27]. In this work, we use co-simulation formatted FMUs, since the simulation environment used is only capable to process this type. The procedure worked out for converting a MATLAB neural network to an FMU consists of the following steps:

1. embed the network within a matlab function
2. generate executable C-code out of the function by utilizing matlab coder
3. embed the C-code within a s-function within simulink
4. export the simulink model as FMU for co-simulation

A more detailed explanation of this procedure can be found in Appendix C.1.

9.2. Integration and Test in twin Software

In order to test the energy model, the assembly process was simulated using the twin software. A 3D model was created consisting of the assembly table and the industrial robots. The control logic to implement the robot motion was implemented based on the real use case. The created energy model was also integrated as a FMU and the simulated axis angles of one robot were used as inputs.
Figure 9.2 shows the 3D model of the assembly structure with the underlying logical layer where the movements of the rigid bodies are controlled and energy prediction by the FMU is performed.

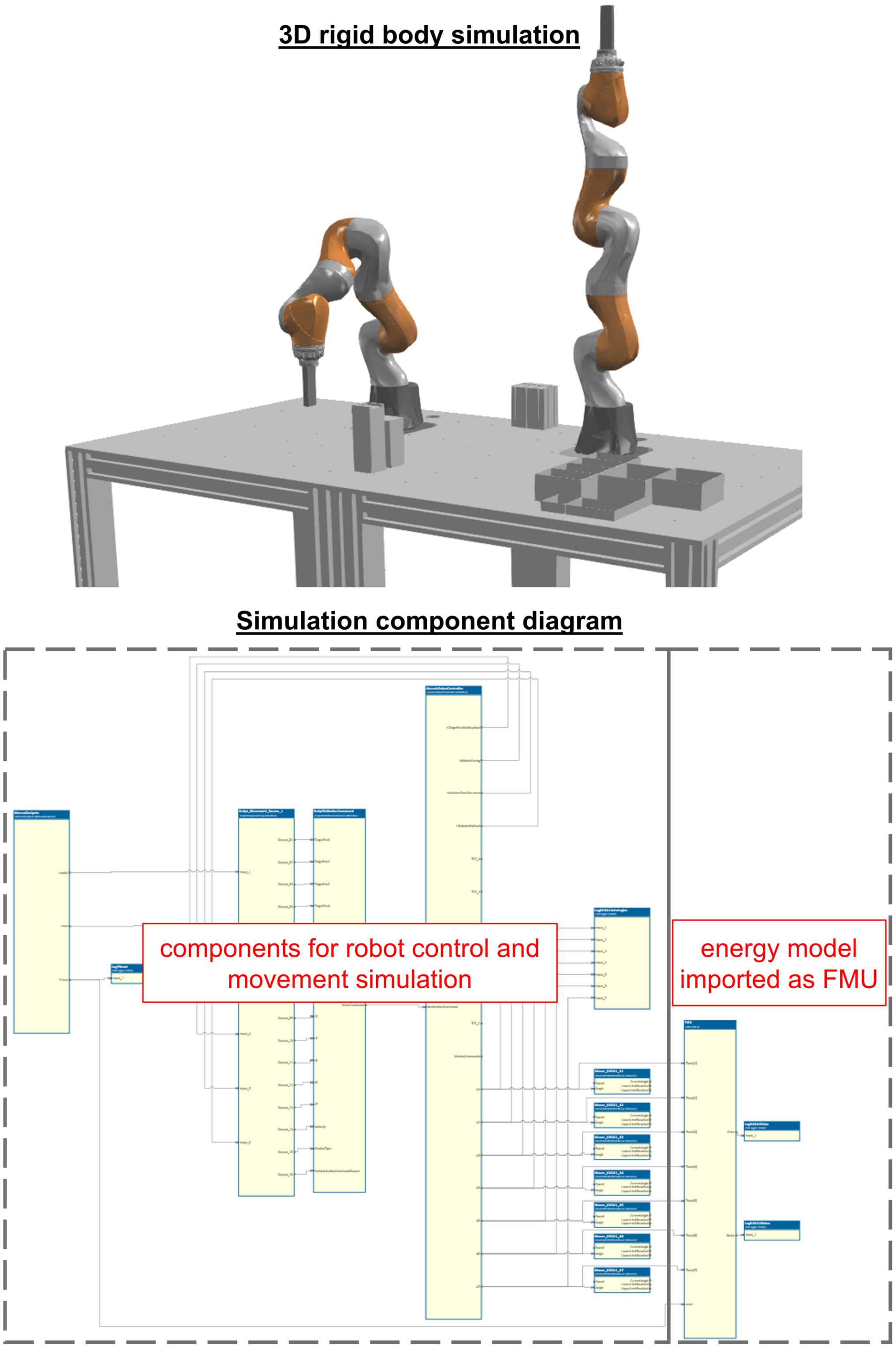

3D rigid body simulation
Simulation component diagram
components for robot control and movement simulation
energy model imported as FMU

Figure 9.2.: Simulation setup twin software: top the 3D rigid body representation; bottom movement control and energy simulation

9.3. Simulation Results

This section presents the simulation results based on the generated FMU and the simulated robot motion. Figure 9.3 shows the motion information of the simulation as well as the measurements from the real system when the assembly is performed. On the left side, the measured and simulated axis angles are shown. The right side represents the Cartesian position of the end-effector, which was determined by forward kinematics using the axis angles. The comparison shows, that the simulated positions of axes 3 and 5 differ significantly from the measured values. However, it is apparent that although the axis angles of the virtual robot controller differ from the real system, the calculated positions of the end-effector are close to the real counterpart.

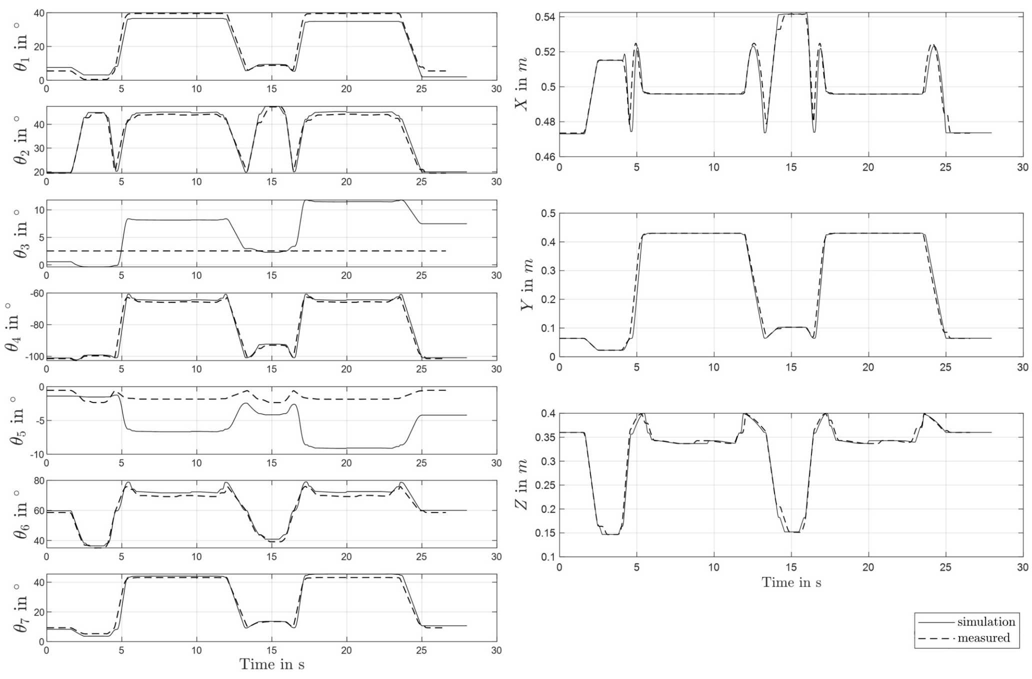


Figure 9.3.: Comparison simulated and measured movement: left the axis angles; right end-effector position

Finally, with the confirmation that the simulated robot motion performs similar to the real system, we can present the results of the simulated power consumption. For comparison purposes, the power consumption was also predicted based on the measured axis angles of the real system, referred to as *predicted power* in the following, whereas the results of the simulation using the imitated robot motion are referred to as *simulated power*. Figure 9.4 shows the respective comparison of predicted power and simulated power with the real power consumption measured. The results show a high initial peak in the measurements which can be attributed to the inrush currents of the axes motors when the breaks of the axes are released. This peak in current consumption can unfortunately not be replicated by the derived models because no information about the state of the brakes was provided during training. Therefore, for additional reference, the performance metrics were also calculated without the initial peak. This leads to the values $MAE_{pred} = 4.15$, $RMSE_{pred} = 7.98$, $MAE_{sim} = 6.60$, $RMSE_{sim} = 13.14$. The corresponding plots to these results can be found in the Appendix C.2.

