

Hybrid Continuous-Time Deep Neural Networks for Robot System Identification and Control

Vom Fachbereich Maschinenbau und Verfahrenstechnik
der Rheinland-Pfälzischen Technischen Universität Kaiserslautern-Landau
zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte

Dissertation

von

Herrn

M.Sc. Jonas Benjamin Weigand

aus Neustadt an der Weinstrasse

Tag der mündlichen Prüfung: 24.10.2023

Dekan: Prof. Dr. rer. nat. Roland Ulber

Promotionskommission:

Vorsitzender: Prof. Dr.-Ing. Hans Hasse

1. Berichterstatter: Prof. Dr.-Ing. Martin Ruskowski

2. Berichterstatter: Prof. Dr.-Ing. Daniel Görge

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und ohne unerlaubte, fremde Hilfe angefertigt habe. Alle Ausführungen, die aus anderen Schriften wörtlich oder sinngemäß übernommen wurden, sind kenntlich gemacht. Alle verwendeten Quellen sind im Literaturverzeichnis zitiert.

Zur sprachlichen und grammatikalischen Korrektur der gesamten Arbeit wurde die Software Grammarly und die Software LanguageTool verwendet. Zur Plagiatsprüfung der gesamten Arbeit wurde am 25.04.2023 die Software PlagAware eingesetzt.

Düsseldorf, den 27. November 2023

JONAS BENJAMIN WEIGAND

Für meine Freunde und Familie

Acknowledgements

I would like to express my sincere appreciation to Prof. Dr.-Ing. Martin Ruskowski, my doctoral supervisor, for his guidance and expertise. His knowledge and passion for the subject have been instrumental in shaping this research. I am thankful for his support and the opportunity to work under his supervision.

I would like to extend my sincere appreciation to Prof. Dr.-Ing. Daniel G6rges, the second supervisor of my thesis. His guidance and support have been invaluable to the success of my thesis. I would also like to acknowledge Dr.-Ing. habil. Achim Wagner, the head of research, for his valuable guidance and support throughout the research process. I am grateful for his continuous support, valuable insights, and commitment to my academic journey. In addition, I would like to express my gratitude to Prof. Dr.-Ing. Hans Hasse for taking over the chairmanship of the doctoral committee.

Regarding the robotics topic, I am particularly grateful to Julian G6tz and Jonas Ulmen for their contributions to the development and validation of the robot controller. Their insights and dedication have greatly enhanced the quality of this research. Furthermore, I would like to extend my appreciation to the team of individuals who played a vital role in the development, implementation, and testing of the industrial robot. These include in alphabetical order Dr.-Ing. habil. Achim Wagner, Aleksander 'Sasha' Sidorenko, Alexandre Janot, Andreas Wagner, Bj6rn Dietrichs, Christopher Lippert, Florian Wendling, Gajanan Kanagalingam, Joachim Spangenberg, Jonas Ulmen, Julian G6tz, Leon Hensel, Magnus Volkmann, Marco Sprenger, Prof. Dr.-Ing. Martin Ruskowski, Nigora Gafur, Niklas Hagen, Patrick Kremser, R6diger Ruppert, Sebastian Harttig, Sergej Gertje, Simon Lamoth, Stephan Belz, Sven Varelmann, William Motsch, and Xiaohai Wang.

Regarding the data-based methods, I am grateful to Magnus Volkmann for his contributions to our first publications and Prof. Dr.-Ing. Martin Ruskowski for his scientific guidance throughout the scientific process. Special recognition goes to Gerben Beintema, Jonas Ulmen, Prof. Dr.-Ing. Daniel G6rges, Asst. Prof. Roland T6th, Asst. Prof. Maarten Schoukens, and Prof. Dr.-Ing. Martin Ruskowski for their collaboration on the topic of state derivative normalization. Their expertise and insights have significantly enriched my understanding of this phenomenon. Furthermore, I would like to acknowledge Dr.-Ing. Michael Deflorian for his doctoral thesis. His work laid the groundwork for my research, and I am grateful for his contributions. I would also like to thank Vas-

silios Yfantis for insightful discussions regarding the stability constraints. I extend my gratitude to Julian Raible and Nico Zantopp for their theses on continuous-time neural networks, which have provided valuable insights and perspectives in the development of this research. Together with Dr.-Ing. Demir Ozan, Dr.-Ing. Adrian Trachte, Dr.-Ing. habil. Achim Wagner and Prof. Dr.-Ing. Martin Ruskowski, we published a great paper on hybrid data-driven inverse control. I would like to express my gratitude to Jonas Ulmen and Gajanan Kanagalingam for many invaluable scientific discussions until late after midnight. Your insights and perspectives have greatly enriched my research, and I am thankful for the intellectual exchange we have had.

I would like to extend my heartfelt gratitude to my dear friends, Dorian Hargarten, Lukas Weiner, Malte Weirauch, Matthias Sawatzski, Nils Habermehl, Sebastian Raffauf, and Tobias Reinmuth. Their unwavering support, encouragement, and friendship have been invaluable throughout my academic journey. I would like to express my special thanks to Jonas Ulmen, Lukas Weiner, and Sebastian Raffauf for their diligent proofreading of the thesis. Their keen attention to detail and constructive feedback have significantly improved the clarity and quality of this work. I am truly grateful for their time and effort in helping me refine my ideas and enhance the overall presentation.

I would like to take a moment to express my heartfelt appreciation to my family, grandparents, and my partner for their unwavering love, encouragement, and support throughout my academic journey. I am truly fortunate to have such a loving and supportive family, grandparents, friends, and partner. Thank you for always being there for me, for celebrating my successes, and for being my pillars of strength during difficult times. I am eternally grateful for your love and support.

To all those mentioned above, as well as anyone else who has contributed to this thesis in any way, I offer my sincere thanks. Your support and collaboration have been instrumental in the successful completion of this work.

Table of Contents

Abstract	I
Kurzfassung	II
List of Abbreviations	III
1 Introduction	1
1.1 Problem Definition	3
1.2 Objectives	6
1.3 Approach	7
2 Foundations of Physical Robot Modeling and Control	9
2.1 State-of-the-Art Literature Robotic Modelling and Control	9
2.1.1 Robot Machining	9
2.1.2 Nonlinear Flexible Joint Model	10
2.1.3 Robot Parameter Identification	11
2.1.4 Nonlinear Robot Control	12
2.2 Nonlinear Robot Model	13
2.2.1 Kinematic Model	13
2.2.2 Rigid Joint Model	13
2.2.3 Rigid Joint Feed-Forward Controller	14
2.2.4 Friction Model	14
2.2.5 Inertia, Coriolis, Centripetal, and Gravitational Model	15
3 Foundations of Continuous-Time Neural Networks	17
3.1 State-of-the-Art Literature Neural Networks	17
3.1.1 System Identification with Neural Networks	17
3.1.2 Discrete-Time Neural Networks	18
3.2 General Form of Continuous-Time Neural Networks	20
3.3 ODE Configuration	22
3.3.1 Explicit Fixed-step Runge-Kutta Solvers	22
3.3.2 Euler Neural Networks	23
3.3.3 Runge-Kutta Neural Networks	24

3.3.4	Neural Ordinary Differential Equations	25
3.4	Neural Network Configuration	26
3.4.1	Activation Functions	27
3.4.2	Model Ensemble	29
3.5	Training Configuration	30
3.5.1	Loss Functions	30
3.5.2	Inequality Barrier Methods	33
3.5.3	Equality Barrier Methods	34
3.5.4	Reduction of Overfitting	35
3.5.5	Forecast Horizon	37
3.5.6	Discretize-then-Optimize vs. Optimize-then-Discretize	39
4	Physical Robot Model and Control	41
4.1	Advanced Nonlinear Robot Model	41
4.1.1	Elastic-Joint Model	41
4.1.2	Stiffness Model	42
4.1.3	Advanced Inertia, Coriolis, Centripetal and Gravitational Model	45
4.1.4	Hydraulic Weight Counterbalance	46
4.1.5	Advanced Friction Model	48
4.2	Friction Parameter Estimation	49
4.2.1	Identification Algorithm	49
4.2.2	Design of Experiments	51
4.2.3	Experimental Setup	52
4.2.4	Single Batch Identification	53
4.2.5	Computational Efficiency of Multi Batch Identification	57
4.3	Robot Control	59
4.3.1	Control Structure	59
4.3.2	Feedback Controller using Secondary Encoders	62
4.3.3	Flatness-based Feed-Forward Controller	63
4.4	Originality and Background	65
5	Continuous-Time Neural Networks	67
5.1	State Derivative Normalization	67
5.1.1	Normalization Definition	68
5.1.2	Normalization Factor Estimation	71
5.1.3	Empirical Normalization Results	74
5.2	Stability Constraints	80
5.2.1	Empirical Stability Observation	80
5.2.2	Preliminaries for Stability	85
5.2.3	Existence and Uniqueness of Equilibrium	86

5.2.4	Constraints on the Neural Network Parameters	87
5.2.5	Weak Stability Method	89
5.2.6	Empirical Stability Results	90
5.2.7	Additional Methods to Improve Model Stability	91
5.3	Demonstration of Continuous-Time Memory Efficiency	92
5.3.1	Discrete-Time Baseline	92
5.3.2	Continuous-Time Baseline	94
5.3.3	Baseline Comparison	96
5.4	Originality and Background	100
6	Comparison of Physical, Data-based, and Hybrid Models	103
6.1	Design of Experiments	103
6.2	Model Configuration	111
6.2.1	Physical Model Configuration	111
6.2.2	Data-based Model Configuration	111
6.2.3	Hybrid Model Configuration	113
6.3	Experimental Results	117
6.3.1	Comparison of Robot Model Accuracy	117
6.3.2	Comparison of Robot Path Accuracy	121
6.3.3	Comparison of Robot Model Safety	127
7	Summary and Outlook	131
Appendix		137
A	Proof of Stability Theorem	137
B	Technical Robot Details	141
B.1	Dimensions and Limits	141
B.2	Robot Controller Layout	143
B.3	Sensor Specification	146
B.4	Physical Robot Model Parameters	148
C	Additional Experiments Physical Model	149
C.1	Simulation Results	149
C.2	Experimental Results	154
C.3	Improvements Based on Experimental Results	159
D	Additional Experiments Data-based Model	161
E	Source Code	168
E.1	Dynamic Model Update	168
E.2	Advanced Friction Flatness-based Feed-forward Control	169
E.3	Feed-Forward Controller Code	171
E.4	Weak Stability Barrier	175

E.5	Differential Algebraic Equation Barrier	176
Bibliography		179
	Articles and Monographs	179
	Own Publications	194
	Supervised Student Thesis	195
Curriculum Vitae		198

Abstract

Industrial robots are vital in automation technology, but their limitations become evident in applications requiring high path accuracy. This research focuses on improving the dynamic path accuracy of industrial robots by integrating additional sensor technology and employing intelligent feed-forward control. Specifically, the inclusion of secondary encoder sensors enables explicit measurement and compensation of robot gear deformations. Three types of model-based feed-forward controllers, namely physics-based, data-based, and hybrid, are developed to effectively counteract dynamic effects.

Firstly, a physics-based feed-forward control method is proposed, explicitly modeling joint deformations, hydraulic weight compensation, and other relevant features. Nonlinear friction parameters are accurately identified using a globally optimized design of experiments. The resulting physics-based model is fully continuously differentiable, facilitating its transformation into a code-optimized flatness-based feed-forward control.

Secondly, a data-based feed-forward control approach is introduced, leveraging a continuous-time neural network. The continuous-time approach demonstrates enhanced model generalization capabilities even with limited data. Furthermore, a time domain normalization method is introduced, significantly improving numerical properties by concurrently normalizing measurement timelines, robot states, and state derivatives. Based on previous work, a method ensuring input-to-state and global-asymptotic stability is presented, employing a Lyapunov function. Model stability is enforced already during training using constrained optimization techniques. Moreover, the data-based methods are evaluated on public benchmarks, extending its applicability beyond the field of robotics.

Both the physics-based and data-based models are combined into a hybrid model. Comparative analysis of the three models reveals that the continuous-time neural network yields the highest model accuracy, while the physics-based model delivers the best safety properties. The effectiveness of all three models is experimentally validated using an industrial robot.

Kurzfassung

Industrieroboter sind in der Automatisierungstechnik weit verbreitet. Für Anwendungen, welche eine hohe Präzision erfordern, stoßen Industrieroboter jedoch an ihre Grenzen. Diese Arbeit verfolgt den Ansatz, die dynamische Bahngenauigkeit von Industrierobotern durch zusätzliche Sensorik und eine intelligente Vorsteuerung zu verbessern. Dabei wird eine abtriebsseitige Sensorik integriert, welche die Verformung von Robotergetrieben explizit messen und kompensieren kann. Um darüber hinaus dynamischen Effekten entgegen zu wirken, wird eine physikbasierte, eine datenbasierte und eine hybride Vorsteuerung entwickelt und evaluiert.

Erstens wird eine physikbasierte Vorsteuerung entwickelt, welche explizit die Gelenkverformungen, den hydraulischen Gewichtsausgleich und andere relevante Eigenschaften modelliert. Nicht lineare Reibungsparameter werden durch eine global optimierte Versuchsplanung präzise identifiziert. Das nicht lineare physikbasierte Modell ist vollständig stetig differenzierbar und kann dadurch in eine codeoptimierte flachheitsbasierte Vorsteuerung überführt werden.

Zweitens wird eine datenbasierte Vorsteuerung entwickelt, welche auf einem zeitkontinuierlichen neuronalen Netz basiert. Der zeitkontinuierliche Ansatz zeigt eine Verbesserung der Generalisierungsfähigkeit, selbst bei einer begrenzten Datenmenge. Erweitert wird der Ansatz um eine Normalisierungsmethode im Zeitbereich, welche die numerischen Eigenschaften deutlich verbessert, in dem die Zeitpunkte der Messungen, die Zustände des Roboters sowie deren Abteilungen gemeinsam normalisiert werden. Aufbauend auf Vorarbeiten wird zudem eine Methode zur Garantie von Input-To-State und Global-Asymptotischer-Stabilität vorgestellt, welche auf einer Lyapunov Funktion basiert. Mittels beschränkter Optimierungsverfahren kann die Stabilität bereits während des Trainings garantiert werden. Die Arbeiten zur datenbasierten Modellierung werden zusätzlich auf Anwendungen getestet, welche über den Bereich der Robotik hinaus gehen.

Das physikbasierte und datenbasierte Modell werden zu einem hybriden Modell kombiniert. Der Vergleich der drei Modelle zeigt, dass das zeitkontinuierliche neuronale Netzwerk eine überlegene Modellgenauigkeit liefert, während das physikbasierte Modell hervorragende Sicherheitseigenschaften aufweist. Die Effektivität aller drei Modelle wird anhand eines Industrieroboters experimentell nachgewiesen.

List of Abbreviations

AD	Automatic Differentiation	6
AIC	Akine Information Criterion	27
BIC	Bayes Information Criterion	27
BLA	Best Linear Approximation	26
BPTT	Backpropagation Through Time	18
CAD	Computer Aided Design	15
C-FB	Conventional Feedback Controller	149
CNC	Computerized Numerical Control	1
CNF	Continuous Normalizing Flows	17
CPU	Central Processing Unit	52
CTC	Computed Torque Control	59
CTS	Cascaded Tank System	67
DAE	Differential Algebraic Equations	22
DFT	Discrete Fourier Transform	73
Disc-Opt	Discretize-then-Optimize	27
DoF	Degrees of Freedom	7
EMPS	Electro Mechanical Positioning System	67
FB-FF	Flatness-Based Feed-Forward Controller	149
FEM	Finite Element Methods	15
GAS	Global Asymptotic Stability	18
GRU	Gated Recurrent Units	20
HWC	Hydraulic Weight Counterbalance	3
IMU	Inertial Measurement Unit	20
ISS	Input to State Stability	18
LMI	Linear Matrix Inequalities	137
LSTM	Long Short Term Memory	17
LTC	Liquid Time Constant	17
MAE	Mean Absolute Error	31
MAPE	Mean Absolute Percentage Error	31
MB-FB	Model-Based Feedback Controller	149
MB-FF	Model-Based Feed-Forward Controller	154
MLP	Multi Layer Perceptron	92
MPC	Model Predictive Control	39
MSE	Mean Squared Error	22
NARX	Nonlinear Autoregressive with Exogenous Input	18

NCDE	Neural Controlled Differential Equations	20
NLSS	Nonlinear State-Space Model	78
NN	Neural Network	4
NODE	Neural Ordinary Differential Equations	6
NRMSE	Normalized Root Mean Squared Error	31
ODE	Ordinary Differential Equations	8
OPC-UA	Open Platform Communications Unified Architecture	143
Opt-Disc	Optimize-then-Discretize	39
PD	Proportional Derivative	13
PID	Proportional Integral Derivative	12
PLC	Programmable Logic Controller	127
ReLU	Rectified Linear Function	72
R-FF	Rigid-Model Feed-Forward Controller	149
RK	Runge-Kutta	18
RKNN	Runge-Kutta Neural Networks	17
RMSE	Root Mean Squared Error	31
RNN	Recurrent Neural Network	17
ROS	Robot Operating System	4
RPTU	Rheinland-Pfälzische Technische Universität	2
R^2	Coefficient of Determination	31
SDN	State Derivative Normalization	67
SE	Secondary Encoders	1
SMAPE	Smoothed Mean Absolute Percentage Error	31
TCP	Tool Center Point	1
TSEM	Truncated Simulated Error Minimization	26
VJM	Virtual Joint Method	11

1 Introduction

Industrial robots are a key element of automation technology. According to ISO 8373 [21] an industrial robot is defined as “automatically controlled, reprogrammable multipurpose manipulator programmable in three or more axes”. Typical applications of industrial robots include welding, painting, assembly, pick-and-place tasks, packaging, and palletizing. In the year 2022, 3.02 million industrial robots were in operation worldwide [IF22]. Industrial robots are mainly designed for good repeatability but not for high-precision tasks, such as milling or robot machining. An example of an industrial robot in a machining cell is given in Fig. 1.1. The main drawback of using an industrial robot in machining processes is its low stiffness, for example, caused by joint elasticity between the actuators and the driven links [Fr14]. As a result, conventional Computerized Numerical Control (CNC) machines, possessing high stiffness and simple kinematics, can still not be replaced by industrial robots for machining tasks. Furthermore, the low and highly pose-dependent overall stiffness of industrial robots, up to 100 times less than conventional CNC machines, can lead to vibrations and chattering effects so that the machining quality suffers [Br18; Yu18].

Nevertheless, robot machining is an interesting field of application, as the long reach of the robot enables operations on large workpieces. CNC machines, on the contrary, can typically only process smaller workpieces. The low stiffness and low robot’s Tool Center Point (TCP) path accuracy are core hurdles for enabling robot machining applications with industrial robots.

One possibility to improve path accuracy is by utilizing additional sensors, which measure path deviations in terms of joint deformations and enhance the robot control software. Compared to hardware upgrades for drivetrain and mechanical components a software-based approach is more cost-effective. Besides traditional control design approaches, machine learning techniques made substantial progress in the past decades, including system identification and control.

A sensor suitable for measuring joint deformations is the so-called Secondary Encoders (SE). Attaching SE to the link side can reduce the oscillatory behavior and chattering effects by providing sensor information to a feedback controller. An improvement of static precision by a factor of 10 was achieved in [De11c] using SE. Drive-based damping utilizing SE was developed in [VKL17]. Nevertheless, the approach is associated with high costs,

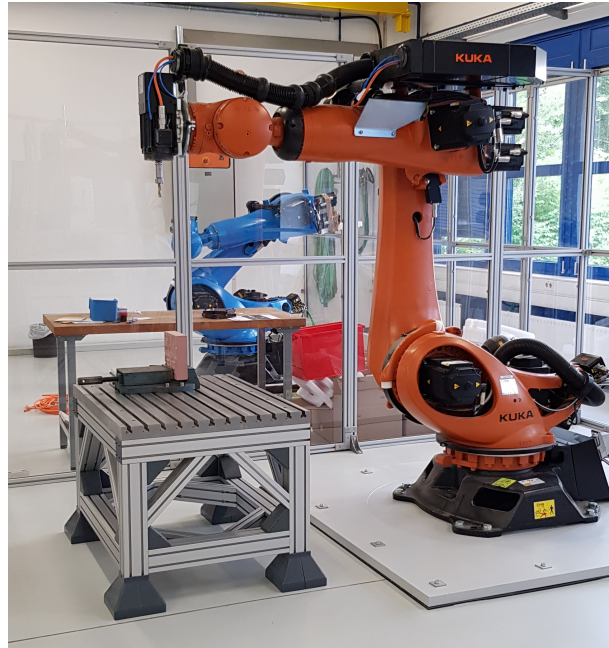


Figure 1.1: Picture of a robot machining cell at the Chair of Machine Tools and Control Systems, Rheinland-Pfälzische Technische Universität (RPTU) Kaiserslautern-Landau.

high implementation effort, communication delays, and sensor noise [Sc14]. Furthermore, reducing dynamic deflections due to time-varying dynamical effects remains a challenge for feedback controllers. Knowledge of nonlinear deterministic effects, such as pose-dependent stiffness, lost motion, backlash, and friction cannot be accounted for in the feedback control loop.

To improve path accuracy, a sophisticated feedback and feed-forward controller is a promising solution. To build model-based feed-forward controllers, an accurate and reliable robot model is required. The feedback control can compensate for external forces and unknown disturbances. The feed-forward control can compensate for deterministic dynamic effects in advance. Any model errors might lead to path deviations, which need to be additionally handled by the feedback control. In contrast, if the feed-forward model is accurate, less path deviation occurs in the first place. Moreover, the feedback control can be designed stiffer to compensate for external forces even better.

The problem of designing a proper model-based feed-forward controller is broader than solely considering its contribution to path accuracy. As it is a safety-critical task, the reliability of the controller is a core criterion, too. Furthermore, in order to extend the developed results to different industrial robots, development and adaptation effort needs to be considered, too.

Two feed-forward model types are subject to this thesis, a physical model and a data-based model. An additional hybrid model combines the physical and data-based one. Each type brings its advantages and disadvantages [Sc16a]. A brief look at the shortcomings of

physical and data-based models is presented in the following.

1.1 Problem Definition

Shortcomings of Physical Robot Models

Most state-of-the-art physical models are based on rigid joints. An extension to flexible joints is at least required for model-based development of the SE-based feedback control loop. Although a flexible joint model is not obligatory for feed-forward control, it can greatly improve its performance by incorporating additional physical knowledge. Flexible joint models can account for gearbox deformations, such as backlash, lost motion, and linear stiffness. Although flexible joint models can predict the robot's state accurately, state-of-the-art flexible joint models are not continuously differentiable. Consequently, these models are detrimental to the downstream software architecture. Applying the model in parameter optimization algorithms is at least with all gradient-based optimizers elaborately. Furthermore, code generation is beneficial to derive an optimized feed-forward controller. Typical symbolic code generation tools can hardly cope with a non-differentiable source code.

The Hydraulic Weight Counterbalance (HWC) attached to the second joint of the robot has a significant impact on the effective gravitational load of this axis. However, the literature lacks geometric and fluid dynamic models to account for this influence. Model errors of the second joint are due to the long lever especially severe in terms of additional path deviations.

The friction of the robot is temperature-dependent and therefore potentially changes at run time. Several approaches have been made to address this issue [Ku08]. However, most approaches are predetermined to specific model formulations. From a software development perspective, an algorithm that can optimize for arbitrary model parameters at runtime is beneficial. This feature would decouple robot modeling from robot model parameter identification. However, this task is very demanding, as it requires very efficient optimization strategies, including model formulation and optimization algorithm, and a sufficient excitation of parameters.

Finally, a feed-forward control algorithm is required, which incorporates the advanced nonlinear model. In an ideal setting, this algorithm is computationally efficient and implemented on a low-level controller. In the best case, the feed-forward control algorithm is given in a single, compilable programming language, without additional software library dependencies or any approximations of the nonlinear model.

Shortcomings of Data-based Robot Models

Data-based models are an alternative to physical models. However, data-based models must be handled with care when applied directly as control, as extrapolation might cause safety issues. Furthermore, data-based models are not 100% explainable like first-principle expert models are. In addition, applying the data-based model to the robot in a sample time of four ms makes human validation of each prediction impossible. For these three reasons - uncertain extrapolation, lack of model explainability, and unfeasible human validation at runtime - safety issues are one major shortcoming of data-based models applied to industrial robots.

Besides safety issues, data-based approaches potentially suffer from requiring large amounts of training data. Data acquisition is time-consuming and expensive. If the data is provided by another research group or direct access to the demonstrator is not possible, collecting additional data is even infeasible. In general, providing methods which can create valuable models using few data is desirable.

Furthermore, data is in general not collected in a uniform time grid, yet discrete-time models require uniform data collection. Especially for non-real-time systems such as the robot middleware Robot Operating System (ROS), this issue is present. Therefore, some data preprocessing such as interpolation, missing value information estimation, or partial data deletion is required. However, these methods manipulate the data, potentially losing and adding artificial information. Moreover, in general, the non-availability of measurements is also a piece of information, which is erased using these methods. Models which could handle arbitrary time grids are beneficial.

An important choice to address the issues mentioned before is the choice of the data-based model architecture: discrete-time or continuous-time Neural Network (NN). Discrete-time models directly predict the succeeding state given the current measured state. Continuous-time models predict the change rate based on the current measured state, analog to differential equations. Regarding the issue of irregularly collected data, continuous-time models shine. Continuous-time models can directly handle any, including irregular, time grids [An23; De20; LH20; RCD19].

Yet, discrete-time models are typically employed for robot manipulator control [Ji18; MAS03]. This is beneficial if the data is measured at discrete-time steps, e.g., every millisecond for a technical process. Real-time digital control devices work at discrete, usually uniform, time grids. However, the real world states are continuous. Discrete-time models structurally ignore that technical processes are modeled as differential equations and that objects in the real world obey the laws of physics. Physical expert models are often defined as change rates and are not just a time-independent state without any relation to its history.

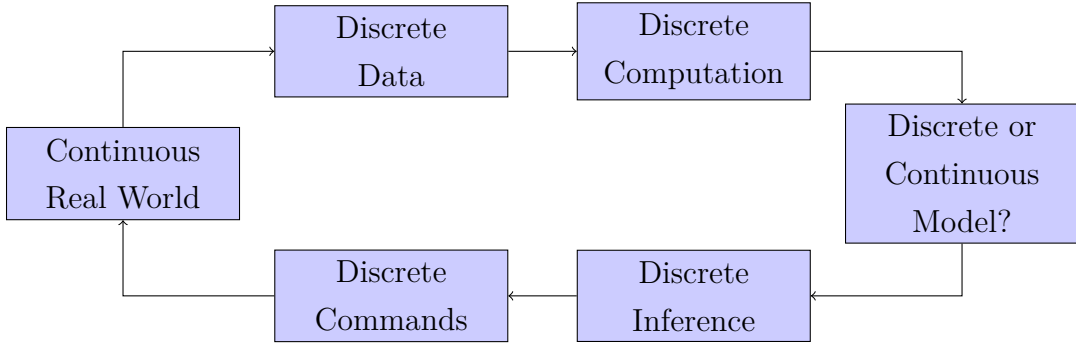


Figure 1.2: Discrete and continuous data-based workflow.

Consequently, the digital tools (data and computational resources) are discrete-time, yet the ground-truth underlying dynamics are continuous-time. This leads to motivation for both discrete-time and continuous-time models. Discrete-time models are often easier to implement, as hardware and data sets are usually already in discrete-time. Continuous-time models require additional algorithms and transformations, yet the model space is closer to the ground-truth process. The data-based workflow is depicted in Fig. 1.2. The real world process is typically in continuous-time. Data acquisition, model computation, model inference, and control command are in discrete-time, if a digital controller is utilized. So the question arises if the model is discrete-time, thus closer to the platform it is implemented on, or continuous-time, and closer to the ground truth process it is supposed to model.

The distinction of the model time domain is not only about model precision accuracy. It affects also more than just algorithmic complexity. The model time domain, e.g., discrete-time or continuous-time, highly influences attributes such as coping with the integration of expert knowledge, irregularly sampled data, memory efficiency, long-term forecast accuracy, and data normalization.

Yet, continuous-time NN faces issues to be addressed. Long-term forecast accuracy is troublesome. Forecasting beyond the training horizon, which is always limited to available computational resources, requires extrapolation in time. In this case, data-based models can significantly lose prediction accuracy over the forecast horizon.

Furthermore, given discrete-time data, a normalization of the data in the value domain is sufficient. Given continuous-time NN, the output is the state derivative and is therefore directly linked to the time domain. Literature does not utilize a normalization in value and time domain, which results in untapped potential. Only the importance of data normalization in the value domain to improve numerical properties is well known in the literature.

In the case of a few training data, continuous-time models are, by default, more efficient than the discrete-time pendant. This routes back to the time continuity prior to continuous-time models. Defining a prediction as a change rate rather than estimating in-

dependent states reflects the restricted stiffness of gradients in many real world dynamical processes.

Finally, the literature lacks a recent survey of continuous-time NN for system identification. Early progress with continuous-time NN, such as [WL98], was not vigorously continued. In 2018, [Ch18] gained much attention, introducing Neural Ordinary Differential Equations (NODE). A general survey on NODE and its variants has been presented in the Ph.D. thesis [Ki22]. Yet, the literature lacks a survey applying continuous-time NN in system identification settings for industrial robot control.

1.2 Objectives

The objective of this thesis is to improve the trajectory-tracking accuracy of an industrial robot. Therefore, a model-based and sensor-based approach are combined. The feedback control is upgraded by integration of link side sensors, Secondary Encoders (SE), which explicitly measure gearbox deformations. The feed-forward control is improved by two methods, a physical and a data-based model. An additional hybrid model is derived to combine both. All three models are compared and evaluated regarding path accuracy, implementation effort, and safety. The key contributions made to the physical model address the shortcomings discussed before.

- **Differentiable flexible joints.** A continuously differentiable formulation of the flexible joint model is developed. This enables the use of downstream Automatic Differentiation (AD) tools to solve the symbolic differential equations and to decrease computational demands.
- **Hydraulic counter weight.** A nonlinear HWC model is derived to increase the accuracy of the gravitational loads acting on the second joint. It is based on hydraulic and geometric modeling.
- **Parameter identification.** A parameter identification procedure is designed as two global, nonlinear optimization problems: One is to determine the ideal design of experiments and obtain reference trajectories that optimally excite the robot friction parameters. Thereafter, using a measurement based on these trajectories, the friction parameters are identified in a second optimization problem.
- **Flatness-based feed-forward control.** A flatness-based feed-forward control is derived, which incorporates the nonlinear robot model without approximations. Due to the continuous differentiability, symbol code generation tools are applied to generate low-level, performance-optimized C++ code, without external software library dependencies.

Regarding the data-based model, the core contributions are stated in the following. To the best of the authors' knowledge, continuous-time neural networks have not been applied to industrial robot feed-forward control yet. One work considers a comparison of machine learning approaches, including a continuous-time NN, for system identification of a 3 Degrees of Freedom (DoF) anthropomorphic robotic manipulator [EK99]. However, [EK99] does not consider a 6 DoF robot, an industrial robot, or the application of NN for feed-forward control. Many works in the literature address robot control with discrete-time neural networks [Ji18; MAS03]. However, as discussed before, discrete-time neural networks account for severe drawbacks regarding memory efficiency, handling of irregularly sampled data, time-domain normalization, long-term forecast accuracy, and the integration of expert knowledge. This thesis contributes to addressing the three last-mentioned shortcomings.