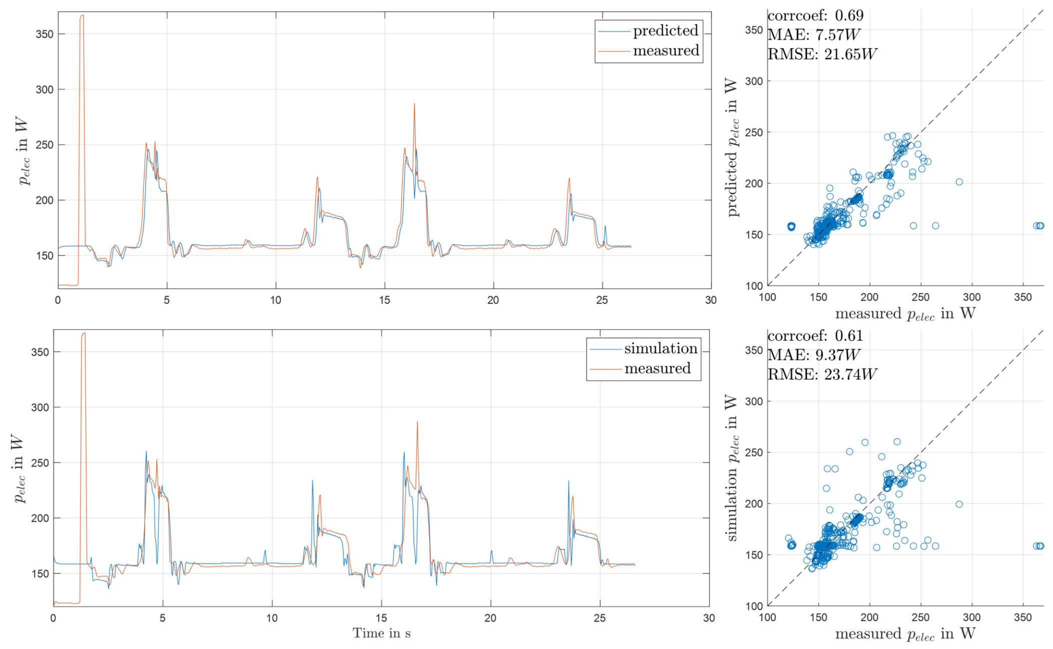


Figure 9.4.: Comparison of predicted and simulated with measured power: top: power predicted with measured axis angles, bottom: power from simulation

Apart from the deviation at the beginning, the created energy model is able to represent the real energy consumption well. Furthermore, the simulated power and predicted power are very similar except for larger deviation peaks at times 4.7, 11.8, 16.5 and 23.6. These peaks can be attributed to the differences between simulated and measured movement shown in Figure 9.3.

10. Discussion of Results

In this thesis, the complete procedure for the creation of a data-driven energy model was presented and all necessary steps from data acquisition to the implementation of the created model within a simulation environment were demonstrated. As a result we were able to create a model with, in the author's opinion, satisfactory accuracy. However, the limitations of the created model should not be ignored. Thus, no external forces acting on the manipulator or a change in mass, e.g. due to the gripping of an object, were taken into account for model building. This limits the field of applications to robotic tasks having limited interaction with the environment and no significant mass changes on the manipulator. Furthermore, it should be noted that for energy simulation a sequence of axis data has to be provided. In practice, this requires path planning and calculation of the inverse kinematics by a suitable tool. In terms of modeling, a very interesting aspect is the demonstrated ability of the convolutional layers to automatically extract meaningful features from the raw data, resulting in models with a performance almost on the same level as those created with manually engineered features whose creation requires a certain level of expertise. We also investigated the learned filter kernels and tried to evaluate their properties by comparing them with known filter kernels and by applying them to a selected signal. Interestingly, similarities were detected between the manually and automatically extracted features, and also the shape of the filter masks could partially be related to the known kernels like DoG and LoG. It must be mentioned that these observations are based on a first purely qualitative judgment by the author and that the analysis on the basis of only one selected signal cannot be considered as sufficient to comprehensively assess the properties of the observed filters. This would require further investigations, e.g. by means of digital signal processing methods such as frequency analysis, which unfortunately could not be carried out within the scope of this thesis. Besides further evaluations of the filter kernels, it would be desirable to test other structures and technologies of different neural networks in future studies to achieve a reduction of the prediction error. However, a possible improvement in model performance could also be achieved by appropriate time synchronization of the measurement systems or possibly by applying different trajectories for the system excitation.
Despite the limitations of the created model and the possibilities for improve-

ment in methodology and evaluation, this work shows a comprehensive elaboration of a complete procedure for the creation of a data-driven energy model by applying interdisciplinary methods from mechatronics, robotics and machine learning.

11. Conclusion

This thesis presented a data-driven method for creating energy models predicting the energy consumption of dynamic systems. To accomplish this, a measurement concept was developed and implemented to aggregate data from the real system. Furthermore, different excitation trajectories were designed and executed. Analyzing the recorded data revealed an error, which was noticeable by a shift in the temporal information of the time series of the electrical and mechanical data. A solution based on cross-correlation calculations was devised for compensating this error in order ot make the data accessible for the modeling process. Different ANN architectures were set up for model building. In addition, both manually engineered features as well as automatic feature extraction were investigated. An interesting aspect of the comparison of the models was, that th automatic feature extraction using convolutional layers performed very closely to the manually extracted features. Furthermore, the engineered features proved to be of good quality, as the prediction accuracy could be improved using them as input. For the networks with convolutional layers the properties of the learned filter masks were examined more closely. Interestingly, the learned filter masks extracted similar features as the manually created ones. Although the performance of the automatic feature extraction networks is not quite as good as that of the manually extracted models, this architecture is very attractive because models can be created with sufficient accuracy even without deeper knowledge of the system to be modeled.
Finally, a possibility of using such a model in the case of a real application was presented. For this purpose, a selected model was converted into a suitable exchange format (FMU) and integrated into a simulation environment. The results showed that the created model is able to make satisfying predictions.

Bibliography

[1] V. Bargsten, P. Zometa, and R. Findeisen. “Modeling, parameter identification and model-based control of a lightweight robotic manipulator”. In: *2013 IEEE International Conference on Control Applications (CCA)*. ISSN: 1085-1992. Aug. 2013, pp. 134–139. DOI: 10.1109/CCA.2013.6662756.

[2] Beckhoff Automation GmbH & Co. KG. *Documentation EtherCAT Bus Coupler, EK110x-00xx, EK15xx*. URL: https://download.beckhoff.com/download/Document/io/ethercat-terminals/ek110x_ek15xxen.pdf (visited on 05/06/2020).

[3] Beckhoff Automation GmbH & Co. KG. *Documentation: Power monitoring oversampling terminal for 690V, EL3783*. URL: https://download.beckhoff.com/download/Document/io/ethercat-terminals/el3783en.pdf (visited on 05/06/2020).

[4] *Beckhoff Information System - English: Supported network controllers*. URL: https://infosys.beckhoff.com/content/1033/tc3_overview/9309844363.html?id=1489698440745036069 (visited on 06/08/2021).

[5] S. A. Billings. *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. Chichester, West Sussex, United Kingdom: John Wiley & Sons, Inc, 2013. ISBN: 978-1-118-53555-4.

[6] Stan Birchfield. *Image Processing and Analysis*. 1st edition. Mason, OH: Cengage Learning, 2016. ISBN: 978-1-285-17952-0.

[7] Alexei Botchkarev. “A New Typology Design of Performance Metrics to Measure Errors in Machine Learning Regression Algorithms”. en. In: *Interdisciplinary Journal of Information, Knowledge, and Management* 14 (Jan. 2019), pp. 045–076. DOI: 10.28945/4184.

[8] Steven L. Brunton and Jose Nathan Kutz. *Data-driven science and engineering: machine learning, dynamical systems, and control*. Cambridge: Cambridge University Press, 2019. ISBN: 978-1-108-42209-3.

[9] François Chollet. *Deep learning with Python*. OCLC: ocn982650571. Shelter Island, New York: Manning Publications Co, 2018. ISBN: 978-1-61729-443-3.

[10] Microsoft Documentation. *File Times - Win32 apps*. en-us. URL: https://docs.microsoft.com/en-us/windows/win32/sysinfo/file-times (visited on 07/23/2021).

[11] *EtherCAT Technology Group | EtherCAT*. URL: https://www.ethercat.org/en/technology.html (visited on 06/07/2021).

[12] Jeff Friesen. *Java XML and JSON: Document Processing for Java SE*. 2nd ed. 2019. Berkeley, CA: Apress : Imprint: Apress, 2019. ISBN: 978-1-4842-4330-5. DOI: 10.1007/978-1-4842-4330-5.

[13] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. eng. Fourth edition, global edition. New York, NY: Pearson, 2018. ISBN: 978-1-292-22304-9.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016. ISBN: 978-0-262-03561-3.

[15] Martin T. Hagan et al. *Neural network design*. eng. 2nd edition. Wrocław: Amazon Fulfillment Poland Sp. z o.o, 2014. ISBN: 978-0-9717321-1-7.

[16] Jerry L. Hintze and Ray D. Nelson. "Violin Plots: A Box Plot-Density Trace Synergism". In: *The American Statistician* 52.2 (May 1998), pp. 181–184. ISSN: 0003-1305. DOI: 10.1080/00031305.1998.10480559.

[17] *HP ProDesk 600 G2 Desktop-Mini-PC - Technische Daten | HP® Kundensupport*. URL: https://support.hp.com/at-de/product/hp-prodesk-600-g2-desktop-mini-pc/8376393/document/c04850252#AbT5 (visited on 06/08/2021).

[18] *ISO/IEC 20922: Information technology - Message Queuing Telemetry Transport (MQTT) v3.1.1*. 2016. URL: https://www.iso.org/standard/69466.html.

[19] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[20] KUKA Deutschland GmbH. *BA KUKA Sunrise Cabinet V10; Controller KUKA Sunrise Cabinet Operating instructions*.

[21] KUKA Deutschland GmbH. *BA LBR iiwa V8; LBR iiwa 7 R800, LBR iiwa 14 R820 Operating instructions*.

[22] Aiming Liu et al. “Energy consumption modeling of industrial robot based on simulated power data and parameter identification”. en. In: *Advances in Mechanical Engineering* 10.5 (May 2018). Publisher: SAGE Publications, p. 1687814018773852. ISSN: 1687-8140. DOI: 10.1177/1687814018773852.

[23] Kevin M. Lynch and Frank C. Park. *Modern robotics: mechanics, planning, and control*. OCLC: ocn983881868. Cambridge, UK: Cambridge University Press, 2017. ISBN: 978-1-107-15630-2.

[24] *MATLAB Documentation: Long short-term memory (LSTM) layer*. URL: https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.lstmlayer.html (visited on 08/24/2021).

[25] *MATLAB Documentation: Options for training deep learning neural network trainingOptions*. URL: https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html (visited on 08/26/2021).

[26] *MATLAB Documentatoin: Cross-correlation xcorr*. URL: https://www.mathworks.com/help/matlab/ref/xcorr.html (visited on 08/26/2021).

[27] Modelica Association. *FMI-Specification-2.0.2; Functional Mock-up Interface for Model Exchange and Co-Simulation*. 2020. URL: https://github.com/modelica/fmi-standard/releases/download/v2.0.2/FMI-Specification-2.0.2.pdf.

[28] *MQTT - The Standard for IoT Messaging*. URL: https://mqtt.org/ (visited on 06/06/2021).

[29] *MQTT Version 5.0. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 15 May 2018. OASIS Committee Specification 02*. URL: http://docs.oasis-open.org/mqtt/mqtt/v5.0/cs02/mqtt-v5.0-cs02.html.%20Latest%20version:%20http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[30] Olalekan Ogunmolu et al. “Nonlinear Systems Identification Using Deep Dynamic Neural Networks”. en. In: *arXiv:1610.01439 [cs]* (Oct. 2016). arXiv: 1610.01439. URL: http://arxiv.org/abs/1610.01439 (visited on 09/13/2020).

[31] Paryanto et al. “Reducing the energy consumption of industrial robots in manufacturing systems”. In: *The International Journal of Advanced Manufacturing Technology* 78.5 (May 2015), pp. 1315–1328. ISSN: 1433-3015. DOI: 10.1007/s00170-014-6737-z. URL: https://doi.org/10.1007/s00170-014-6737-z.

[32] Sudharsan Ravichandiran. *Hands-on deep learning algorithms with Python: master deep learning algorithms with extensive math by implementing them using TensorFlow*. English. OCLC: 1083564019. 2019. ISBN: 978-1-78934-415-8.

[33] Edwin Schicker. *Datenbanken und SQL: eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL*. ger. 5., aktualisierte und erweiterte Auflage. Informatik & Praxis. OCLC: 964674055. Wiesbaden: Springer Vieweg, 2017. ISBN: 978-3-658-16128-6.

[34] Jan Swevers, Walter Verdonck, and Joris De Schutter. "Dynamic Model Identification for Industrial Robots". In: *IEEE Control Systems Magazine* 27.5 (Oct. 2007). Conference Name: IEEE Control Systems Magazine, pp. 58–71. ISSN: 1941-000X. DOI: 10.1109/MCS.2007.904659.

[35] Yury Tiumentsev. *Neural network modeling and identification of dynamical systems*. 1st edition. Cambridge, MA: Elsevier, 2019. ISBN: 978-0-12-815254-6.

[36] Jessica Walther and Matthias Weigold. "A Systematic Review on Predicting and Forecasting the Electrical Energy Consumption in the Manufacturing Industry". en. In: *Energies* 14.4 (Jan. 2021). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 968. DOI: 10.3390/en14040968.

[37] Jun Wu, Jinsong Wang, and Zheng You. "An overview of dynamic parameter identification of robots". en. In: *Robotics and Computer-Integrated Manufacturing* 26.5 (Oct. 2010), pp. 414–419. ISSN: 0736-5845. DOI: 10.1016/j.rcim.2010.03.013. URL: http://www.sciencedirect.com/science/article/pii/S0736584510000232 (visited on 05/27/2020).

[38] Ke Yan et al. "Digital Twin-Based Energy Modeling of Industrial Robots". en. In: *Methods and Applications for Modeling and Simulation of Complex Systems*. Ed. by Liang Li, Kyoko Hasegawa, and Satoshi Tanaka. Communications in Computer and Information Science. Singapore: Springer, 2018, pp. 333–348. ISBN: 9789811328534. DOI: 10.1007/978-981-13-2853-4_26.

[39] Shubin Yin, Wei Ji, and Lihui Wang. "A machine learning based energy efficient trajectory planning approach for industrial robots". en. In: *Procedia CIRP*. 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019 81 (Jan. 2019), pp. 429–434. ISSN: 2212-8271. DOI: 10.1016/j.procir.2019.03.074.

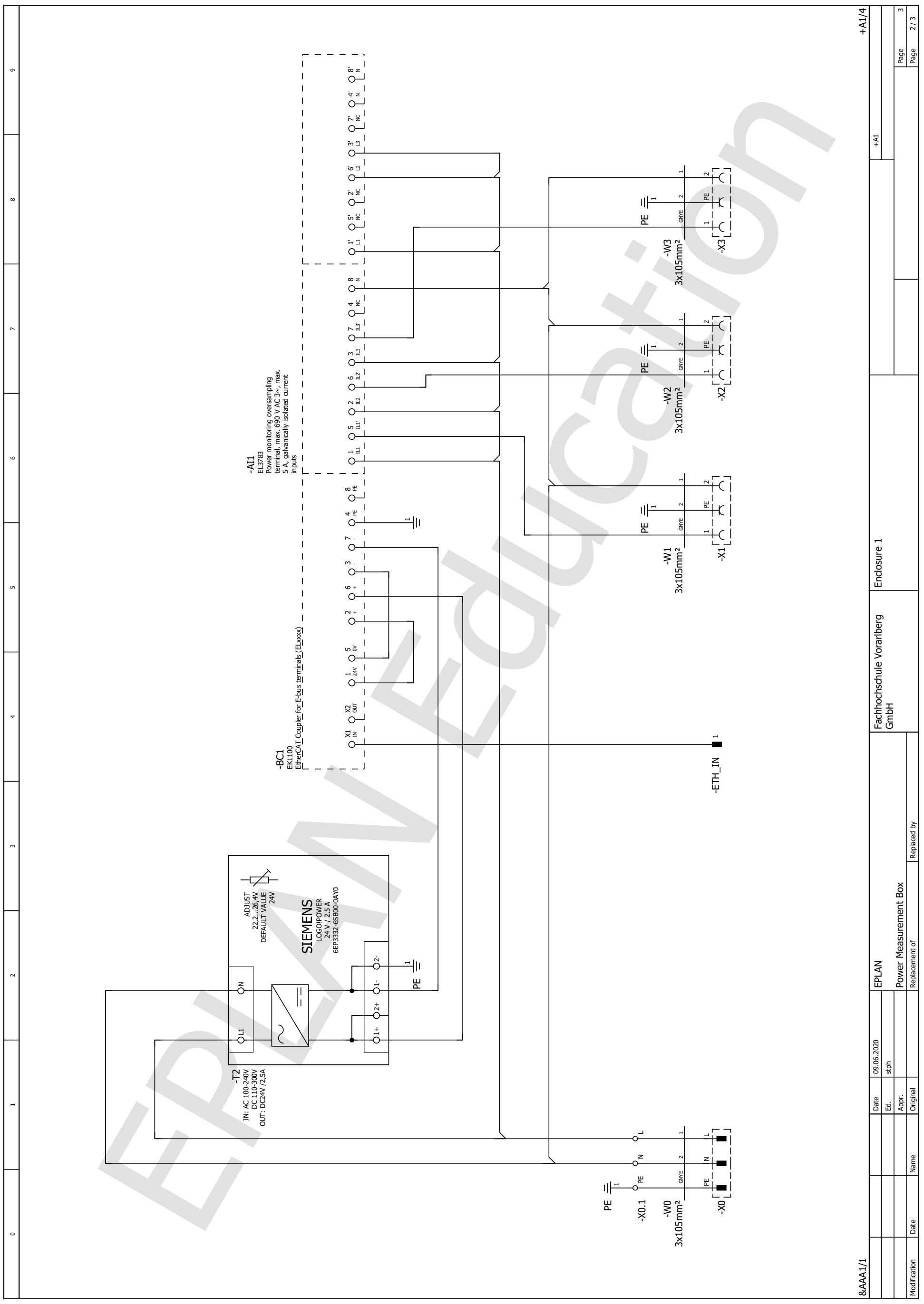
[40] Mingyang Zhang and Jihong Yan. “A data-driven method for optimizing the energy consumption of industrial robots”. en. In: *Journal of Cleaner Production* 285 (Feb. 2021), p. 124862. ISSN: 0959-6526. DOI: 10.1016/j.jclepro.2020.124862.