- **State Derivative Normalization.** To improve the numerical properties of continuous-time NN, a state derivative normalization method is presented. This transforms data collected in a sample time suited for the real-time system, to a time grid suitable for the model. It does not affect model execution time. It deploys and evaluates the NN on a virtual time grid rather than applying the measurement time grid directly.
- **Stability properties and long-term forecast accuracy.** A benefit of the similarity between continuous-time models and the well-understood system theory is the applicability of stability theorems. Based on a Lyapunov approach, combined with a constraint optimization problem, model stability for long-term forecast horizons can be analytically guaranteed. This holds for noisy measurements and any forecast horizon length and can already be enforced during model training.
- **Explainability and expert knowledge.** As a result of the resemblance between continuous-time models and the ground-truth dynamics, integration of expert knowledge is enabled on a parameter level of the data-based models. This reinforces hybrid models, which combine expert knowledge and data-based models. Besides increased explainability, hybrid models can explicitly contribute to model safety.

1.3 Approach

The following chapter 2 reflects state-of-the-art concerning physical robot modeling. First, a literature overview with a focus on robot machining, elastic joint models, parameter identification, and robot control is given in section 2.1. This embeds the contributions made in the following chapters in the literature. Second, in section 2.2, a state-of-the-art nonlinear rigid joint model is presented. This serves as a baseline and starting point for

the improved robot model and control in chapter 4.

Chapter 3 reflects the state-of-the-art of system identification with neural networks. Similar to the previous literature chapter, it starts with an overview in section 3.1, which focuses on discrete-time and continuous-time approaches. Then, in section 3.2, a mathematical introduction to continuous-time NN is presented. Special emphasis is given regarding the Ordinary Differential Equations (ODE) configuration in section 3.3, the NN model architecture in section 3.4, and the training configuration in section 3.5. The most important aspects of continuous-time NN for system identification are defined, and the possibilities regarding each aspect are presented. The advantages and disadvantages of design choices are discussed, and recommendations are derived. This part aims to guide new researchers and developers into the field of system identification using continuous-time NN.

Chapter 4 defines the advanced physical nonlinear robot model. First, several components such as a continuously differentiable flexible joints model, a HWC model, and an advanced friction model are developed in section 4.1. Second, using an optimization-based design of experiments, the friction model parameters are fitted to robot measurements in section 4.2. Finally, in section 4.3, a flatness-based feed-forward control structure is derived, which computes the motor torque using the complete nonlinear model. The feed-forward control complements a feedback control utilizing SE, which can explicitly measure joint deformations.

Chapter 5 addresses data-based models. Section 5.1 upgrades continuous-time NN with state derivative normalization. State derivative normalization defines methods to normalize the hidden state and hidden state derivative. It improves the numerical properties of continuous-time models. Section 5.2 derives stability constraints to guarantee model stability for long forecast horizons. The method exploits the similarity between continuous-time NN and Lyapunov stability and combines this with constraint optimization. Section 5.3 demonstrates the superior generalization capabilities for robot identification given few training data of continuous-time versus discrete-time NN. Both models, continuous-time, and discrete-time, are directly comparable as the data processing, the training pipeline, and the model architecture are almost identical. The only difference is the embedding of the discrete-time architecture in an Eulerscheme to obtain the continuous-time model.

In chapter 6, the physical, data-based, and hybrid models are applied to the industrial robot. First, the design of experiments is explained in section 6.1. Then, configurations for all three model types are derived in section 6.2. Finally, in section 6.3, all models are compared on directly comparable experiments. The models are evaluated regarding model accuracy, robot path accuracy, and model safety.

A summary and concluding remarks are given in chapter 7.

2 Foundations of Physical Robot Modeling and Control

This chapter defines first-principle physical models for industrial six-joint robots. A literature review on robot machining, flexible joint modeling, parameter identification, and robot control is given in section 2.1. In section 2.2, the mathematical baseline of a state-of-the-art nonlinear robot model is presented. Section 2.2 serves as starting point for the advanced modeling presented in chapter 4 of this thesis.

2.1 State-of-the-Art Literature Robotic Modelling and Control

2.1.1 Robot Machining

Industrial robots are highly flexible due to their open and complex kinematic chain. However, they are mainly designed for good repeatability but not for high-precision tasks, such as milling [Ol12] or robot machining (see Fig. 2.1). The main drawback of using an industrial robot in machining processes is the lack of position accuracy caused by low stiffness, meaning low Eigenfrequencies and joint elasticity between the actuators and the driven links [Fr14; Sc14]. As a result, conventional CNC machines, possessing high stiffness and simple kinematics, can still not be replaced by industrial robots for machining tasks [ISA15; WKK18]. An analysis of the sources of errors leading to low path accuracy is presented in [Sc16a]. Furthermore, the low and highly pose-dependent overall stiffness of industrial robots, up to 100 times less than conventional CNC machines, can lead to vibrations and chattering effects so that the machining quality suffers [Br18; VKL17; Yu18].

In addition, the influence of the placement of a workpiece concerning the robot can lead to different machining results due to the pose-dependent stiffness of industrial robots [LZD17]. Researchers consider improving the structure of the robot, which leads to a hardware and therefore cost-intensive approach [Ve15]. Two different methods to increase the machining accuracy of industrial robots are reported in the literature, either to use a model-based approach accounting for compliance with flexible joints or to use a sensor-based approach

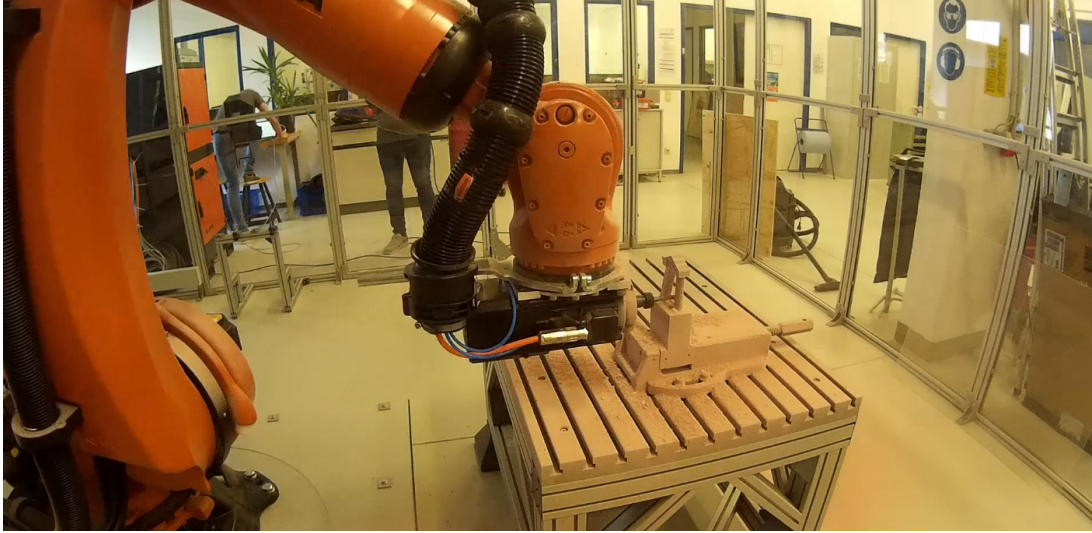


Figure 2.1: Exemplary picture of robot machining.

by tracking deformations [Fr17; Ol12; Sc14; WZF09]. The sensor-based approach can lead to a higher position accuracy than a model-based approach [Sc14].

Attaching SE to the link side can reduce the oscillatory behavior and chattering effects by providing sensor information to a feedback controller. An improvement of static precision by a factor of 10 was achieved in [De11c] using SE. Drive-based damping utilizing SE was developed in [VKL17]. However, the approach is associated with high costs, high implementation effort, communication delays, and sensor noise [Sc14]. However, reducing dynamic deflections due to time-varying dynamical effects remains a challenge that a model-based approach can only handle. By applying the model-based approach, nonlinear effects, such as time-variant stiffness, lost motion, backlash, and friction, can be accounted for. The model can be applied to design an appropriate model-based control law for the high-precision operation of the industrial robot.

2.1.2 Nonlinear Flexible Joint Model

Modeling joint flexibility involves considering stiffness and friction between the motor and the driven link and the effects occurring in transmission drives. High-precision transmission drives, such as cycloidal gears and harmonic drives, are typically used for industrial robots to meet high-precision requirements. Simplified models considering linear stiffness and damping between the motor and the driven link are used in [AOH07; DL98; Me20; Sp87; WL92]. An overview of further simplified models can be found in [DB16].

Nonlinear modeling approaches account for torsional compliance by considering hysteresis effects [CH17; Hu20; K114; RHB09]. Hysteresis behavior results from the structural damping of transmission elements and their piecewise elastoplastic properties [RHB09]. It was shown that hysteresis effects could significantly contribute to a low path accuracy

[CH17]. The authors point out that it primarily impacts the positional error of the base joint due to the large lever arm. Moreover, hysteresis behavior, which has a bidirectional behavior, leads to alternating stresses in the gears [Br16]. The authors develop a numerical method for backlash identification with the help of experimental data from laser trackers and recorded control data. The nonlinear characteristics of lost motion and backlash in transmission drive considerably affect lower path accuracy in machining tasks. Lost motion is defined as the torsion angle at the midpoint of the hysteresis curve, where not all tooth flanks of the gearbox are in full contact. In contrast, the backlash is defined as the angle difference in the output shaft at zero output torque, where gear teeth are not in contact [HEV18; TPA16]. The authors in [HEV18] show a considerable improvement in the milling accuracy of aluminum by taking into account backlash effects and all joint elasticities.

Moreover, it was shown in [Ya16] that backlash leads to the accumulation of positioning errors while joints change their rotation direction, leading to torque oscillations and, thus, earlier gear system failure. However, the literature on modeling lost motion effects is scarce. The impact of the backlash caused by transmission drives was recognized by [KG97] in his experimental study of harmonic drive, showing that torque transmission has a nonlinear characteristic and the input torque cannot be entirely transmitted to the driven link. The impact of backlash is also investigated and modeled in [Fr14; YYH15].

Another extensive work concerning nonlinear dynamical models and identification of elastic robot joints with hysteresis and backlash effects is addressed in [RHB09]. Elasticities of individual components have often been modeled using the Virtual Joint Method (VJM), where the components are modeled by virtual springs located in joints [PKC11]. Additionally, it is essential to account for frictional torque as it can suddenly change direction and the hysteresis can significantly contribute to very low accuracy [CH17]. One of the recent studies modeling backlash for a planetary gear transmission and highlighting the problem mentioned above is [YYH15].

2.1.3 Robot Parameter Identification

Robot identification is a topic that has been intensively investigated in the last four decades [WWY10]. Moreover, recent years have witnessed a renewal of interest in this problem due to a rapid increase in robotic hardware platforms capable of accurate model-based control, e.g., [JW21; WKS18]. Conventional approaches to the identification problem make use of the linearity of the inverse dynamics model for the unknown parameters, and this allows identification to be formulated as a least-squares problem, see [GDH01; GJV13; JW21]. However, although this offline identification method has been validated on several robots and benchmarks, it will no longer be applicable when friction exhibits nonlinear

characteristics. Furthermore, parameter-linear models lead to complicated physical feasibility guarantees, which require nonlinear constraints, if possible. For example, enforcing a positive definite inertia matrix usually requires nonlinear constraints, even if the model is linear concerning these parameters [BTC18].

Identification of friction models has gained a lot of attention [HGG15; JL92; KG13]. The complex friction modeling includes viscous, Coulomb, and Stribeck effects, analytical models, and load dependency. Appropriate identification procedures have been presented [BIS06; KPM07]. Nevertheless, only a few works consider online identification, such as in [Lä09]. As mentioned earlier, all other works consider dedicated, offline identification procedures. Gautier [GK92] even argues that a dedicated, high energy-exciting trajectory is crucial for a good identification process, which is not realizable in an online identification setting. Although an offline identification procedure can capture time-varying effects by modifying the dynamic model, online identification methods appear more convenient in accounting for effects such as temperature-dependent friction coefficients. A comparison of methods dedicated to payload identification is presented in [KGL07].

2.1.4 Nonlinear Robot Control

Enhancing trajectory tracking accuracy requires considering the most relevant dynamic effects and designing a proper control law [MH08]. Chattering effects, which significantly influence the surface quality in a milling process, can only be eliminated using a flexible model-based controller [KC19]. For the flexible model-based controller, the trajectory should be continuously differentiable up to the 4th-order, e.g., up to the jerk derivative. In contrast, a 2nd-order trajectory is sufficient for rigid model-based controllers. A continuously differentiable trajectory can be computed, e.g., based on the dynamic model or estimated by numerical differentiation, which is error-prone due to high sampling frequencies, measurement noise, or model uncertainties. In [OI12], a two-phase model-based approach is applied to compensate for compliance and gearboxes kinematic errors. A local identification was used to identify the joint's stiffness, and a subsequent offline correction method was used to improve the accuracy.

Spong [Sp87] was the first to show the global feedback linearization of the flexible joint system, where all nonlinearities are in the control input range, leading to robust control law. Many other control approaches emerged since then, such as controllers with gravity compensation, energy shaping, Proportional Integral Derivative (PID) regulators with semi-global stability properties, singular perturbation, backstepping, passivity, two-time scale separation, and sliding mode techniques, which are summarized in [DB16].

The authors in [YP18] propose a robust adaptive control method for trajectory tracking and online parameter estimation of a 6 DoF industrial robot. The proposed controller

differs from other controllers in the literature. It is designed in the robot's end effector's task space to achieve better trajectory tracking accuracy than controllers designed in joint space. The method significantly reduces trajectory tracking error compared to a conventional Proportional Derivative (PD) controller. Similarly, the authors in [Me20] show an improvement in the dynamic path accuracy of a robot manipulator in a machining process, proposing a controller built on an independent joint control. A damping control algorithm is designed and validated experimentally, based on velocity feedback using SE. A state estimation extends the approach. With the proposed controller, it is possible to systematically alter the stiffness and damping of the control system with the help of two proportional gains. The authors in [Zh20] show that nonlinear modeling effects, such as friction, and subsequent derivation of a model-based controller, can significantly improve an industrial robot's position and velocity accuracy.

2.2 Nonlinear Robot Model

2.2.1 Kinematic Model

This work considers a six-joint industrial robot. In particular, the demonstrator is a KUKA KR300 R2500 ultra SE robot as presented in Fig. 4.12. The complete thesis refers to this type of robot. Its kinematics is depicted in Fig. 2.2. The model is based on generalized coordinates $q_i \in \mathbb{R}^{N_Q}$, where N_Q stands for the number of DoF, with $N_Q = 6$ for the demonstrator. To simplify notion, a vector of coordinates $q = [q_1, q_2, \dots, q_{N_Q}]$ is introduced.

On the six-joint manipulator, SE are mounted on the base, shoulder, and elbow joints, e.g., joints 1, 2, and 3. The motor encoders of the first three joints are used for velocity control only. Joints 4, 5, and 6 are not equipped with SE and apply motor encoders for position and velocity control.