A. Power Measurement

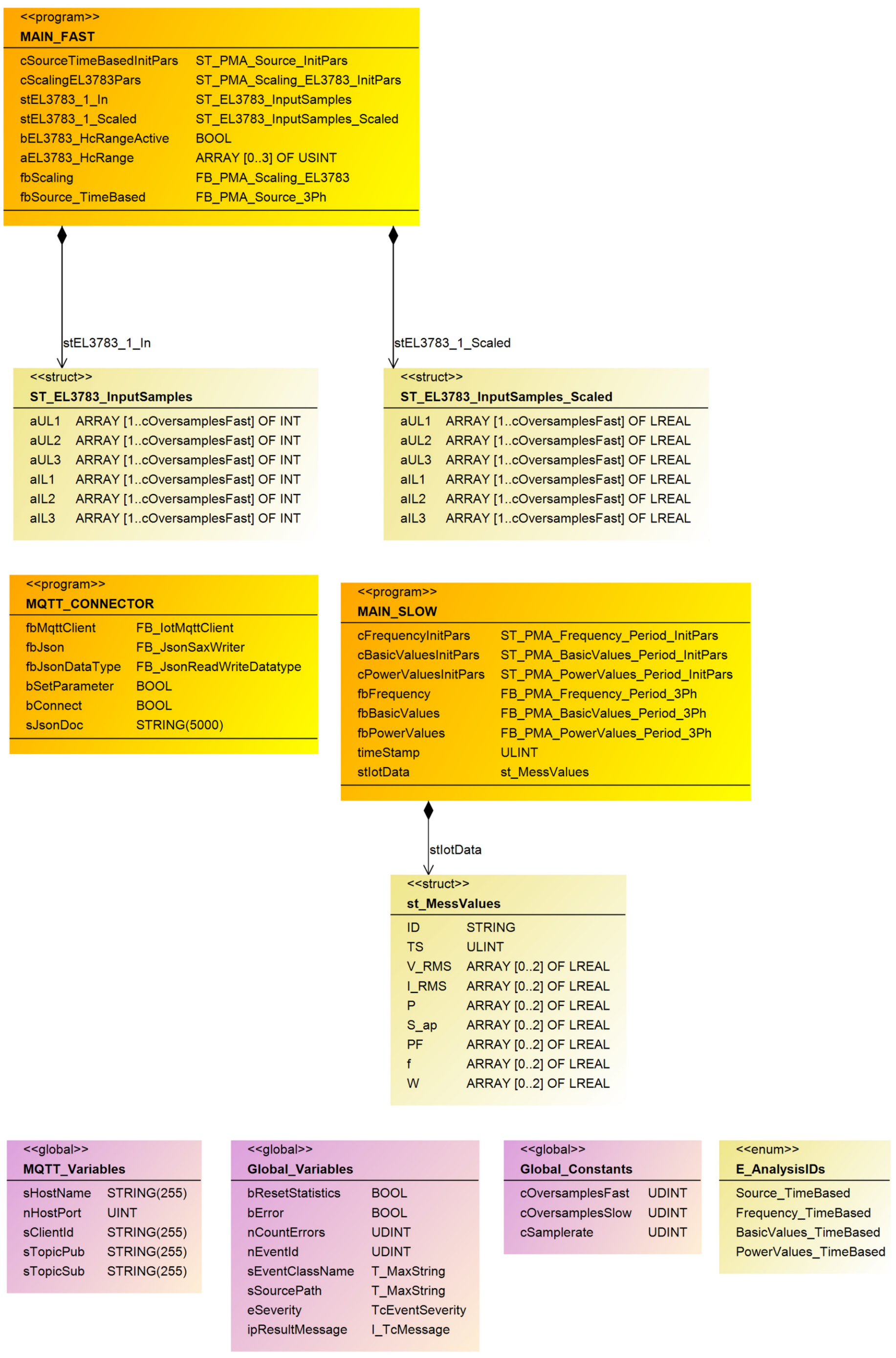
A.1. List of Software Components

- **TwinCAT 3 Engineering**
  Development environment for PLC programming.
  Link to product homepage
- **TwinCAT 3 Runtime**
  Real-time runtime system/environment
- **Utilized TwinCAT libraries**
  PLC Lib: Tc2_Standard
  PLC Lib: Tc2_System
  PLC API: Tc3_IotBase
  PLC Lib: Tc3_JsonXml
  PLC Api: Tc3_PowerMonitoring
- **HiveMQ**
  MQTT-Broker
  Link to product homepage

A.2. Electrical Schematic Measurement Box



A.3. Class Diagram of PLC-Program



<<program>>
MAIN_FAST

cSourceTimeBasedInitPars	ST_PMA_Source_InitPars
cScalingEL3783Pars	ST_PMA_Scaling_EL3783_InitPars
stEL3783_1_In	ST_EL3783_InputSamples
stEL3783_1_Scaled	ST_EL3783_InputSamples_Scaled
bEL3783_HcRangeActive	BOOL
aEL3783_HcRange	ARRAY [0..3] OF USINT
fbScaling	FB_PMA_Scaling_EL3783
fbSource_TimeBased	FB_PMA_Source_3Ph

<<struct>>
ST_EL3783_InputSamples

aUL1	ARRAY [1..cOversamplesFast] OF INT
aUL2	ARRAY [1..cOversamplesFast] OF INT
aUL3	ARRAY [1..cOversamplesFast] OF INT
aIL1	ARRAY [1..cOversamplesFast] OF INT
aIL2	ARRAY [1..cOversamplesFast] OF INT
aIL3	ARRAY [1..cOversamplesFast] OF INT

<<struct>>
ST_EL3783_InputSamples_Scaled

aUL1	ARRAY [1..cOversamplesFast] OF LREAL
aUL2	ARRAY [1..cOversamplesFast] OF LREAL
aUL3	ARRAY [1..cOversamplesFast] OF LREAL
aIL1	ARRAY [1..cOversamplesFast] OF LREAL
aIL2	ARRAY [1..cOversamplesFast] OF LREAL
aIL3	ARRAY [1..cOversamplesFast] OF LREAL

<<program>>
MQTT_CONNECTOR

fbMqttClient	FB_IotMqttClient
fbJson	FB_JsonSaxWriter
fbJsonDataType	FB_JsonReadWriteDatatype
bSetParameter	BOOL
bConnect	BOOL
sJsonDoc	STRING(5000)

<<program>>
MAIN_SLOW

cFrequencyInitPars	ST_PMA_Frequency_Period_InitPars
cBasicValuesInitPars	ST_PMA_BasicValues_Period_InitPars
cPowerValuesInitPars	ST_PMA_PowerValues_Period_InitPars
fbFrequency	FB_PMA_Frequency_Period_3Ph
fbBasicValues	FB_PMA_BasicValues_Period_3Ph
fbPowerValues	FB_PMA_PowerValues_Period_3Ph
timeStamp	ULINT
stIotData	st_MessValues

<<struct>>
st_MessValues

ID	STRING
TS	ULINT
V_RMS	ARRAY [0..2] OF LREAL
I_RMS	ARRAY [0..2] OF LREAL
P	ARRAY [0..2] OF LREAL
S_ap	ARRAY [0..2] OF LREAL
PF	ARRAY [0..2] OF LREAL
f	ARRAY [0..2] OF LREAL
W	ARRAY [0..2] OF LREAL

<<global>>
MQTT_Variables

sHostName	STRING(255)
nHostPort	UINT
sClientId	STRING(255)
sTopicPub	STRING(255)
sTopicSub	STRING(255)

<<global>>
Global_Variables

bResetStatistics	BOOL
bError	BOOL
nCountErrors	UDINT
nEventId	UDINT
sEventClassName	T_MaxString
sSourcePath	T_MaxString
eSeverity	TcEventSeverity
ipResultMessage	I_TcMessage

<<global>>
Global_Constants

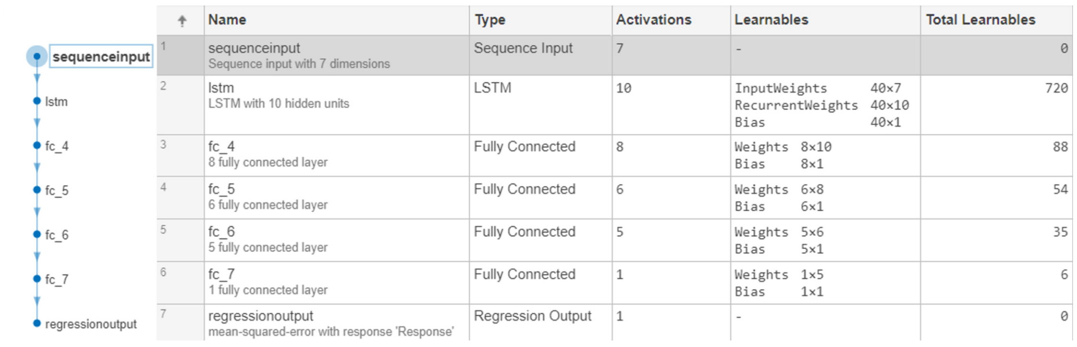
cOversamplesFast	UDINT
cOversamplesSlow	UDINT
cSamplerate	UDINT

<<enum>>
E_AnalysisIDs

Source_TimeBased
Frequency_TimeBased
BasicValues_TimeBased
PowerValues_TimeBased

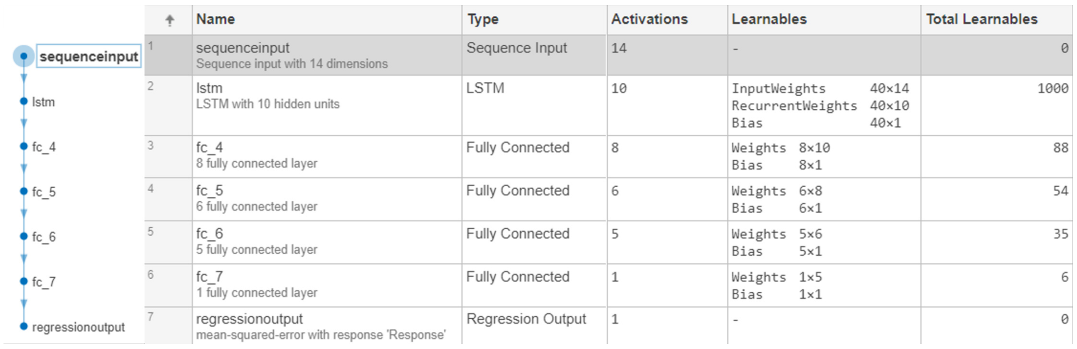
B. Network Structures

B.1. 7x1dim



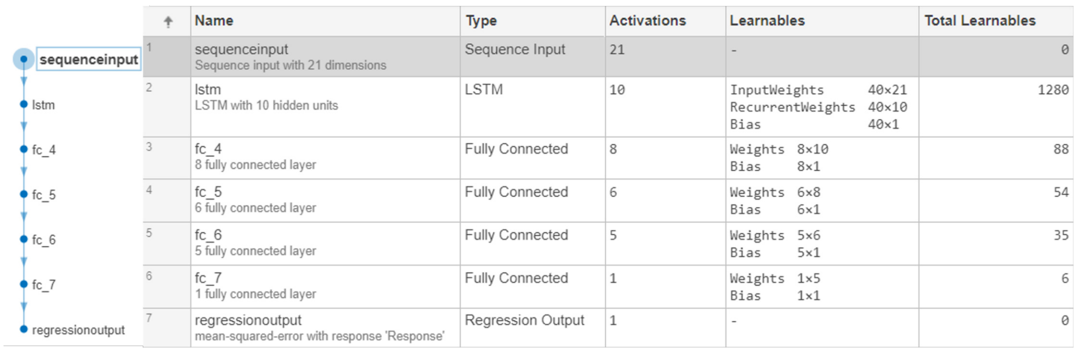
	Name	Type	Activations	Learnables	Total Learnables
1	sequenceinput Sequence input with 7 dimensions	Sequence Input	7	-	0
2	lstm LSTM with 10 hidden units	LSTM	10	InputWeights 40×7 RecurrentWeights 40×10 Bias 40×1	720
3	fc_4 8 fully connected layer	Fully Connected	8	Weights 8×10 Bias 8×1	88
4	fc_5 6 fully connected layer	Fully Connected	6	Weights 6×8 Bias 6×1	54
5	fc_6 5 fully connected layer	Fully Connected	5	Weights 5×6 Bias 5×1	35
6	fc_7 1 fully connected layer	Fully Connected	1	Weights 1×5 Bias 1×1	6
7	regressionoutput mean-squared-error with response 'Response'	Regression Output	1	-	0

B.2. 14x1dim



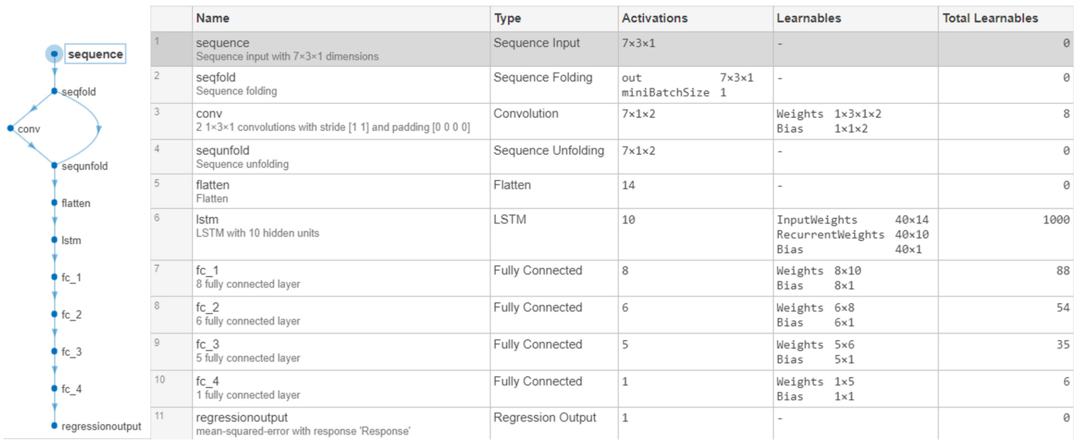
	Name	Type	Activations	Learnables	Total Learnables
1	sequenceinput Sequence input with 14 dimensions	Sequence Input	14	-	0
2	lstm LSTM with 10 hidden units	LSTM	10	InputWeights 40×14 RecurrentWeights 40×10 Bias 40×1	1000
3	fc_4 8 fully connected layer	Fully Connected	8	Weights 8×10 Bias 8×1	88
4	fc_5 6 fully connected layer	Fully Connected	6	Weights 6×8 Bias 6×1	54
5	fc_6 5 fully connected layer	Fully Connected	5	Weights 5×6 Bias 5×1	35
6	fc_7 1 fully connected layer	Fully Connected	1	Weights 1×5 Bias 1×1	6
7	regressionoutput mean-squared-error with response 'Response'	Regression Output	1	-	0

B.3. 21x1dim



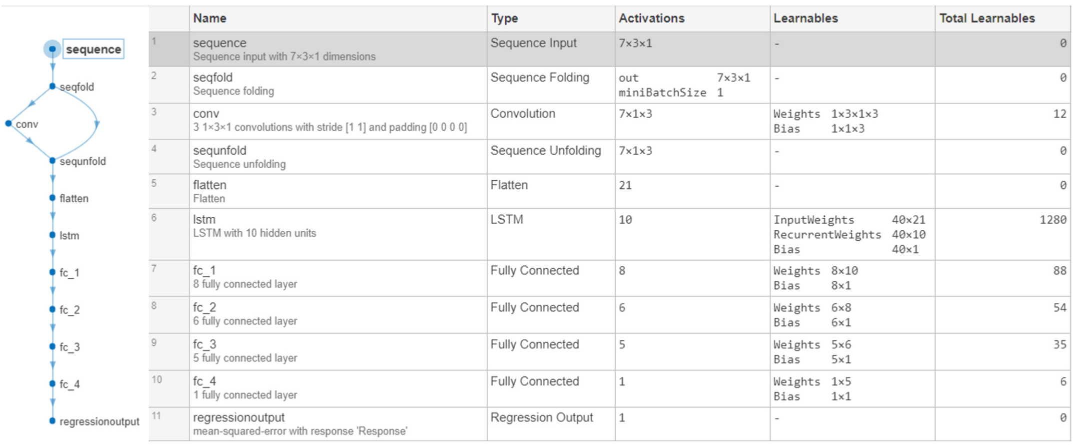
	Name	Type	Activations	Learnables	Total Learnables
1	sequenceinput Sequence input with 21 dimensions	Sequence Input	21	-	0
2	lstm LSTM with 10 hidden units	LSTM	10	InputWeights 40×21 RecurrentWeights 40×10 Bias 40×1	1280
3	fc_4 8 fully connected layer	Fully Connected	8	Weights 8×10 Bias 8×1	88
4	fc_5 6 fully connected layer	Fully Connected	6	Weights 6×8 Bias 6×1	54
5	fc_6 5 fully connected layer	Fully Connected	5	Weights 5×6 Bias 5×1	35
6	fc_7 1 fully connected layer	Fully Connected	1	Weights 1×5 Bias 1×1	6
7	regressionoutput mean-squared-error with response 'Response'	Regression Output	1	-	0

B.4. 7x3dim 2conv1x3



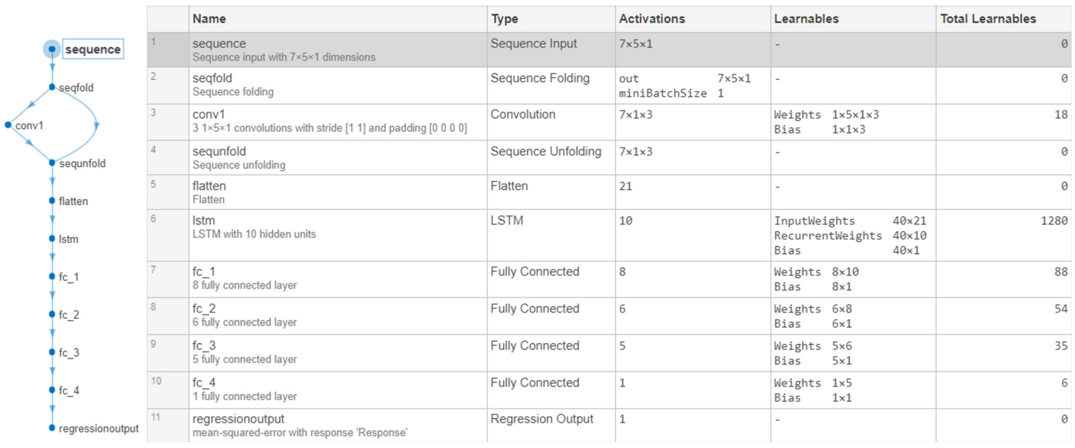
	Name	Type	Activations	Learnables	Total Learnables
1	sequence Sequence input with 7×3×1 dimensions	Sequence Input	7×3×1	-	0
2	seqfold Sequence folding	Sequence Folding	out 7×3×1 miniBatchSize 1	-	0
3	conv 2 1×3×1 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×1×2	Weights 1×3×1×2 Bias 1×1×2	8
4	sequnfold Sequence unfolding	Sequence Unfolding	7×1×2	-	0
5	flatten Flatten	Flatten	14	-	0
6	lstm LSTM with 10 hidden units	LSTM	10	InputWeights 40×14 RecurrentWeights 40×10 Bias 40×1	1000
7	fc_1 8 fully connected layer	Fully Connected	8	Weights 8×10 Bias 8×1	88
8	fc_2 6 fully connected layer	Fully Connected	6	Weights 6×8 Bias 6×1	54
9	fc_3 5 fully connected layer	Fully Connected	5	Weights 5×6 Bias 5×1	35
10	fc_4 1 fully connected layer	Fully Connected	1	Weights 1×5 Bias 1×1	6
11	regressionoutput mean-squared-error with response 'Response'	Regression Output	1	-	0