2.2.2 Rigid Joint Model

Details on the robot and control are given in previous works [WGR20]. The well-known rigid body robot model

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) + \tau_F(\dot{q}) + \tau_H(q) = U\tau_M(q, \dot{q}) \quad (2.1)$$

consists of the pose-dependent inertia matrix $M(q) \in \mathbb{R}^{N_Q \times N_Q}$, the centrifugal and Coriolis matrix $C(q, \dot{q}) \in \mathbb{R}^{N_Q \times N_Q}$, the vector of gravitational torques $g(q) \in \mathbb{R}^{N_Q}$, the nonlinear

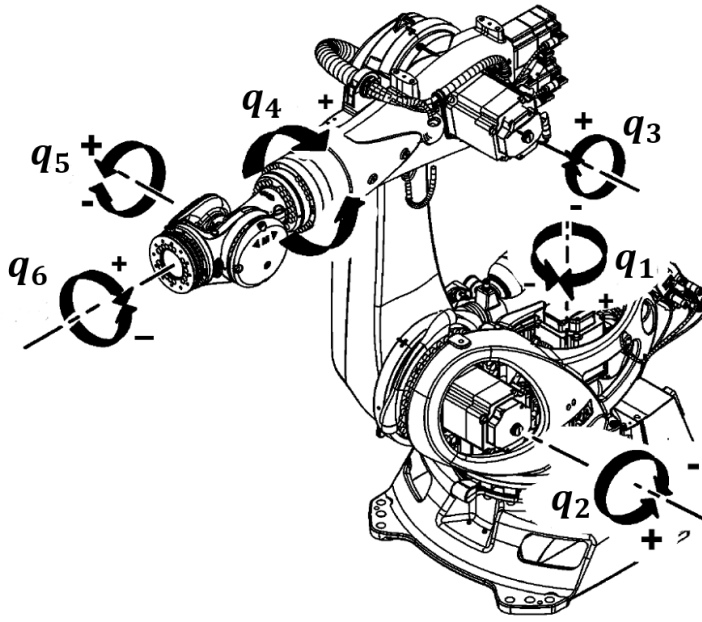


Figure 2.2: Kinematics of the KUKA KR300 R2500 ultra SE robot (Figure based on [KU21]).

friction torque vector $\tau_F(\dot{q}) \in \mathbb{R}^{N_Q}$, the torque vector resulting from the Hydraulic Weight Counterbalance (HWC), $\tau_H(q) \in \mathbb{R}^{N_Q}$, the diagonal gearbox transmission matrix $U \in \mathbb{R}^{N_Q \times N_Q}$, $U = \text{diag}(u_1, u_2, \dots, u_{N_Q})$ and the input torque transmitted from the electrical motors $\tau_M(q, \dot{q}) \in \mathbb{R}^{N_Q}$. The HWC is introduced in section 4.1.4. It is not part of state-of-the-art models. However, subsequent chapters utilize the rigid body robot model including the HWC. For the state-of-the-art model, it is set to zero.

2.2.3 Rigid Joint Feed-Forward Controller

The rigid body inverse robot model required for control is

$$\tau_{FF} = U^{-1} (M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) + \tau_F(\dot{q}) + \tau_H(q)). \quad (2.2)$$

Equation (2.2) is a constant mapping in which the output τ_{FF} does not rely on previous motor torques. Considering the actual control architecture, the motor torque is not completely discontinuous. Nevertheless, note that the motor torque is not a time function.

2.2.4 Friction Model

A nonlinear friction model accounting for viscous and Coulomb friction is applied

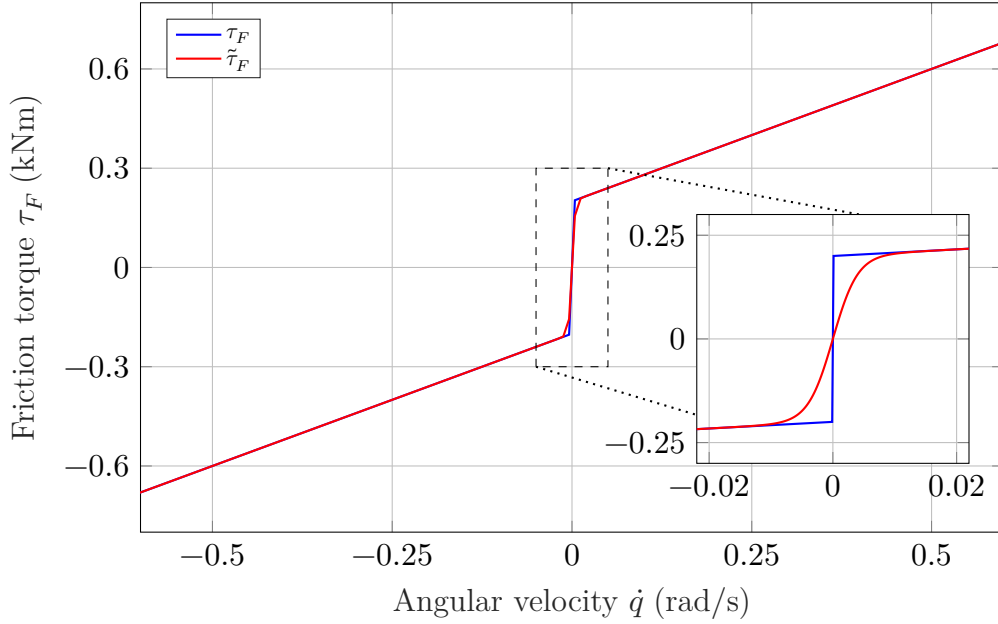


Figure 2.3: Coulomb and viscous friction model with discontinuous and continuous approximation, exemplary for joint 1

$$\tau_F(\dot{q}) = f_v \dot{q} + f_c \tanh(s_f \dot{q}), \quad (2.3)$$

with the viscous friction coefficient $f_v \in \mathbb{R}^{N_q}$, the Coulomb friction coefficient $f_c \in \mathbb{R}^{N_q}$, the sign smoothness factor $s_f \in \mathbb{R}^{N_q}$. The smoothing is required for the derivation of the flatness-based control. The function (2.3) for different values of Coulomb friction steepness s_f is presented in Fig. 2.3.

2.2.5 Inertia, Coriolis, Centripetal, and Gravitational Model

The inertia, Coriolis, centripetal matrix, and gravitational load depend on the kinematic parameters and the mass and inertia of each robot link. The kinematic parameters are given in the manual [KU21]. Mass and inertia parameters are derived from a Computer Aided Design (CAD) model of the robot [Ha18a]. The solid body CAD model has transformed into a hollow body CAD model, material parameters were added, and masses, inertia as well as centers of gravity were calculated using Finite Element Methods (FEM).

Using the *MATLAB robotics toolbox* by Peter Corke [Co17], the pose- and velocity-dependent estimation of the inertia, Coriolis, and centripetal, as well as the gravitational loads, were possible. Moreover, this implementation was computationally sufficiently efficient for trajectory generation, robot simulation, and feed-forward control.

3 Foundations of Continuous-Time Neural Networks

This chapter defines the foundation of system identification using neural networks. This idea dates back several decades [FN93; WL98; WZ89]. It has gained much attention since the publication of NODE [Ch18]. For a brief history, see [Ki22]. The notation standard in the system identification community rather than the Continuous Normalizing Flows (CNF) community is utilized in this thesis.

Section 3.1 gives an overview of system identification utilizing NN with a focus on discrete-time and continuous-time methods. Starting in section 3.2, formal insight of continuous-time NN is presented. Section 3.2 until section 3.5 are written for readers new to the field, giving an overview of important configurations and presenting corresponding recommendations.

3.1 State-of-the-Art Literature Neural Networks

3.1.1 System Identification with Neural Networks

Nonlinear system identification is a core discipline in control engineering (e.g., [Ne01; Sj95]). For more than 30 years, NNs have been developed and widely applied for identifying nonlinear dynamic systems (e.g., [Ah10; CBG90; DK11; Ha90; NP90; Og16; Sc22; SL19; WDR21; WZ89]). Models can be divided into discrete-time models, such as Recurrent Neural Network (RNN) [TB97] or Long Short Term Memory (LSTM) [HS97; Sh20] and continuous-time models such as NODE [Ch18], Runge-Kutta Neural Networks (RKNN) [WL98], Deep Encoder Networks [BST22] or Liquid Time Constant (LTC) [Ha18b].

Many contributions focus on discrete-time NNs. As pointed out by [WL98] there are several problems in using discrete-time NNs. The first-order discretization induces additional approximation errors. The long-term prediction accuracy is often amendable, and the NN can only predict the system behavior at fixed time intervals [WL98]. Furthermore, a variable time step in training and application leads to complete model failure, opening the requirement for data preprocessing. This is mainly because feed-forward networks and RNN tend to learn the system states instead of the change rates of system states. The

discrete sample time is inseparably connected to the NN weights [WDR21].

Another approach is continuous-time NNs. This type of network does not suffer from the disadvantages mentioned above. However, continuous-time NNs do not consider the discrete-time nature of measurements or any digital controller. They often require a quasi-continuous sample time. This results in a large amount of training data and a restriction of applications. In industrial real-time applications, a high sample time can be prohibitive; thus, the discrete-time nature of the measurements has to be taken into account. RKNN bridges this gap by constructing a NN in the continuous-time domain and explicitly integrating the Runge-Kutta (RK) approximation method. Thus, a precise estimate of the change rates of the system states is possible [WL98]. Therefore, RKNN does not suffer from the disadvantages mentioned earlier of discrete-time or continuous-time NNs and is superior in generalization and long-term prediction accuracy, as theoretically proven by [WL98] and shown by simulations by [EK00]. More recently, [BTS21a; FP21; MFP20; WDR21] achieved excellent results with continuous-time NNs for system identification.

Model stability is crucial for nonlinear system identification, especially with data-based NNs. Accordingly, the stability properties of discrete-time NNs are widely studied (e.g., [BP02; HW02; WX06; Yu04]). The continuous-time model stability properties are broadly analyzed too, see, e.g., [HW02] for stability, and [SP99] for the more general concept of Input to State Stability (ISS). ISS extends the stability criterion of Global Asymptotic Stability (GAS) in the sense of Lyapunov to non-autonomous systems, e.g., systems with an input unequal to zero. [So08] shows that some GAS stable systems can diverge even for some inputs that converge to zero.

3.1.2 Discrete-Time Neural Networks

In the 90s, the idea of combining tapped delay lines with NN arose. A discrete input signal is delayed several time steps using a zero-order-hold memory. The current and past values are combined into the NN. It is also called Nonlinear Autoregressive with Exogenous Input (NARX) network and depicted in Fig. 3.1.

A drawback of RNN is the vanishing or exploding gradient problem when trained with Backpropagation Through Time (BPTT) [We90]. A significant improvement regarding the vanishing or exploding gradient issue for discrete-time NN has been introduced with LSTM [HS97] networks in 1997. The main idea of LSTM is to make the decay rate of the internal memory a trainable network parameter. So the optimizer can adjust the memory depending on the requirements of the data set. Fig. 3.2 presents the scheme of a LSTM cell.

The scientific popularity of discrete-time versus continuous-time NN can be illustrated by

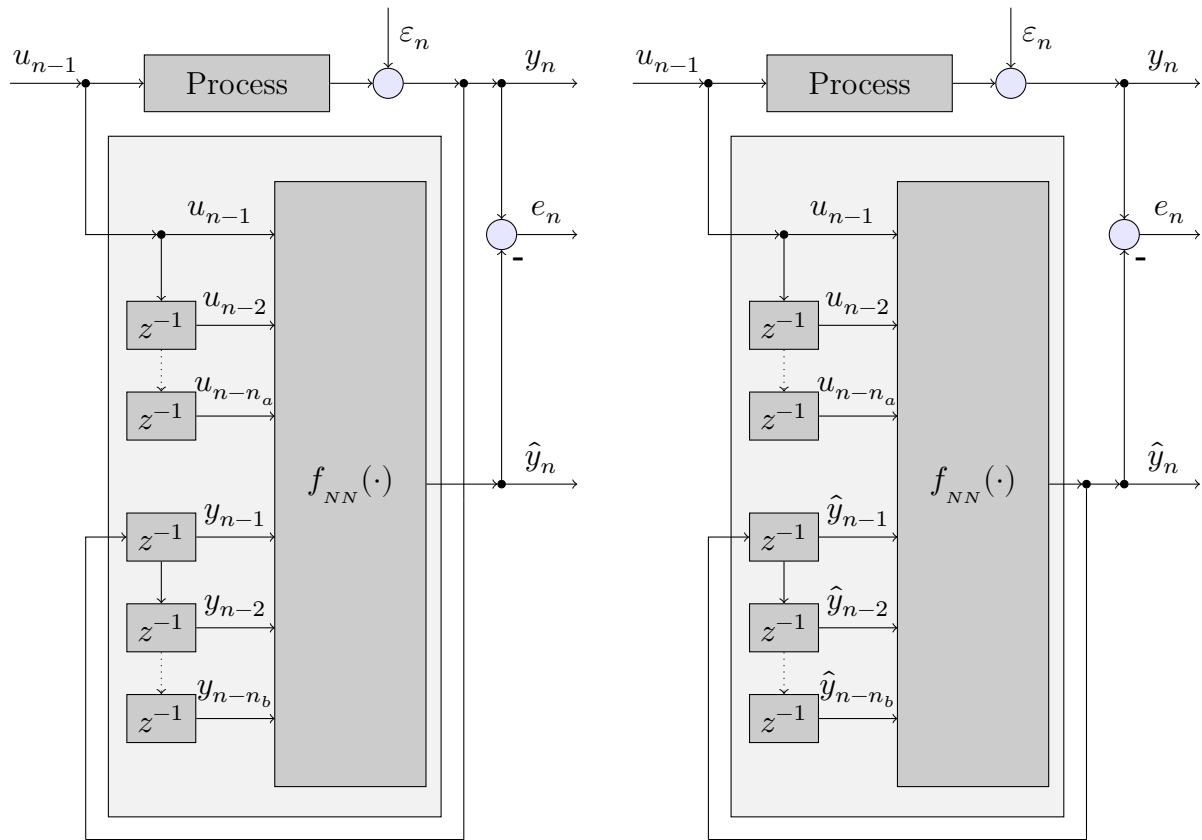


Figure 3.1: NARX neural network architecture in the predictor (left) and simulation (right) mode. [De10; XJ02]

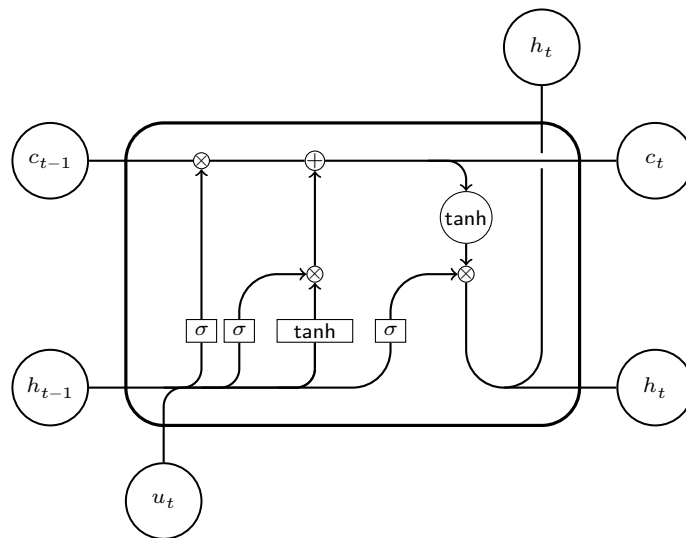


Figure 3.2: Long Short Term Memory Unit [HS97].

the number of citations of selected, most influential publications. The LSTM work by Hochreiter and Schmidhuber can still be considered state-of-the-art, with over 73,400 citations. Gated Recurrent Units (GRU) [Ch14] developed in 2014 a similar idea, with different mechanisms for adjusting the hidden state’s decay rate. The GRU publication achieved 10,800 citations. In contrast, one of the most cited continuous-time NN [WL98] from 1998 achieved just over 100 citations in the same decades as the LSTM paper. Furthermore, LSTM and GRU are implemented in both major deep learning libraries, *TensorFlow* [Ma15] and *PyTorch* [Pa19], continuous-time models are implemented by default in none of those. In the domain of identification of dynamic systems, good results have been achieved for example in [KF18].

3.2 General Form of Continuous-Time Neural Networks

The main idea of continuous-time NN in this work is to combine the universal function approximation capabilities of a NN with an explicit ODE solver. However, methods exist to directly apply the network in a continuous-time setting or a closed-form solution [Ha22b]. In both cases, the NN gets the state $x(t) \in \mathbb{R}^{N_x}$ and additional features $u(t) \in \mathbb{R}^{N_u}$ as input, and computes the derivative of the hidden state $\dot{x}(t) \in \mathbb{R}^{N_x}$ as output,

$$\dot{x}(t) = f_{\text{NN}}(x(t), u(t)). \quad (3.1)$$

Furthermore, an output network is defined

$$\hat{y}(t) = h_{\text{NN}}(x(t), u(t)), \quad (3.2)$$

which creates the predictions $\hat{y}(t) \in \mathbb{R}^{N_y}$ which correspond to the target values $y(t) \in \mathbb{R}^{N_y}$. For classification tasks, $\hat{y}(t)$ contains the output distribution with $N_y \in \mathbb{N}$ equal to the number of target classes. For regression tasks, the number of output channels N_y corresponds to the number of independent target variables to predict.

In some works, the exogenous input $u(t)$ is omitted, for example in ResNet or NODE [Ch18; He16]. ResNet, as well as CNF, depends on the initial state exclusively. In the system and control community terminology, neglecting the exogenous input corresponds to an autonomous system. In the terminology of the CNF community, networks with exogenous inputs are referred to as Neural Controlled Differential Equations (NCDE)

Depending on the setup, the observable output $y(t)$ can be fed back to the model. For example in a classification task, a sequence of Inertial Measurement Unit (IMU) measure-

ments of a human is given, and the task is to predict if the individual is sitting, walking, or running. Thus, the discrete target variable is constant over a time interval and not fed into the network. In image classification, too, the hidden state $x(t)$ is forwarded to a second network (3.2) with a different architecture to estimate $y(t)$. In time series forecasting, especially in a multistep-ahead setting, the forecast variable $\hat{y}(t)$ is a part of the hidden state $x(t)$ and is recursively fed back to the system.

The dimension of the hidden state $x(t)$ can be augmented. Augmented states refer to [DDT19], who argues that increasing the number of hidden state dimensions above the physically motivated level benefits the model’s flexibility. Indeed, augmented states reduce the information bottleneck in the state vector and improve model performance in the sense of accuracy, at least on the training data. On the other hand, model performance in the sense of interpretability is diminished (as non-physical states are incorporated), and model generalizability is likely reduced, given a limited amount of training data, as the number of model parameters is increased.

Depending on the application, initial hidden state estimation can be applied. Four standard options to estimate the initial state are given in the following.

1. The initial state is known by prior knowledge.
2. The initial state is set to zero.
3. The initial state is set to a combination of the initial output variable $y(0)$, the initial exogenous input $u(0)$, its derivatives $\dot{y}(0)$, and $\dot{u}(0)$, its higher order derivatives, and zeros.
4. The initial state is estimated using a Deep Encoder Network [BST22] using the information of a past horizon of input and output variables.

For many data sets, prior knowledge of the initial state is infeasible. However, if prior knowledge is available, if the model is based more on expert knowledge than on data, or if the hidden state is wholly known, then accessing this prior knowledge is usually the first choice.

Setting $x(0) = 0$ reduces model prediction accuracy, as some information at the beginning of each sequence is missing. Nevertheless, the method enables good generalization capabilities, as the learned model does not require accounting for different initial states.

A continuous-time data-driven approach uses the output variable, its derivative, the input variable, and zeros for all remaining state dimensions. Given a sufficiently high order of derivatives of $y(0)$, no information loss occurs in the initial state estimation for flat systems. However, considering noisy measurements, non-flat systems, and a limited state dimension, the theoretical requirements are cumbersome to satisfy. This method is recommended when no prior knowledge is available, additional model parameters are not intended, and

loss of information and, in consequence, loss of prediction accuracy is not intended.

The final option, Deep Encoder Networks [BST22] is a very flexible approach without losing information. It overcomes the difficulties mentioned above. However, Deep Encoder Networks introduce an additional model, which increases computational effort and memory load and requires sufficiently rich data to be trained. The Deep Encoder Network is defined as

$$x_{n_0} = e_{NN}(y_{n_0}, y_{n_0-1}, \dots, y_{n_0-n_a}, u_{n_0}, u_{n_0-1}, \dots, u_{n_0-n_b}) \quad (3.3)$$

with an initial state at time step $n_0 \in \mathbb{N}$, and past horizons $n_a \in \mathbb{N}$, $n_b \in \mathbb{N}$ of input and output data respectively. The initial step is $n_0 = \max(n_a, n_b)$.

A further optional enhancement is a Differential Algebraic Equations (DAE) network,

$$0 = g_{NN}(x(t), u(t)), \quad (3.4)$$

which enables the model to cover systems in the form of DAEs. In the proposed implementation, the DAE is exclusively trained in the loss function and not applied during inference. For the loss function, a penalty method is introduced

$$L_g = \lambda_g \frac{1}{N} \sum_{n=0}^{N-1} g_{NN}(x_n, u_n)^2 \quad (3.5)$$

with a scaling parameter $\lambda_g \in \mathbb{R}^+$. Equation (3.5) is the Mean Squared Error (MSE) over the complete simulation horizon. As a consequence, the loss function ensures that $0 = g_{NN}(x(t), u(t))$ holds for every time step of the horizon.

3.3 ODE Configuration

3.3.1 Explicit Fixed-step Runge-Kutta Solvers

The continuous-time neural network (3.1) is embedded in an S -stage explicit RK method with sample time h to generate the discrete-time system

$$x_{n+1} = F_{\text{RKNN}}(x_n, u_n) \quad (3.6a)$$

$$\hat{y}_n = H_{\text{RKNN}}(x_n, u_n). \quad (3.6b)$$

The discrete sample time $n \in \mathbb{N}$ is defined so that $x_n = x(t = nh)$. The RK scheme provides the rule to compute (3.6) from (3.1) for a given sample time h . The general explicit S -stage RK method is defined by the discrete-time system

$$x_{n+1} = x_n + h \sum_{s=1}^S b_{B,s} z_{n,s} \quad (3.7a)$$

$$z_{n,s} = f_{\text{NN}} \left(x_n + h \sum_{j=1}^S a_{B,s,j} z_{n,j}, u(nh + c_{B,s}h) \right) \quad (3.7b)$$

with the Butcher coefficients $a_{B,s,j}$, $b_{B,s}$, $c_{B,s}$ and $\sum_{s=1}^S b_{B,s} = 1$, $c_{B,s} = \sum_{j=1}^S a_{B,s,j}$ [Bu16]. A common representation is the Butcher array

$$\frac{c_B \mid A_B}{\mid b_B^T} = \begin{array}{c|cccc} c_{B,1} & a_{B,1,1} & & & \\ c_{B,2} & a_{B,2,1} & a_{B,2,2} & & \\ \vdots & \vdots & & \ddots & \\ c_{B,S} & a_{B,S,1} & a_{B,S,2} & \dots & a_{B,S,S} \\ \hline & b_{B,1} & b_{B,2} & \dots & b_{B,S} \end{array}$$

The RK scheme (3.7) is completely determined by the parameter matrix and vectors A_B , b_B and c_B . Note that for an explicit RK method, the matrix A_B is lower triangular [Bu16]. Although the notation (3.7) allows a varying u within the sample interval, it is assumed to be constant during one arbitrarily short sample interval.

In the following, the state network is defined as

$$z_{n,s} = f_{\text{NN}} \left(x_n + h \sum_{j=1}^S a_{B,s,j} z_{n,j}, u_n \right). \quad (3.8a)$$

The solution of the output network is straightforward since it does not contain any derivatives or any feedback,

$$y_n = H_{\text{RKNN}}(x_n, u_n) = f_{\text{NN}}(x_n, u_n). \quad (3.9)$$

3.3.2 Euler Neural Networks

Euler Neural Networks are named by the Euler-Forward scheme as ODE solver. It is given by the Butcher tableau [Bu16]

$$\frac{0 \mid}{\mid 1}.$$

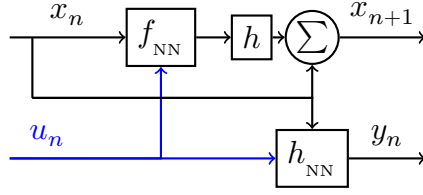


Figure 3.3: Representation of the Euler Neural Network architecture. The Euler Neural Network explicitly includes the integration method as a neural network graph extension.

The Butcher tableau results in the corresponding equations

$$z_{n,1} = f_{\text{NN}}(x_n, u_n) \quad (3.10a)$$

$$x_{n+1} = x_n + h z_{n,1} \quad (3.10b)$$

$$\hat{y}_n = h_{\text{NN}}(x_n, u_n). \quad (3.10c)$$

The Euler-Forward is the simplest ODE solver scheme. However, mainly due to its simplicity, it gains some advantages, such as low computational requirements. Therefore, it is the default choice for LTC models [Ha21]. The Euler-Forward model is depicted in Fig. 3.3.

3.3.3 Runge-Kutta Neural Networks

The RKNN applies the 4-stage Runge-Kutta algorithm as ODE solver. Its Butcher tableau and the corresponding equations are

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	2/6	2/6	1/6

$$z_{n,1} = f_{\text{NN}}(x_n, u_n) \quad (3.11a)$$

$$z_{n,2} = f_{\text{NN}}(x_n + h/2 z_{n,1}, u_n) \quad (3.11b)$$

$$z_{n,3} = f_{\text{NN}}(x_n + h/2 z_{n,2}, u_n) \quad (3.11c)$$

$$z_{n,4} = f_{\text{NN}}(x_n + h z_{n,3}, u_n) \quad (3.11d)$$

$$x_{n+1} = x_n + h/6 (z_{n,1} + 2z_{n,2} + 2z_{n,3} + z_{n,4}) \quad (3.11e)$$

$$\hat{y}_n = h_{\text{NN}}(x_n, u_n). \quad (3.11f)$$

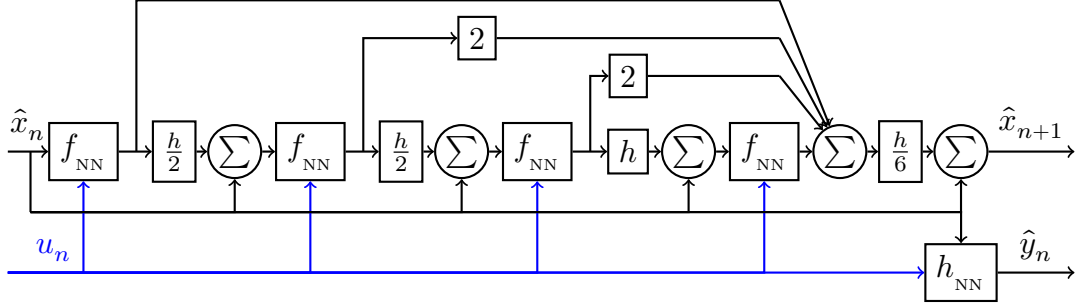


Figure 3.4: Representation of the RKNN architecture. The RKNN explicitly includes the integration method as a neural network graph extension.

The RKNN architecture as NN graph is depicted in Fig. 3.4. Already decades ago, researchers highlighted the superior prediction accuracy of RKNN [Ah10; WL98].

3.3.4 Neural Ordinary Differential Equations

NODE are developed as the continuous-time limit of the ResNet architecture [Ch18; OR20]. NODE propose to apply any black-box ODE solver. Importantly, both fixed-step and adaptive-step solvers are applied. The implementation of [Ch18] contains the following list of ODE solvers.

- Adaptive-step
 - Dormand-Prince-Shampine of order 8
 - Dormand-Prince-Shampine of order 5 (recommended as default)
 - Bogacki-Shampine of order 3
 - Runge-Kutta-Fehlberg of order 2
 - Adaptive Heun of order 2
- Fixed-step
 - Euler method of order 1
 - Midpoint method of order 2
 - Runge-Kutta with 3/8 rule of order 4
 - Explicit Adams-Bashforth
 - Implicit Adams-Bashforth-Moulton

[Ch18] argues that the choice of the solver can explicitly trade computational resources for ODE solver accuracy. Especially using adaptive-step algorithms, numerical error-control methods can be applied, and the error can be qualified explicitly.

However, it is important to note that only ODE solver accuracy can be directly traded using adaptive-step or high-order methods. In the forecast and classification tasks, the overall accuracy depends on both the ODE solver accuracy and the model performance.

The model’s performance depends on its architecture, training configuration, and data set. However, as the loss function always accounts for the overall accuracy, the neural network model can partially compensate for low-accuracy ODE solvers. In the latter case, utilizing the same ODE solvers in training and inference is vital. Remarkable results using a fixed-step Euler method have been realized with ResNet [He16], LTC networks [Ha20] and Truncated Simulated Error Minimization (TSEM) [FP21].

Fixed-step solvers, compared to adaptive-step solvers, obtain the advantage of being fully deterministic. This results from the fact that failed iteration steps are impossible with fixed-step solvers. Determinism is essential regarding real-time applications when the algorithm’s run-time has a limited time budget. It is also crucial regarding the optimization process. Adaptive-step solutions potentially lead to discontinuity in the Jacobian of the solver. These discontinuities can be reduced by setting appropriate error-control methods. Nevertheless, discontinuity caused by non-uniform time grids does not occur for fixed-step solvers in the first place. For these two reasons, a fixed-step solver is chosen in this thesis.

Besides its ODE solver, [Ch18] argues for applying an adjoint method instead of BPTT. This discussion is postponed to section 3.5.6.

3.4 Neural Network Configuration

This thesis focuses on a specific network configuration, the nonlinear state-space neural network. This configuration consists of a linear state-space core and an additive nonlinear term. It is a subclass of the general form (3.1) and (3.2) and described by

$$\frac{dx(t)}{dt} = A_{NN} x(t) + B_{NN} u(t) + \tilde{f}_{NN}(x(t), u(t)) \quad (3.12a)$$

$$\hat{y}(t) = C_{NN} x(t) + D_{NN} u(t) + \tilde{h}_{NN}(x(t), u(t)). \quad (3.12b)$$

The linear core easily incorporates physical knowledge, even on a network parameter level. The configuration links to the stability theory, which is essential in section 5.2. Due to the similarity, it can be initialized with a Best Linear Approximation (BLA) (see for the discrete-time state-space NN [ST20]). BLA is a linear model which performs best in the sense of least-squares prediction error over all linearization points. It can be obtained using analytical solutions such as the subspace identification method with N4SID weight initialization [VD94].

From theory, one hidden layer is sufficient for universal function approximation capabilities, given bounded activation functions and sufficiently many hidden units [Ho91].

However, several works indicate that deeper networks increase the approximation capabilities of the network [BWS15; LS17]. In addition, [BDS21] showed that a minimal layer width is required to ensure that the decision regions are unbounded.

For continuous-time neural networks in a Discretize-then-Optimize (Disc-Opt) (see section 3.5.6), the setting, the forward evaluation, and the backpropagation of the network need to account for all layers multiplied by the number of steps of the ODE scheme and multiplied by the number of time steps of the simulation. This can be a limiting factor in applications with long forecasting horizons.

The number of hidden units per layer is chosen as a multiple of the augmented number of hidden states, as the latter represents the information bottleneck of the network. More hidden units lead to possibly more accurate prediction capabilities. This results from increased model flexibility. However, the additional model flexibility also increases the tendency of overfitting. As a consequence, the number of hidden units is decreased for applications that are safety critical (for example [We21]) or applications with long simulation horizons. The Akaike Information Criterion (AIC) and Bayes Information Criterion (BIC) are common evaluation metrics (see section 3.5.1) for developing models with reduced parameters.

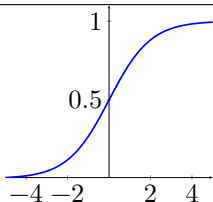
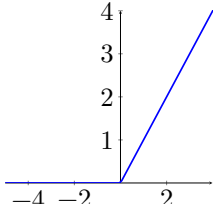
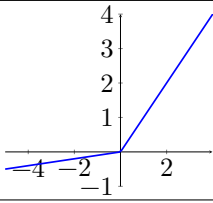
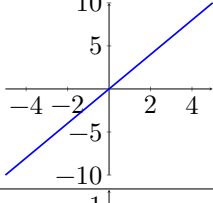
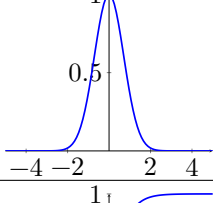
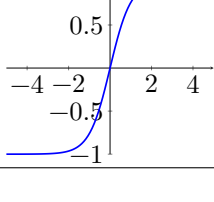
Dropout [Sr14] or Batch-Norm layers [SC15] are not applied. Normalization is discussed in the chapter 5.1. Dropout regularizes the network and increases its generalization capabilities during inference. However, a significant effect using dropout in the experiments could not be observed. Therefore, dropout is renounced.