B.5. 7x3dim 3conv1x3



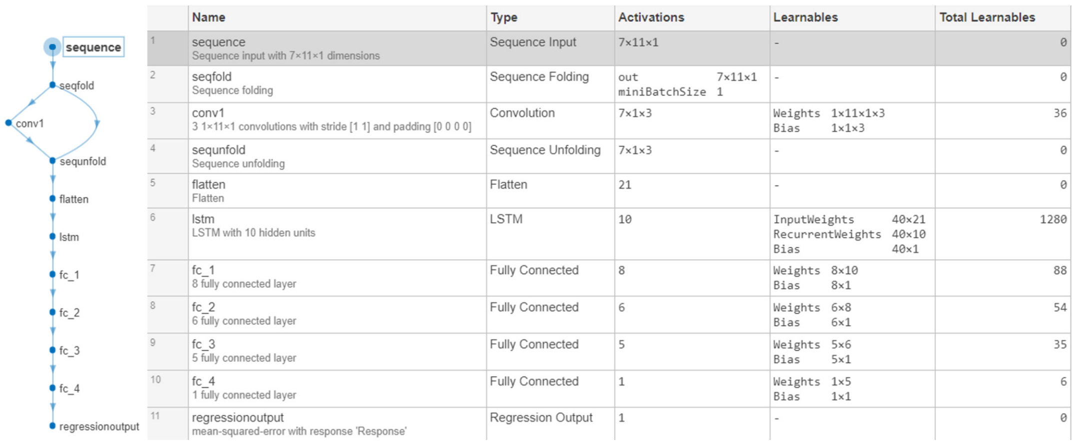
	Name	Type	Activations	Learnables	Total Learnables
1	sequence Sequence input with 7×3×1 dimensions	Sequence Input	7×3×1	-	0
2	seqfold Sequence folding	Sequence Folding	out 7×3×1 miniBatchSize 1	-	0
3	conv 3 1×3×1 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×1×3	Weights 1×3×1×3 Bias 1×1×3	12
4	sequnfold Sequence unfolding	Sequence Unfolding	7×1×3	-	0
5	flatten Flatten	Flatten	21	-	0
6	lstm LSTM with 10 hidden units	LSTM	10	InputWeights 40×21 RecurrentWeights 40×10 Bias 40×1	1280
7	fc_1 8 fully connected layer	Fully Connected	8	Weights 8×10 Bias 8×1	88
8	fc_2 6 fully connected layer	Fully Connected	6	Weights 6×8 Bias 6×1	54
9	fc_3 5 fully connected layer	Fully Connected	5	Weights 5×6 Bias 5×1	35
10	fc_4 1 fully connected layer	Fully Connected	1	Weights 1×5 Bias 1×1	6
11	regressionoutput mean-squared-error with response 'Response'	Regression Output	1	-	0

B.6. 7x5dim 3conv1x5



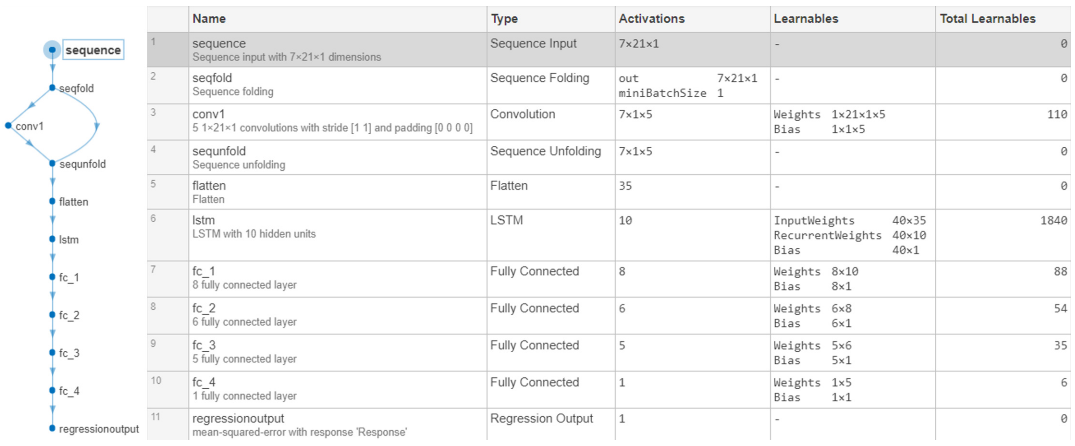
	Name	Type	Activations	Learnables	Total Learnables
1	sequence Sequence input with 7×5×1 dimensions	Sequence Input	7×5×1	-	0
2	seqfold Sequence folding	Sequence Folding	out 7×5×1 miniBatchSize 1	-	0
3	conv1 3 1×5×1 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×1×3	Weights 1×5×1×3 Bias 1×1×3	18
4	sequnfold Sequence unfolding	Sequence Unfolding	7×1×3	-	0
5	flatten Flatten	Flatten	21	-	0
6	lstm LSTM with 10 hidden units	LSTM	10	InputWeights 40×21 RecurrentWeights 40×10 Bias 40×1	1280
7	fc_1 8 fully connected layer	Fully Connected	8	Weights 8×10 Bias 8×1	88
8	fc_2 6 fully connected layer	Fully Connected	6	Weights 6×8 Bias 6×1	54
9	fc_3 5 fully connected layer	Fully Connected	5	Weights 5×6 Bias 5×1	35
10	fc_4 1 fully connected layer	Fully Connected	1	Weights 1×5 Bias 1×1	6
11	regressionoutput mean-squared-error with response 'Response'	Regression Output	1	-	0

B.7. 7x11dim 3conv1x11



	Name	Type	Activations	Learnables	Total Learnables
1	sequence Sequence input with 7×11×1 dimensions	Sequence Input	7×11×1	-	0
2	seqfold Sequence folding	Sequence Folding	out 7×11×1 miniBatchSize 1	-	0
3	conv1 3 1×11×1 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×1×3	Weights 1×11×1×3 Bias 1×1×3	36
4	sequnfold Sequence unfolding	Sequence Unfolding	7×1×3	-	0
5	flatten Flatten	Flatten	21	-	0
6	lstm LSTM with 10 hidden units	LSTM	10	InputWeights 40×21 RecurrentWeights 40×10 Bias 40×1	1280
7	fc_1 8 fully connected layer	Fully Connected	8	Weights 8×10 Bias 8×1	88
8	fc_2 6 fully connected layer	Fully Connected	6	Weights 6×8 Bias 6×1	54
9	fc_3 5 fully connected layer	Fully Connected	5	Weights 5×6 Bias 5×1	35
10	fc_4 1 fully connected layer	Fully Connected	1	Weights 1×5 Bias 1×1	6
11	regressionoutput mean-squared-error with response 'Response'	Regression Output	1	-	0

B.8. 7x21dim 5conv1x21



	Name	Type	Activations	Learnables	Total Learnables
1	sequence Sequence input with 7×21×1 dimensions	Sequence Input	7×21×1	-	0
2	seqfold Sequence folding	Sequence Folding	out 7×21×1 miniBatchSize 1	-	0
3	conv1 5 1×21×1 convolutions with stride [1 1] and padding [0 0 0 0]	Convolution	7×1×5	Weights 1×21×1×5 Bias 1×1×5	110
4	sequnfold Sequence unfolding	Sequence Unfolding	7×1×5	-	0
5	flatten Flatten	Flatten	35	-	0
6	lstm LSTM with 10 hidden units	LSTM	10	InputWeights 40×35 RecurrentWeights 40×10 Bias 40×1	1840
7	fc_1 8 fully connected layer	Fully Connected	8	Weights 8×10 Bias 8×1	88
8	fc_2 6 fully connected layer	Fully Connected	6	Weights 6×8 Bias 6×1	54
9	fc_3 5 fully connected layer	Fully Connected	5	Weights 5×6 Bias 5×1	35
10	fc_4 1 fully connected layer	Fully Connected	1	Weights 1×5 Bias 1×1	6
11	regressionoutput mean-squared-error with response 'Response'	Regression Output	1	-	0

C. Additional Information for Test on Real Use-Cae

C.1. Workflow FMU Generation From MATLAB Neural Network

C.1.1. Requirements

- On Windows, code generation for deep learning networks with the codegen function requires Microsoft Visual Studio or the MinGW compiler
- MATLAB Coder Interface for Deep Learning Libraries
- MATLAB Deep Learning Toolbox
- Trained Neural Network

C.1.2. Code Generation Using Matlab Coder

To generate executable C-Code out of a MATLAB recurrent network the first step is to write an enty-point function that

- uses the *coder.loadDeepLearningNetwork* function to construct and set up a network object
- calls *predictAndUpdateState* method on network with a given input, returns the predicted values and saves the updated network in the persistent variable
- resets the network state and sets the output to zero if reset condition is true

Listing C.1 shows the entry-point function used in this thesis.