3.4.1 Activation Functions

Regarding the activation functions, the LeakyReLU unit is applied. This choice is motivated by the history of activation functions. In the nineties, sigmoid and hyperbolic tangent activation functions were pervasive. Sigmoid suffered from the disadvantage that these functions are not symmetrical around zero. Hyperbolic tangent is superior in this respect, yet hyperbolic tangent suffers from gradient saturation, just like sigmoid. When the hyperbolic tangent input is an immense (positive or negative) value, the gradient of the network parameters given this input vanishes, making it hard for the optimization algorithm. ReLU function solve this issue, at least for large positive values.

Nevertheless, ReLU is not symmetrical around zero and has a non-smooth function. On the other hand, LeakyReLU applies a linear function on both sides, each with different slopes. These activation functions reduce the vanishing gradient effect, at least within one layer, for all real numbers [Le17; NH10]. Tab. 3.1 gives an overview of the activation functions.

Table 3.1: Overview of activation functions.

Name	Equation	Graphic
Sigmoid	$\sigma_S(x) = \frac{1}{1+e^{-x}}$	
ReLU	$\sigma_R(x) = \max(0, x)$	
LeakyReLU	$\sigma_R(x) = \max(\alpha \cdot x, x)$ $\alpha \in \mathbb{R}_{>0}$	
Linear	$\sigma_L(x) = \beta \cdot x$ $\beta \in \mathbb{R}$	
Gauss	$\sigma_G(x) = e^{-(x^2)}$	
tanh	$\sigma_T(x) = \frac{e^{2 \cdot x} - 1}{e^{2 \cdot x} + 1}$	

3.4.2 Model Ensemble

Additional robustness for data-based models can be gained when model ensembles are applied. In literature, model ensembles are also called model committees [De11a]. The idea is to reduce model variance by training several models in parallel. Consequently, model variance can be explicitly traded off for the computational effort. Furthermore, comparing the different models leads to an explicit estimate of the output variance, so an explicit measure of the certainty of the models is available.

Note that model ensembles do not refer to the multi-model approach for long-time horizons as discussed in section 3.5.5.

Three main design choices need to be made to design model ensembles.

- **Data splitting.** Submodels can be trained on different batches or even on completely different data sets. Distinguishable data sets increase the robustness of the ensemble, as local minima are less likely. However, it increases the development effort and complicates the model analysis. Due to different training and validation data, the submodels are not comparable during these training and validation phases. Only a common test data set can provide insight and a comparison of submodels. As a further disadvantage, the submodels cannot be trained in an ensemble setting. Therefore, any effects caused by different model types or caused by the output aggregation cannot be incorporated into the loss function. So exactly for the same reason that the models are more independent and more robust, the ensemble performance might be diminished as the independence requires a loss of information. As a consequence, the same data is applied for all submodels.
- **Model types.** In principle, it is possible to combine any model type. From a developers' perspective, combining many models is detrimental, as it gets harder to validate the contribution of every single model. Without knowledge of the specific model contributions, model tuning is aggravated. Given sufficient computational resources, this disadvantage can be diminished using black-box hyperparameter tuning. In this work, each model's attributes are analyzed in depth. As a consequence, only a single base model type is utilized. Another advantage of using a single base model is that the model interface of the ensemble and the submodel can be identical. This enables a direct comparison of the ensemble as a whole with the submodel itself. A common interface is still possible for different submodels. However, difficulties associated with a common interface increase with the diversity of the submodels.
- **Output aggregation.** For classification tasks, a voting method can be applied. The arithmetic mean function or the median can be applied for regression tasks as applied in this thesis. The median is more robust to outliers. The median is not

defined for ensembles with an even number of submodels. In this case, a linear interpolation between the output of the two resulting submodels can be utilized. However, depending on the concrete implementation, the median neglects the results of the non-median submodels, which complicates training. Recall all submodels contribute to the arithmetic mean aggregation, yet only one or two submodels define the median aggregation. Further submodels do not even fractionally influence the median. So it is likely that some submodels, due to their worse initial performance, are not trained at all. To ensure the contributions of all submodels, applying the mean function for output aggregation is recommended.

3.5 Training Configuration

3.5.1 Loss Functions

The neural network is trained using a (usually first-order and gradient-based) optimization technique, such as ADAM [KB14] or SGD [Am93; Bo12]. For notational convenience, all trainable parameters are summarized in a vector $\Theta \in \mathbb{R}^P$ with some $P \in \mathbb{N}$ parameters and with the (locally) optimal solution $\Theta^* \in \mathbb{R}^P$. Given an initial hidden state x_0 , a target sequence y , and an exogenous input sequence u , the loss function is defined as

$$\begin{aligned} \Theta^* = \arg \min_{\Theta} & L_{Fit}(x_0, y, u, \Theta) \\ & + \lambda_B L_{Barrier}(x_0, y, u, \Theta) \\ & + \lambda_R L_{Reg}(\Theta) \end{aligned} \tag{3.13a}$$

with scaling parameters $\lambda_B \in \mathbb{R}_{>0}$, $\lambda_R \in \mathbb{R}_{>0}$ for the optional barrier $L_{Barrier}(\cdot)$ and regularization $L_{Reg}(\cdot)$ loss functions, respectively. The explicit initial state can be omitted if the model provides a method to compute it, for example using Deep Encoder Networks (see section 3.2).

Regularization

In order to increase the generalization capabilities of the estimated model, one can include a technique named parameter regularization or weight decay [KH91]. The main idea is to insert an additional term in the cost function, which punishes too large parameters and encourages the same numerical range for all parameters. The regularization loss in (3.13) can be defined as Tikhonov Regularization,

$$L_{Reg,Tik} = \frac{1}{T} \int_0^T \|\Theta(t)\|_2 dt \quad (3.14)$$

with the L2-Norm $\|\cdot\|_2$. Time-dependent trainable parameters $\Theta(t)$ are not applied. Therefore, the integral in (3.14) reduces to a sum and can finally be expressed by

$$L_{Reg} = \|\Theta\|_2. \quad (3.15)$$

It is possible to include other norms or a weighting matrix to tune the amount every single parameter contributes to the loss function.

Regression

For the fit loss function in a regression task, Tab. 3.2 and Tab. 3.3 give an overview.

Regression tasks often apply the MSE as a loss function. MSE provides smooth first derivatives, is computationally efficient to compute, provides the best least-squares performance, and focuses on outliers and therefore improves the robustness of the model. When maximum accuracy in the sense that the mean error is required, or if a one-sided estimation tendency is intended, the Pinball loss is recommended. The M4 machine learning forecasting competition was won using a Pinball loss [Sm20].

A closely related task is the choice of one or multiple metric functions for algorithm development. For metrics functions, comparability across different applications and interpretability is essential. Regarding comparability, a normalization of the metric, for example, a percentage range instead of absolute target values, is desirable. This is provided in Normalized Root Mean Squared Error (NRMSE), Coefficient of Determination Coefficient of Determination (R^2), Pinball, Mean Absolute Percentage Error (MAPE) and Smoothed Mean Absolute Percentage Error (SMAPE). Regarding interpretability, it is useful if the metric retains the unit of the output and target values, such as Root Mean Squared Error (RMSE) or Mean Absolute Error (MAE). For developers, insight is helpful about the parameter efficiency concerning the number of training data points, using AIC or BIC.

Table 3.2: Selected loss functions for regression applications (Part I of II).

Name & Formula	Properties
Mean Squared Error (MSE) $L_{MSE} = \frac{1}{N} \sum_{n=0}^{N-1} (\hat{y}_n - y_n)^2$	good first derivative focus on outliers difficult to interpret
Root Mean Squared Error (RMSE) $L_{RMSE} = \sqrt{L_{MSE}}$	non-smooth first derivative focus on outliers unit is interpretable
Normalized Root Mean Squared Error (NRMSE) $L_{NRMSE} = \frac{L_{RMSE}}{\sigma_y}$ $\sigma_y = \sqrt{\text{var}(y)}$	non-smooth first derivative focus on outlier's loss is normalized
Bayes Information Criterion (BIC) $L_{BIC} = 2p_1 \ln(p_2) + \ln(L_{MSE})$ p_1 : number of para p_2 : number of data points	accounts for model complexity by means of model parameters and required data
Akine Information Criterion (AIC) $L_{AIC} = 2p_1 + p_2 \ln(L_{MSE})$ p_1 : number of para p_2 : number of data points	accounts for model complexity by means of model parameters and required data
Coefficient of Determination (R^2) $L_{R2} = 1 - \frac{\hat{y}_n - y_n}{\bar{y}_n - y_n}$ $\bar{y}_n = \frac{1}{N} \sum_{n=0}^{N-1} y_n$	weights all predictions equally loss is normalized
Mean Absolute Error (MAE) $e_{MAE} = \frac{1}{N} \sum_{N=0}^{N-1} \hat{y}_n - y_n $	derivative is non-smooth easy to interpret weights all predictions equally
Pinball Loss $L_{pinball} = \frac{1}{N-n_0} p_n$ $p_n = \begin{cases} (\hat{y}_n - y_n)\lambda_P & \text{if } \hat{y}_n \geq y_n \\ (y_n - \hat{y}_n)(1 - \lambda_P) & \text{otherwise} \end{cases}$ $0 < \lambda_P < 1$	one-sided MAE reduces one-sided errors

Table 3.3: Selected loss functions for regression applications (Part II of II).

Name & Formula	Properties
Mean Absolute Percentage Error (MAPE) $L_{MAPE} = \sum_{n=0}^{N-1} \frac{ \hat{y}_n - y_n }{M y_n }$	derivative is non-smooth easy to interpret weights all predictions equally loss is normalized
Smoothed Mean Absolute Percentage Error (SMAPE) $L_{SMAPE} = \sum_{n=0}^{N-1} \frac{ \hat{y}_n - y_n }{2N(\hat{y}_n + y_n)}$	derivative is non-smooth easy to interpret weights all predictions equally loss is normalized symmetric target and output

3.5.2 Inequality Barrier Methods

Modern deep learning libraries such as *PyTorch* [Pa19] or *TensorFlow* [Ma15] do not support constrained optimization. Barrier methods incorporate the constraints as an additional term in the cost function.

Following [Co14] barrier methods can be divided into the extreme barrier, progressive barrier, differentiable barrier, and classic penalty methods, defined in the following.

Given a constrained optimization problem,

$$\Theta^* = \arg \min_{\Theta} L_{Fit}(y, u, \Theta) \quad (3.16)$$

$$s.t. \quad c(\Theta)_i \leq 0 \quad \forall i \in I \quad (3.17)$$

with a loss function $L_{Fit}(\cdot)$ and a (multiple) nonlinear constraint functions $c(\Theta)_i$. Barrier and penalty methods define a new loss function $L_{Barrier}(\cdot)$, which incorporates both the original loss and all constraints. Extreme barrier methods are infinity if any constraint is violated,

$$L_{Barrier}(\cdot) = \begin{cases} L_{Fit}(\cdot) & \text{if } \max(c(\Theta)_i) \leq 0 \\ \infty & \text{otherwise.} \end{cases} \quad (3.18)$$

Although extreme barriers ensure an unbiased solution, they tend to be numerically detrimental. On the other side, classical penalty functions with a scaling $\lambda_1 \leq 0$ are defined as

$$L_{Barrier}(\cdot) = \begin{cases} L_{Fit}(\cdot) & \text{if } \max(c(\Theta)_i) \leq 0 \\ L_{Fit}(\cdot) + \lambda_1 \text{relu}(c_i)^2 & \text{otherwise.} \end{cases} \quad (3.19)$$

Classical penalty functions are numerically favorable because a discontinuity in the loss function is avoided. However, classical penalty methods cannot guarantee feasibility. Differentiable log barrier methods are defined as

$$L_{Barrier}(\cdot) = L_{Fit}(\cdot) - \lambda_1 \sum_i^I \log(-c_i). \quad (3.20)$$

Log barrier methods provide the best numerical properties, yet they completely modify the original loss function and introduce a bias in the estimate. Progressive barrier methods are defined as

$$L_{Barrier}(\cdot) = \begin{cases} L_{Fit}(\cdot) & \text{if } \max(c_i) \leq \lambda_2 \\ L_{Fit}(\cdot) + \lambda_1 \text{relu}(\max(c_i) + \lambda_2)^2 & \text{if } \lambda_2 < \max(c_i) \leq 0 \\ \infty + \max(c_i) & \text{otherwise.} \end{cases} \quad (3.21)$$

with a scalar safety offset $\lambda_2 < 0$. The formulation $\infty + \max(c_i)$ ensures a differentiable solution, guiding the numerical solver. The progressive barrier methods are recommended, which obtain numerical efficiency, keep the original optimal solution, and still guarantee to satisfy the constraints.

3.5.3 Equality Barrier Methods

If the constraint function is in the form

$$\Theta^* = \arg \min_{\Theta} L_{Fit}(y, u, \Theta) \quad (3.22)$$

$$s.t. \quad c(\Theta)_i = 0 \quad \forall i \in I \quad (3.23)$$

an effective and easy-to-implement solution is to apply the MSE loss with a scaling factor λ_3 ,

$$L_{Barrier}(\cdot) = \lambda_3 \frac{1}{I} \sum_{i=0}^{I-1} c(\Theta)_i^2. \quad (3.24)$$

For instance, the DAE network explained in section 3.4 can be trained using the following Python code E.5 for the *PyTorch* library. The constant $\lambda_3 \approx 10^{-3}$ requires to be adapted

for each data set. The value is chosen rather small not to disturb the unbiased solution too much. The code is given in appendix E.5.

3.5.4 Reduction of Overfitting

An effective method to prevent overfitting is early stopping [Pr98]. Therefore, the data is split into three groups: training, validation, and testing. The model is optimized on the training data. Validation data is applied for model metric evaluation (see section 3.5.1) and for early stopping. Test data enables independent evaluation.

If the application gives no validation data, the original training data can be split into applied training and validation. Random split of the original training data is recommended for systems with non-uniform input excitation. So that validation sequences obtain a random starting time over the whole data horizon. As an alternative, data can be split into two coherent sets. As time series data typically inherits a strong auto-correlation, two coherent sets most likely increase the independence of training and validation data.

Fig. 3.5 demonstrates overfitting on a noisy sine wave. Training and validation data is split coherently. The upper figure presents the optimal solution, the lower figure tends to overfit. Training is stopped if the loss on the validation data (green dots) increases.

In the following, several types of stopping criteria are discussed. Logging the best validation loss and corresponding all model parameters in that iteration is recommended. After one stopping criteria is active, it is beneficial for model generalization capability and performance to discard the final model parameter iteration and load the model parameters from the checkpoint with the best validation error.

- **Stop if no improvement of the validation loss can be achieved over a given number of iterations.** Recommended criteria. Only little prior knowledge is required. Can handle loss processes with high variance.
- **Stop if the validation error successively increases for a given small number of iterations.** Reduces computational effort as fewer obsolete iterations are performed. Tendency to stop too early. Cannot cope with loss processes with high variance.
- **Stop by the number of iterations.** Can fail to get the best model as the number of iterations is not a measure of model performance. Easy to implement. Immune to loss processes with high variance.
- **Stop by the computation time limit.** Can fail to get the best model as the computation is not a measure of model performance. Easy to implement. Immune to loss processes with high variance.

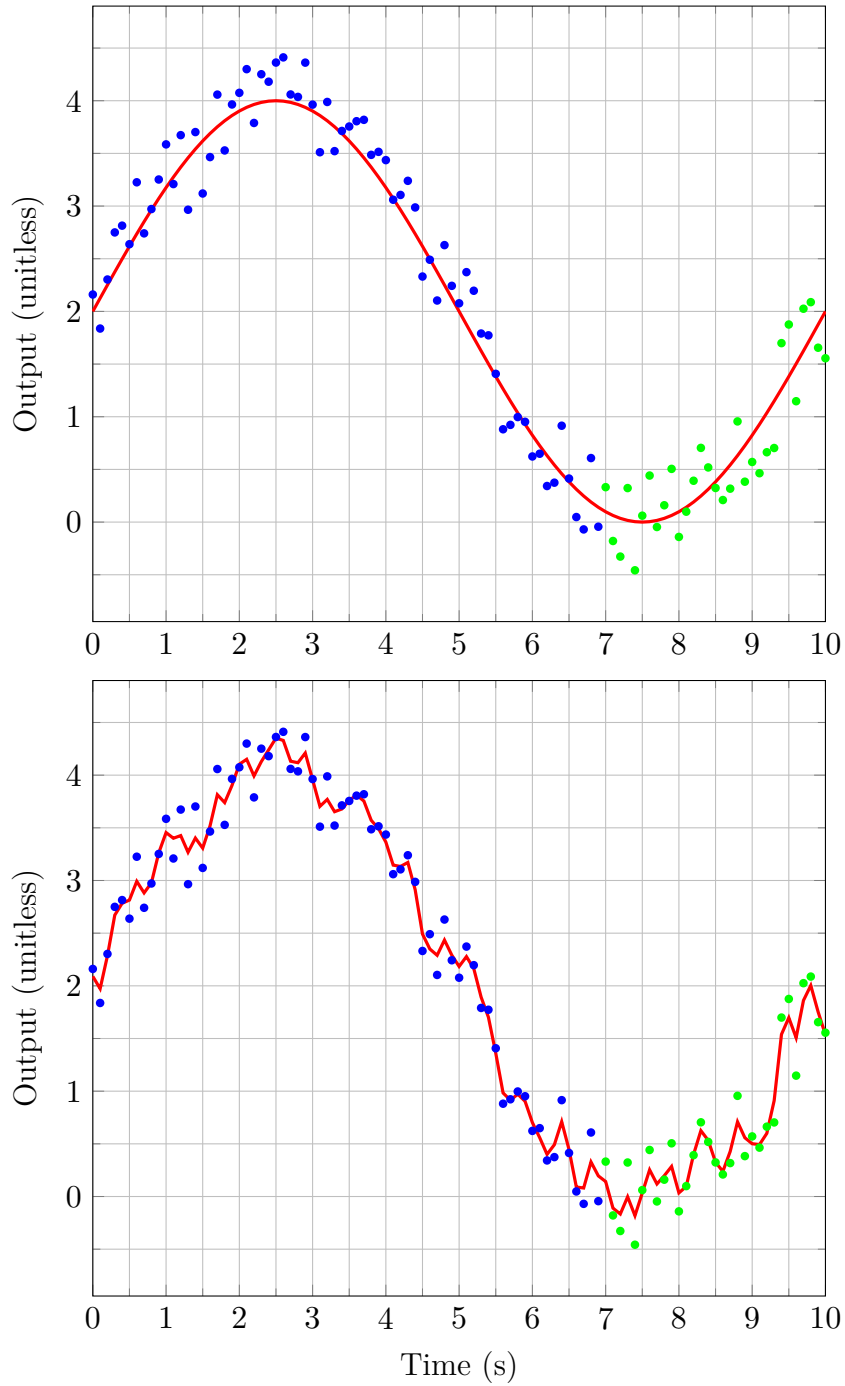


Figure 3.5: Demonstration of overfitting of a noisy sine curve. Blue dots: Training data. Green dots: Validation data. Red line: Model prediction. Compare with [LC13; We17].

- **Stop by surpassing the minimum error value.** Can fail to get the best model, depending on the user’s knowledge. Detailed prior knowledge or fine hyperparameter tuning is required. Immune to loss processes with high variance.
- **Stop by user interaction.** Fails the goal of automating the training pipeline. Can fail to get the best model, depending on the user’s knowledge. Recommended as backup criteria from a practical point of view for unforeseen events or unpredictable behavior.

3.5.5 Forecast Horizon

A central choice for time-series models is the forecast horizon, which depends on the applications. In a (one-step-ahead) prediction configuration, e.g., [WL98], the state x is completely measured for all time steps. Each predicted output and state, \hat{y}_{n+1} and x_{n+1} , is estimated based on the previously measured state and input, e.g., \hat{x}_n and u_n . The estimated state x_n is not fed back into the state network. Consequently, a one-step prediction mode leads to fast training and enables parallel computation. However, training in one-step prediction reduces accuracy for long-term predictions. The model can also be unstable (without enforcing stability constraints, see section 5.2).

If a long-term estimation is required in the application, it cannot be neglected in training. Therefore, in a (multistep-ahead) simulation configuration, only the initial state measurement x_0 is required, and all proceeding states are recursively taken from previous calculations. [DKR10] presents exemplary work. As a consequence, each successive simulated state x_{n+1} in training depends on the previous state estimate x_n and the input u_n . The feedback of estimated states leads to a potential accumulation of state errors and is, therefore, more comprehensive. The challenge can also be viewed as Pareto-Optimization. The difference regarding estimation performance between simulation and prediction configuration is two orders of magnitude [WDR21].

Note that it is irrelevant for the one-step prediction mode how far the predicted value lies in the future. This is because the sample time only influences the dominant effects to capture for the model. For example, a one-minute weather forecast depends on different dominant physical influences than a seven-day forecast. Nevertheless, both one-minute and seven-day forecasts do not encounter the issue of recursive error accumulation, as both are one-step predictions.

For most discrete-time models, the sample time in inference is equal to the sample time in training. Usually, the time grid is also uniform. However, these two properties are relaxed for continuous-time models.

All model types referred to in this thesis types support one-step-ahead prediction. Fur-

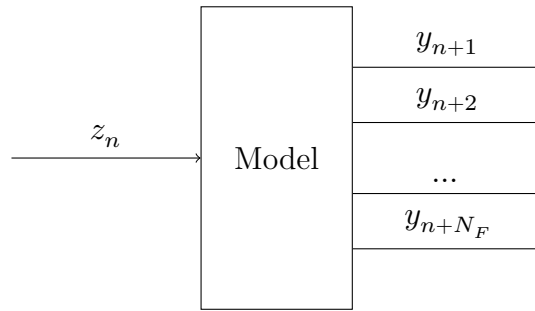


Figure 3.6: Multi-output model.

thermore, many models, such as regression trees, support one-step predictions beyond this thesis. However, three types of model architectures are presented in the following. The first two can convert one-step prediction models into multistep simulation models without facing the issue of error accumulation.

- Multi-output models.** The main idea is to convert the time dimension into output channels. This increases model output channels to $N_Y \cdot N_F$ with $N_F \in \mathbb{N}$ time steps in the simulations. As the number of output channels is limited by computational feasibility, the drawback of this architecture is that the forecast horizon is limited, too. Furthermore, this architecture is not flexible as the forecast horizon can only be reduced and not extended inference. Finally, this architecture leads to an unfavorable dependency between network architecture and forecast horizon, so one cannot be designed without influencing the other. Nevertheless, the accuracy of these models is good, as no prediction error is fed back to the network.
- Multi-models.** The multistep-ahead forecast can be cast into several one-step-ahead forecasts. This method takes advantage of the fact that it is not important for the one-step-ahead prediction of how far to predict into the future. The consequences of multi-models are similar to multi-output models. The forecast horizon is limited, the horizon is inflexible, and the developers need to cope with a dependency on architecture and horizon. However, the accuracy is good.

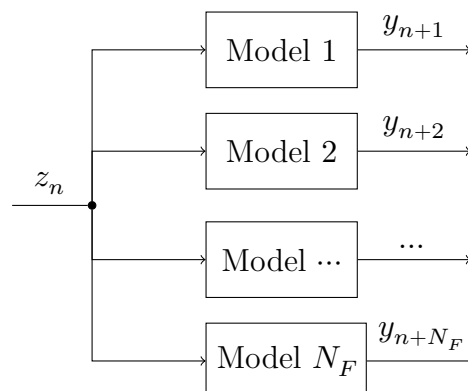


Figure 3.7: Multi-models.

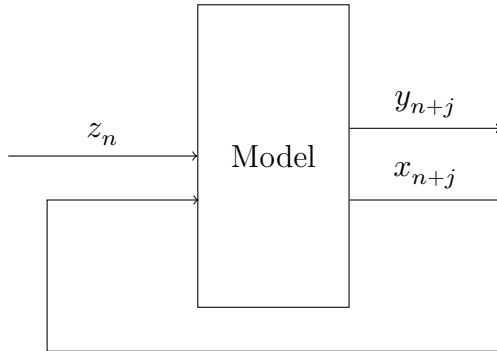


Figure 3.8: Recursive models.

- **Recursive models.** The previously estimated outputs are recursively and time delayed fed back into the model input. This is the most popular choice and the one applied throughout this thesis. It enables very long forecast horizons as it does not influence the network size. Furthermore, the forecast parameters can be flexibly adapted during inference. However, prediction errors are fed back into the model, so error accumulation and model stability require attention.

A special case of multistep-ahead simulation is present when the inference forecast horizon is larger than the training forecast horizon. This is the most challenging case and, subsequently, the one with the worst performance. In some applications, predicting many time steps into the future is mandatory. For example, see section 5.2. However, the required forecast horizon in inference can be so extensive that it would exceed the computer’s computational resources (memory and computation time) during training. In this case, the forecast horizon needs to be reduced during training to ensure computational feasibility. However, it implies a discrepancy between the loss function and the inference objective. The models capable of this task must be recursive, multi-output models, or multi-models.

The application usually determines the required class of the forecast horizon. For example, Model Predictive Control (MPC) requires multistep-ahead forecasts. A horizon that is as small as the application allows for is recommended. The fewer forecast steps in the inference horizon, the less Pareto-Optimality plays a role. Furthermore, recursive models are recommended. Whenever the application allows for it, avoiding inference horizons greater than the training horizons is recommended.

3.5.6 Discretize-then-Optimize vs. Optimize-then-Discretize

An important distinction for continuous-time NN is a Discretize-then-Optimize (Disc-Opt) or Optimize-then-Discretize (Opt-Disc) configuration. In Disc-Opt, the ODE is discretized in each step of the optimizer, and the optimizer only handles discrete values for estimated states and target variables. On the contrary, in Opt-Disc, the ODE is optimized in continuous-time using the Adjoint Equations [Ch18; OR20]. Only the trained model is

discretized in time for inference. For additional information on this distinction, see [HR18; RH20].

Opt-Disc significantly improves the memory efficiency of the algorithm, as demonstrated in [Ch18]. Yet, the quality of the gradients required for optimization depends on the accuracy of the ODE solver in Opt-Disc. Therefore, a high-order and usually adaptive-step ODE solver is required, with all implications discussed in section 3.3.4. Furthermore, Opt-Disc requires the implementation effort and the computational resources for solving the adjoint equations.

In contrast, Disc-Opt always produces good loss function gradients, as the ODE scheme is incorporated into the loss function. This enables good results even with simple ODE solvers such as Euler Backward, as demonstrated by the ResNet architecture. Independence of the optimization gradients from the ODE solver neglects the requirement of explicit ODE solver accuracy estimation. Therefore, fixed-step solvers such as Runge-Kutta, Midpoint, or Euler are suitable for Disc-Opt. Furthermore, fixed-step solvers cannot produce failed steps. In conclusion, AD tools do not require propagating through failed steps.

In contrast to image classification problems, physical models require NN with relatively few parameters. Therefore, the memory inefficiency of Disc-Opt is not a significant disadvantage in the system identification domain.

[Ki22] states a pointed summary of the comparison (quoted literally):

- Discretize-then-optimize – memory inefficient, but accurate and fast.
- Optimize-then-discretize – memory efficient, but approximate and a little slow.

4 Physical Robot Model and Control

This chapter explains the physics-based model and control of the KUKA KR300 R2500 ultra SE robot. Therefore, the nonlinear robot model described in section 2.2 serves as starting point. To improve the robot model, extensions of robot model components are derived in section 4.1. To fit the advanced nonlinear robot model to the demonstrator, sophisticated parameter identification is presented in section 4.2. A feedback and feed-forward controller based on the physical model is derived in section 4.3.

4.1 Advanced Nonlinear Robot Model

4.1.1 Elastic-Joint Model

For modeling joint flexibility, each motor is considered a rigid body connected to a driven link via a transmission device and a rotational spring, illustrated in Fig. 4.1. Such a system can be modeled as a two-mass oscillator leading to two generalized coordinates for every joint. The generalized coordinates are composed of the motor positions $\theta \in \mathbb{R}^{N_Q}$ and the joint positions $q \in \mathbb{R}^{N_Q}$. That leads to a $2N_Q$ DoF system of a robot, so a 12 DoF system for robot kinematics with 6 joints. The motor velocity is $\dot{\theta} \in \mathbb{R}^{N_Q}$, the motor acceleration is $\ddot{\theta} \in \mathbb{R}^{N_Q}$, the link velocity is $\dot{q} \in \mathbb{R}^{N_Q}$ and the link acceleration is $\ddot{q} \in \mathbb{R}^{N_Q}$. The motor velocity can be measured using the motor resolvers. The link angle can be directly measured using the link side mounted SE. The dynamical model of the considered robot manipulator is derived employing Lagrange's equations of the second kind. The transmission ratios are assumed to be high enough so that inertial couplings in the acceleration between the motors and the links can be neglected [DB16]. The governing equations are

$$J\ddot{\theta} + U^{-1}\tau_E(\theta, q) = \tau_M \quad (4.1a)$$

$$M_L(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) + \tau_F(\dot{q}) + \tau_H(q) = \tau_E(\theta, q) \quad (4.1b)$$

where (4.1a) describes the dynamical model of the motor side and (4.1b) represents the

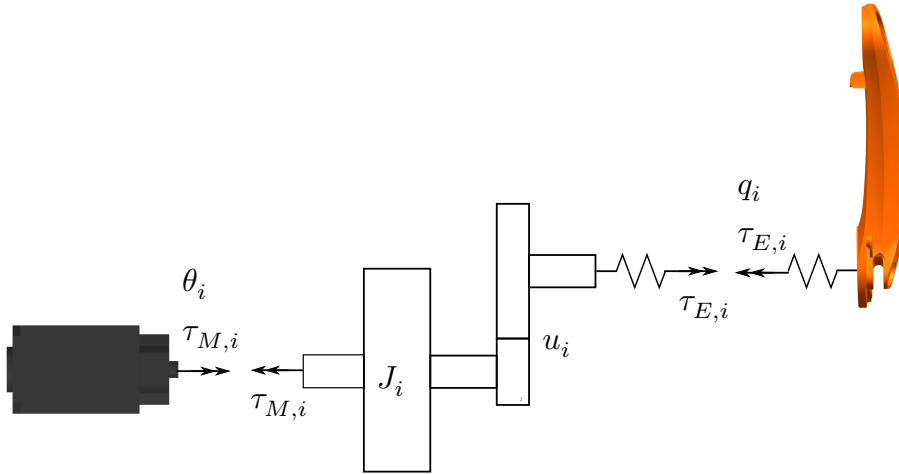


Figure 4.1: Representation of a flexible robot joint, exemplary for joint 2.

dynamics of the links. The two dynamical systems are coupled by the generalized elastic torque vector $\tau_E(\theta, q) \in \mathbb{R}^{N_Q}$. The inertia matrix of the rotors is $J = \text{diag}(J_1, \dots, J_N) \in \mathbb{R}^{N_Q \times N_Q}$. The transmission factor of each joint is u_i , summarized in a diagonal matrix $U = \text{diag}(u_1, \dots, u_N) \in \mathbb{R}^{N_Q \times N_Q}$. The pose-dependent inertia matrix of the links without motors is denoted by $M_L \in \mathbb{R}^{N_Q \times N_Q}$ with

$$M_L = M - JU^2. \quad (4.2)$$

4.1.2 Stiffness Model

The stiffness model considers backlash, lost motion, and linear elasticity. The lost motion describes an effect between backlash and linear elasticity, where not all tooth flanks are in full contact. During backlash the elastic torque vanishes, e.g., $\tau_E = 0$. In the lost motion range, the stiffness is modeled as linear with a smaller slope and an offset. In the torsional rigidity range, the stiffness is modeled linear with an offset, as shown in Fig. 4.2.