Listing C.1: Entry-point function

```matlab
function out = lstm_predictAndUpdate(input,reset)
% A persistent object NET is used to load the series network.
% At the first call to this function,
% the persistent object is constructed and setup.
% When the function is called subsequent times,
% the same object is reused  to predict on inputs.

persistent NET; % instantiate presistent variable

% at the fist call, load the series network into the presistent
    variable
if isempty(NET)
    NET = coder.loadDeepLearningNetwork('Trained_Network.mat');
end

% at reset: reset inner states of network and return 0
%           else predict on the current input data and
%           update network states
if(reset)
    NET = resetState(NET);
    out = single(0);
else
    [NET, out] = predictAndUpdateState(NET,input);
end
end
```

Next, MATLAB coder is utilized to convert the entry-point function into a static library and subsequently save the source files into a package (archive as zip-file) for convenient relocation of the code to another development environment. Listing C.2 shows the script to accomplish this. Note that for correct execution of this code, the Matlab source-file for the entry function must be located in the same directory.

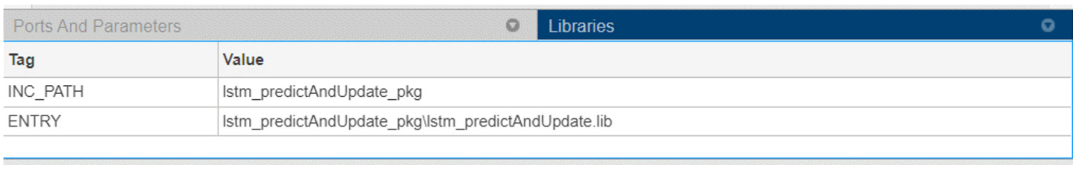
Listing C.2: Matlab script converting entry-point function to C-Code

```matlab
% NET_TO_CODE_SCRIPT
% Generate static library from lstm_predictAndUpdate

%% Create configuration object of class 'coder.
    EmbeddedCodeConfig'.
cfg = coder.config('lib','ecoder',true);
cfg.GenerateReport = true;
cfg.ReportPotentialDifferences = false;

% Create a configuration object of class 'coder.
    DeepLearningConfigBase'.
```

```
cfg.DeepLearningConfig = coder.DeepLearningConfig('
    TargetLibrary', 'none');

%% Define argument types for entry-point 'lstm_predictAndUpdate
    '.
ARGS = cell(1,1);
ARGS{1} = cell(2,1);
ARGS{1}{1} = coder.typeof(0,[21  1]);
ARGS{1}{2} = coder.typeof(false);

%% Invoke MATLAB Coder.
codegen -config cfg lstm_predictAndUpdate -args ARGS{1}

%% Load the buildInfo object
load('codegen\lib\lstm_predictAndUpdate\buildInfo.mat');

%% Create the zip file
packNGo(buildInfo, 'fileName', 'lstm_predictAndUpdate.zip');
```

C.1.3. Embed C-Code within Simulik and Export FMU

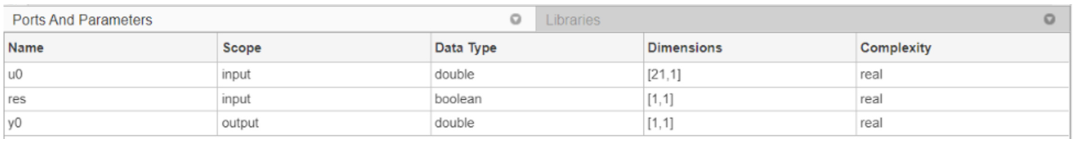
To use the generated code in Simulink, the function block *S-Function Builder* is used. To do this, first create a new Simulink model and insert an S-Function Builder block from the library browser. Then place and extract the source files in the same path as the Simulink model. Open the S-Function block and insert an *S-Function Name* and set *Language* to C. On the Libraries tab, link the source files by adding a new *INC_PATH* and *ENTRY* as shown in Figure C.1.



Tag	Value
INC_PATH	lstm_predictAndUpdate_pkg
ENTRY	lstm_predictAndUpdate_pkg\lstm_predictAndUpdate.lib

Figure C.1.: S-Function library include

In *Ports And Parameters* tab, declare inputs variables as shown in Figure C.2.



Name	Scope	Data Type	Dimensions	Complexity
u0	input	double	[21,1]	real
res	input	boolean	[1,1]	real
y0	output	double	[1,1]	real

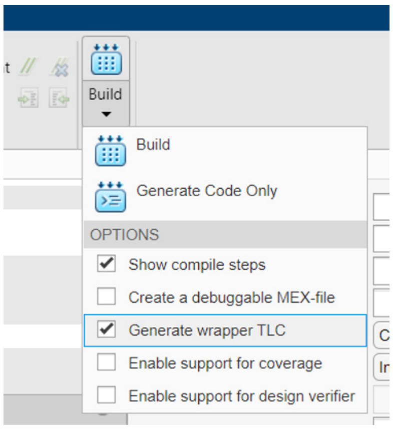
Figure C.2.: S-Function input variable declaration

Include the source h-file within the S-Function Builder and place the updated function within wrapper as shown in Listing C.3

Listing C.3: Code for S-Function Builder

```c
/* Includes_BEGIN */
#include <math.h>
#include "lstm_predictAndUpdate.h"
/* Includes_END */

void s_LSTM_Start_wrapper(void)
{
/* Start_BEGIN */
/* Start_END */
}

void s_LSTM_Outputs_wrapper(const real_T *u0,
                            const boolean_T *res,
                            real_T *y0)
{
/* Output_BEGIN */
y0[0] = lstm_predictAndUpdate(u0,res[0]);
/* Output_END */
}

void s_LSTM_Terminate_wrapper(void)
{
/* Terminate_BEGIN */
/* Terminate_END */
}
```

In the last step, set build options according to Figure C.3, click *Build* and close S-Function Builder after successful build.



- [✓] Show compile steps
- [ ] Create a debuggable MEX-file
- [✓] Generate wrapper TLC
- [ ] Enable support for coverage
- [ ] Enable support for design verifier

Figure C.3.: S-Function Build

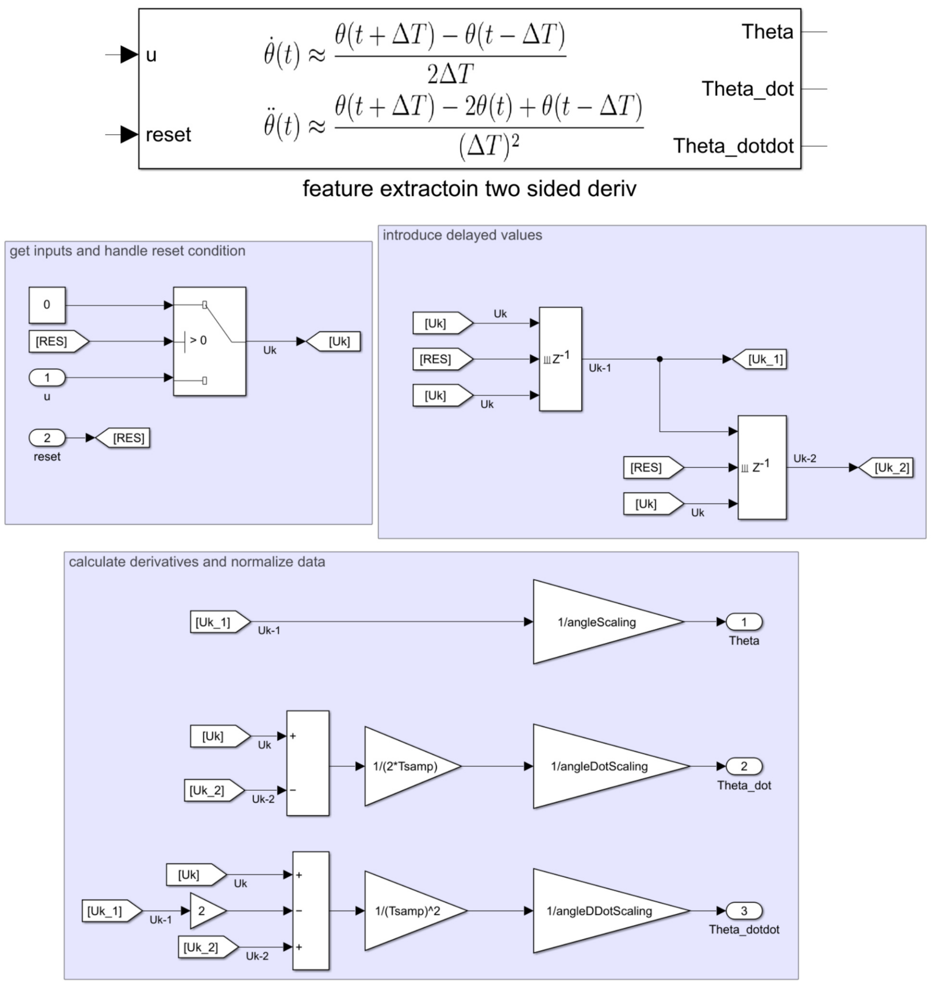
C.1.4. Prepare Simulink model for FMU export

First, setup a matlab script given with Listing C.4 in the same path as the simulink model to generate model parameters.

Listing C.4: Matlab script for parameter setup

```
%% tidy up
clc
clear all
close all

%% generate parameters
Tsamp = 0.04; % simulation timestetp

angleScaling = 3;  % angle scale factor
angleDotScaling = 1.6;  % angular velocity scale factor
angleDDotScaling = 13;  % angular acceleration scale factor
pOffset  = 155;   % power offset
pScale= 166;        % power scale factor

WsTokWh = 1/(3.6e6); % Ws to kWh factor
```

Next, setup a subsystem within the simulink model for feature extraction, as shown in Figure C.4, and place it in front of the S- Function.



$$\dot{\theta}(t) \approx \frac{\theta(t+\Delta T) - \theta(t-\Delta T)}{2\Delta T}$$

$$\ddot{\theta}(t) \approx \frac{\theta(t+\Delta T) - 2\theta(t) + \theta(t-\Delta T)}{(\Delta T)^2}$$

Figure C.4.: Simulink feature extraction

Transform the output of the S-function into unit Watts and handle the reset condition with the blocks shown in Figure C.5. Optionally, add an integrator to obtain the accumulated energy.

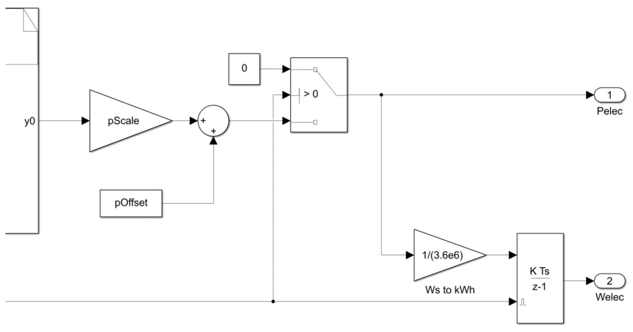
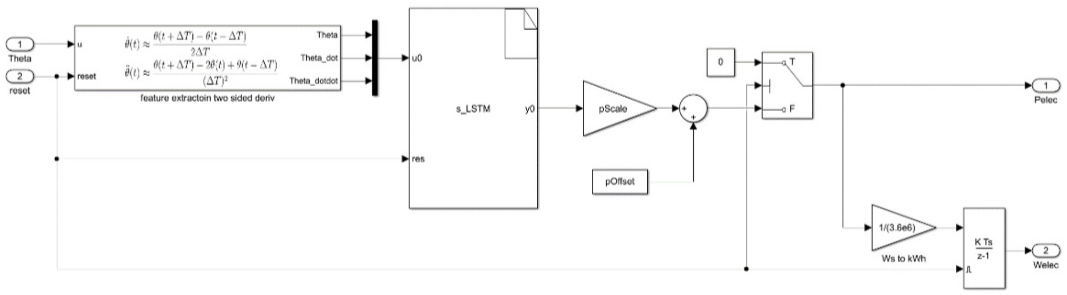
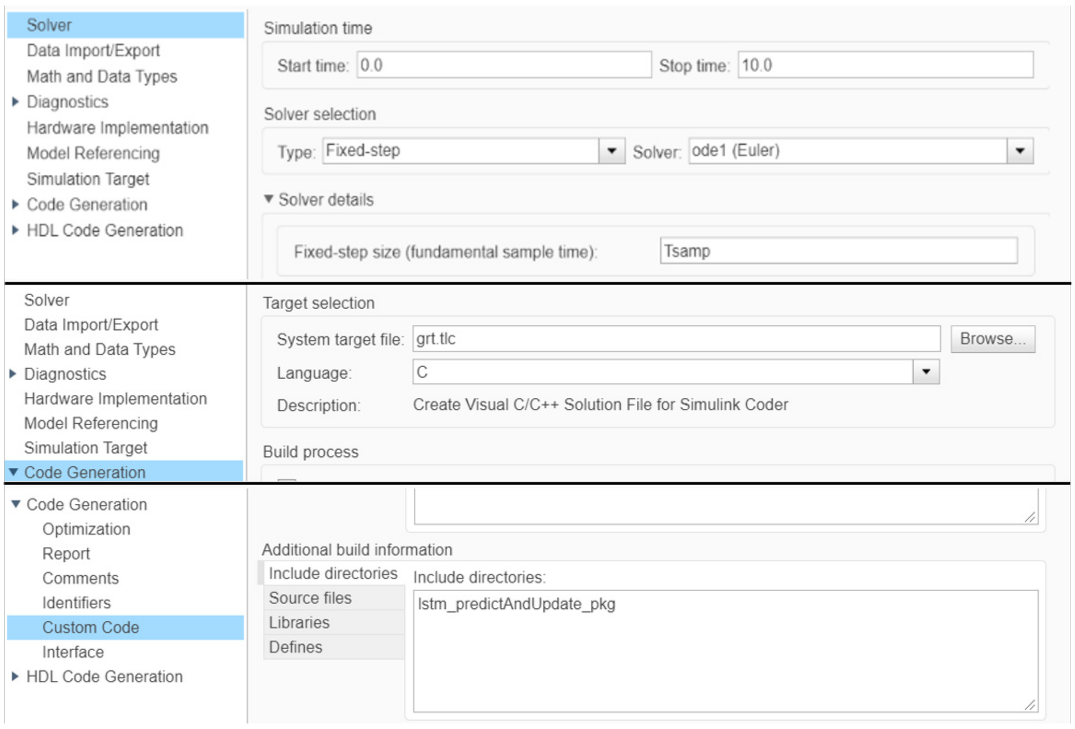


Figure C.5.: Simulink re-scale S-Function output

After adding in- and output ports the model should appear as shown in Figure C.6. Before the model can be exported to an FMU, some additional model


1
Theta
2
reset
u
reset
$\dot{\theta}(t) \approx \frac{\theta(t+\Delta T)-\theta(t-\Delta T)}{2\Delta T}$
$\ddot{\theta}(t) \approx \frac{\theta(t+\Delta T)-2\theta(t)+\theta(t-\Delta T)}{(\Delta T)^2}$
Theta
Theta_dot
Theta_dotdot
feature extractoin two sided deriv
u0
res
s_LSTM
y0
pScale
+
+
pOffset
0
T
F
1
Pelec
1/(3.6e6)
Ws to kWh
K Ts
z-1
2
Welec
Figure C.6.: Simulink FMU template

settings must be made (Figure C.7). In Settings, set the solver to a fixed-step type and specify the step size. In the Code Generation section, ensure that the language is set to C and add the folder with the source files as Additional build information.



Simulation time

Start time: 0.0 Stop time: 10.0

Solver selection

Type: Fixed-step Solver: ode1 (Euler)

Solver details

Fixed-step size (fundamental sample time): Tsamp

Target selection

System target file: grt.tlc

Language: C

Description: Create Visual C/C++ Solution File for Simulink Coder

Build process

Additional build information

Include directories: lstm_predictAndUpdate_pkg

Source files

Libraries

Defines

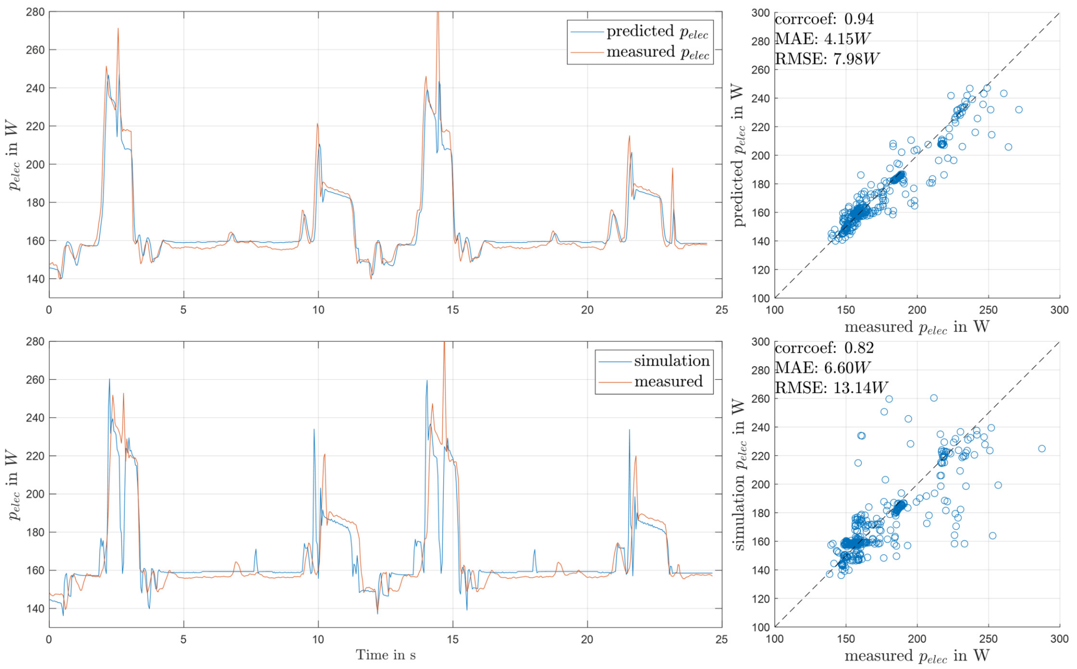
Figure C.7.: Simulink model settings

To generate a co-simulation FMU from the Simulink model, click "Save model as standalone FMU". In the dialog box, uncheck the "Create model after generating standalone FMU" option, specify the path where the FMU should be saved, and click Create.

C.1.5. Useful Links

The following Resources were helpful to set up this workflow.
Prequesites for Deep Learning with MATLAB Coder
Generate Generic C/C++ Code for Deep Learning Networks
Workflow for Deep Learning Code Generation with MATLAB Coder

C.2. Simulation Result Without Initial Power Peak



Statement of Affirmation

I hereby declare that all parts of this thesis were exclusively prepared by me, without using resources other than those stated above. The thoughts taken directly or indirectly from external sources are appropriately annotated. This thesis or parts of it were not previously submitted to any other academic institution and have not yet been published.

Dornbirn, 1.September 2021 Philipp Steurer