The lost motion stiffness coefficient is c_{LM} and the torsional rigidity stiffness coefficient is c_{TR} . The angular backlash range is ϕ_{B^*} and the angular lost motion range is ϕ_{LM} . The torsional rigidity is an effect between 50% and 100% of the nominal torque. Lost motion effects occur directly after the backlash and are measured between $\pm 3\%$ of the nominal torque [TPA16]. Therefore, an effective backlash angle $\phi_B = \phi_{B^*} + \phi_{LM}$ which ensures correct modeling of the lost motion and torsional rigidity range is introduced. With the link torsion angle and its sign function

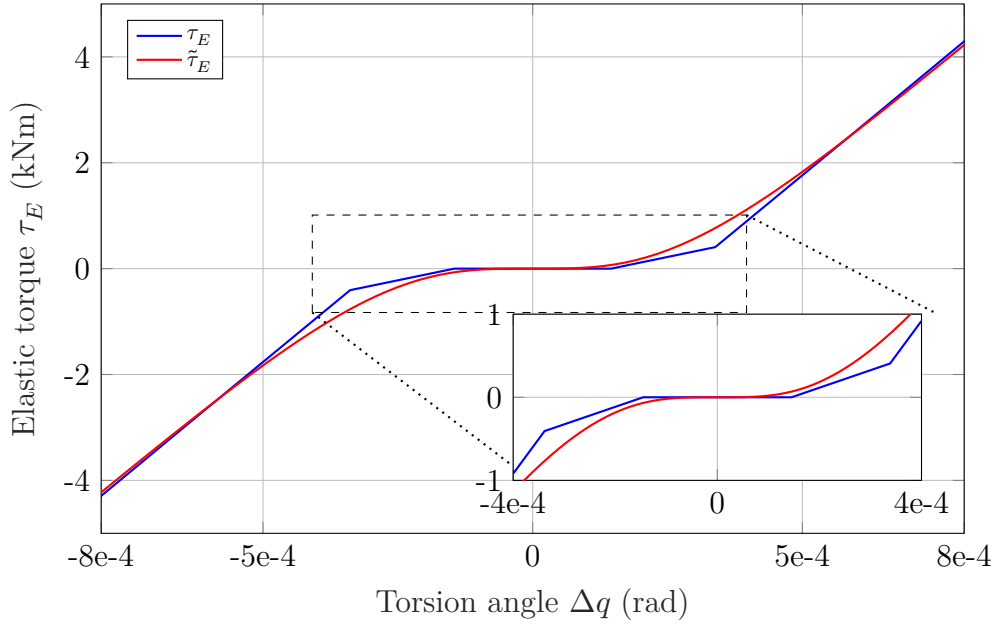


Figure 4.2: Discontinuous and continuous approximation for stiffness model with the effects of backlash, lost motion and linear elasticity, exemplary for joint 1.

$$\Delta q = \frac{\theta}{u} - q \quad (4.3)$$

$$\sigma = \text{sign}(\Delta q) \quad (4.4)$$

and the elastic torque offset $\tau_{E,0} = c_{LM} \phi_{LM}$, the elastic torque can be obtained

$$\tau_E = \begin{cases} 0, & |\Delta q| \leq \phi_{B^*} \\ c_{LM} \left(\Delta q - \frac{\phi_{LM} \sigma}{2} \right), & \phi_{B^*} < |\Delta q| \leq \phi_B \\ c_{TR} \Delta q + \tau_{E,0} \sigma, & \text{otherwise.} \end{cases} \quad (4.5)$$

Although (4.5) describes the nonlinear stiffness precisely, it is difficult to use it in parameter estimation or advanced control strategies since the function is not continuously differentiable. Problems with (4.5) potentially arise with all derivative-based algorithms. In particular, many AD tools, such as implemented in the *Symbolic Math Toolbox* from *MATLAB* [Ma19], the *PyTorch* [Pa19] or the *TensorFlow* library [Ma15] require continuously differentiable equations. A continuously differentiable elastic torque function is presented, which originates from time-domain considerations. Consider a first-order dynamical system (PT1-System) with a ramp input and set the slope of the ramp equal to the linear stiffness.

$$T \frac{\partial y(t)}{\partial t} + y(t) = c_{TR} t. \quad (4.6)$$

The system output slope will converge towards the input slope, with a constant time offset. Although a time-domain function is not required, the algebraic solution of (4.6) can still be used. The time variable t can be replaced with the torsion angle Δq , the output $y(t)$ with the elastic torque $\tau_E(\Delta q)$ and setting all initial conditions to zero. In this work, a 3^{rd} -order system to increase the curvature of the function is utilized. Further, the curvature can be achieved by applying for an arbitrary higher order. Two issues must be considered if an order greater than 1 is applied. First, all poles of the transfer function must coincide. Second, the poles must be adapted to the order $p \in \mathbb{N}$ of the ODE by setting $T = \phi_B/p$. This ensures that the asymptote of the solution matches the full-contact stiffness. The variable-order ODE is given by

$$G(s) = \left(\frac{1}{Ts + 1} \right)^p, \quad (4.7)$$

which leads to the general order partial differential equation

$$\sum_{j=0}^p \left(\frac{\phi_B}{p} \right)^j \binom{p}{j} \frac{\partial^j y(t)}{\partial t^j} = c_{TR} t \quad (4.8)$$

with the binomial coefficient $\binom{p}{j}$. The third-order equation results in

$$\left(\frac{\phi_B}{p} \right)^3 \frac{\partial^3 y(t)}{\partial t^3} + 3 \left(\frac{\phi_B}{p} \right)^2 \frac{\partial^2 y(t)}{\partial t^2} + 3 \left(\frac{\phi_B}{p} \right) \frac{\partial y(t)}{\partial t} + y(t) = c_{TR} t. \quad (4.9)$$

To derive a differentiable approximation $\tilde{\tau}_E \approx \tau_E$ of (4.5), (4.9) is solved in the time domain and the substitution of the stiffness model leads to

$$\begin{aligned} \tilde{\tau}_{E,+}(\Delta q) &= c_{TR} \Delta q - c_{TR} \phi_B \\ &+ c_{TR} \phi_B e^{-(3\Delta q/\phi_B)} \\ &+ 2 c_{TR} \Delta q e^{-(3\Delta q/\phi_B)} \\ &+ 3 c_{TR} / (2 \phi_B) (\Delta q)^2 e^{-(3\Delta q/\phi_B)} \end{aligned} \quad (4.10)$$

for all $\Delta q \geq 0$ and

$$\tilde{\tau}_{E,-}(\Delta q) = -\tilde{\tau}_{E,+}(-\Delta q) \quad (4.11)$$

for all $\Delta q < 0$. A complete, continuously differentiable elastic torque function is given by

$$\tilde{\tau}_E = \tanh(s_{E1} \Delta q) \tilde{\tau}_{E,+}(\Delta q \cdot \tanh(s_{E1} \Delta q)) \quad (4.12)$$

for all $\Delta q \in \mathbb{R}$ with the hyperbolic tangent $\tanh(\cdot)$. It is essential for the tangent slope factor s_{E1} that $s_{E1} \gg 3/\phi_B$ holds. Note that the slope factor is not upper-bounded, except for limits due to numerical considerations. The continuously differentiable stiffness curve is depicted in Fig. 4.2.

A continuously differentiable stiffness function is required for many model applications, such as MPC or parameter identification. However, a flatness-based control is an exception, where both, (4.12) and the inverse stiffness function (4.13) can be employed. Although the inverse stiffness function is neither common nor applicable in general, it is advantageous in the case of flatness-based control to use the following form

$$\Delta q = \begin{cases} 0, & \tau_E = 0 \\ \frac{\tau_E}{c_{LM}} + \frac{\phi_{B*}}{2} \text{sign}(\tau_E), & 0 < |\tau_E| \leq \tau_{E,0} \\ \frac{\tau_E}{c_{TR}} + \phi_B \text{sign}(\tau_E), & \tau_{E,0} < |\tau_E|, \end{cases} \quad (4.13)$$

since the inverse stiffness function reduces the number of required exponential functions from eight in (4.12) to one in (4.14). For (4.13), the same method as for the nonlinear friction can be applied, and the nonlinear, inverse, continuously differentiable stiffness function

$$\Delta q = \frac{\tau_E}{c_{TR}} + \frac{2\phi_B}{1 + e^{-s_{E2} \tau_E}} - \phi_B \quad (4.14)$$

for all $\Delta q \in \mathbb{R}$ with the elastic smoothing factor s_{E2} is obtained.

4.1.3 Advanced Inertia, Coriolis, Centripetal and Gravitational Model

To embed the model in an optimization method, e.g., for model parameter identification, the model required numerical improvements to reduce the computational effort. Therefore, a tool developed by Joern Malzahn [Ma12] was utilized, which converted the model to a symbolic representation consisting only of simple operators, such as addition, multiplication, sine, and cosine functions. This symbolic representation is applied in the *MATLAB* robot identification suite and the *PyTorch* continuous-time NN environment

for machine learning. Some modifications significantly reduced the computational effort, for example, the batch-mode estimation yielded a reduction factor of 64. In the *MATLAB* version, the symbolic representation is translated to C-code and then pre-compiled to speed up computation further. The *PyTorch* library embeds the symbolic code within its AD tools. However, this code remains the computational bottleneck of the robot model. The symbolic representation consists of approximately 2000 lines of code, spread over several submodules, with all in all approximately 2000 additional constants, 10000 multiplications, 2500 additions, 2000 subtractions, 50 cosine functions, 60 sine functions, and 30 square functions. Mainly the calculation of the pose-dependent dynamic effects, such as the estimation of $M(q)$, $C(q, \dot{q})$, and $g(q)$ requires computational resources. To generate this symbolic representation, Joern Malzahn split the estimation of, e.g., the inertia matrix for each matrix row into a separate computational task to account for the computational limitations of the *MATLAB symbolic toolbox*. Further, reducing computational load is especially important for higher-order ODE solvers. The inertia, Coriolis, centripetal matrices, and gravitational loads are computed on demand, given that a sufficiently large change in position or velocity occurs in any joint. The thresholds are set to $q_{threshold} = 2 \text{ deg}$ and $\dot{q}_{threshold} = 2 \text{ deg/s}$. These threshold values are chosen as a trade-off between computational complexity and significant change of $M(q)$, $C(q, \dot{q})$, and $g(q)$. The source code is given in appendix E.1.

4.1.4 Hydraulic Weight Counterbalance

The HWC is modeled as a static hydraulic torque $\tau_{H,S}(q_2) \in \mathbb{R}$ which acts only on joint 2,

$$\tau_H(q) = [0, \tau_{H,S}(q_2), 0, 0, 0, 0]^T. \quad (4.15)$$

The static hydraulic torque $\tau_{H,S}(q_2)$ is generated by a hydraulic force $F_{H,S}$ and an orthogonal height $h(q_2)$. The height $h(q_2)$ can be derived from geometry, as depicted in Fig. 4.3, which results in

$$dl(q_2) = \sqrt{D^2 + k^2 - 2Dk \cos(q_2)} - l \quad (4.16)$$

$$h(q_2) = \frac{Dk \sin(q_2)}{l + dl(q_2)} \quad (4.17)$$

with the distance D from joint center to HWC attachment, HWC attachment radius k , base length $l = D - k$ and extension length $dl(q_2)$. The hydraulic pressure p is generated by and equal to the nitrogen gas pressure in the bladder accumulators. The ideal gas law is applied, including a compressibility factor Z , and

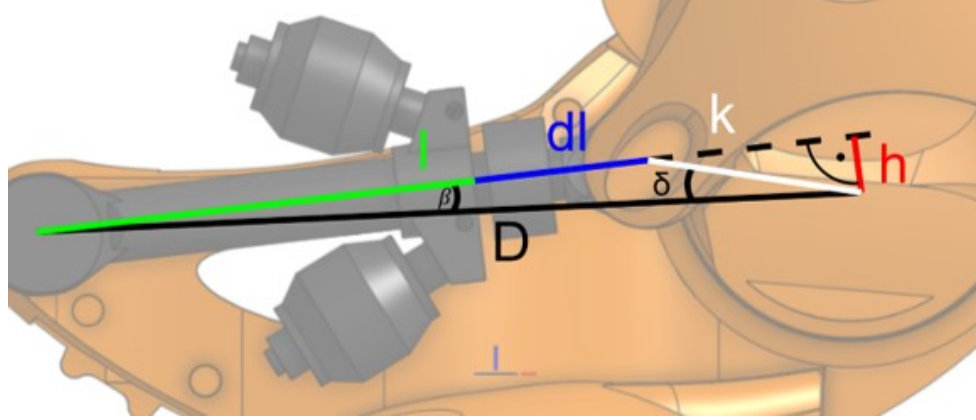


Figure 4.3: The geometry of the Hydraulic Weight Counterbalance of KUKA KR300 R2500 ultra SE robot [Ha18a].

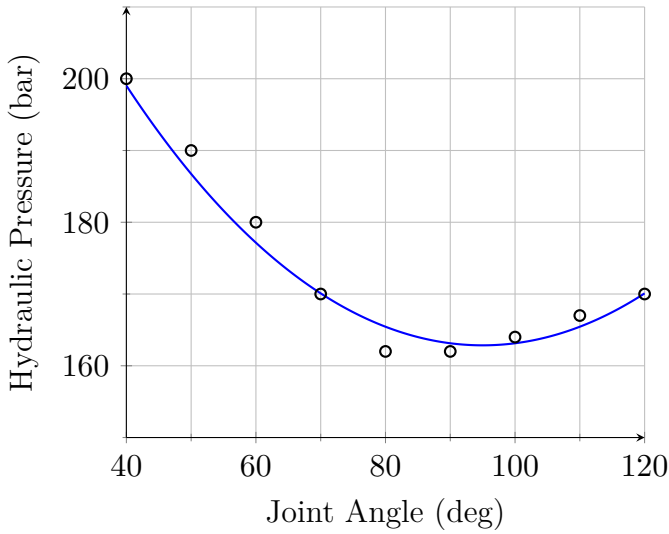
$$p(q_2) = \frac{m_N Z R_S T_N}{V_{max} - dl(q_2)A} \quad (4.18)$$

$$\tau_{H,S} = h(q_2)\tau_{S,F} = h(q_2) \cdot p(q_2)A \quad (4.19)$$

with the specific gas constant R_S , the nitrogen mass m_N , the nitrogen temperature T_N , the maximum bladder volume V_{max} and the effective piston area A is obtained. All geometric parameters are given in the CAD model. The temperature and the compressibility factor are assumed to be constant. The nitrogen mass and the bladder volume are identified using pressure measurements in several static joint positions. The resulting parameters are given in Tab. 4.1. The data-based estimation of the max bladder volume and the nitrogen mass lead to plausible values: The measurements of the static positions, together with the overall HWC model, are presented in Fig. 4.4.

Table 4.1: Hydraulic Counter Weight model parameters.

Parameter	Symbol	Value	Unit
Distance from base anchor point to joint center	D	800	mm
Distance from joint anchor point to joint center	l	200	mm
Rod diameter	d	30	mm
Piston diameter	p	55	mm
Nitrogen mass	m_N	$2 \cdot 0.0795$	kg
Max bladder volume	V_{max}	$2 \cdot 0.00048$	m ³
Compressibility factor nitrogen (176 bar, 328 K)	Z	1.03	
Real gas factor nitrogen	R_S	296.8	J/kg/K
Temperature	T_N	328	K



angle (deg)	pressure (bar)
120	170
110	167
100	164
90	162
80	162
70	170
60	180
50	190
40	200

Figure 4.4: Hydraulic Weight Counterbalance pressure model (blue line) compared with static pressure measurements (black circles) for several joint positions. Torque measurements are not available. Maximum joint movement is from 140 deg to 5 deg. Model is validated for the range of 120 deg to 40 deg, due to cell pose limitations.

4.1.5 Advanced Friction Model

Note that a good friction model is essential for high-accuracy robot control. As a result, a nonlinear friction model based on [Di18; GDH01] was applied. The extended friction model exploits an asymmetrical, viscous, Coulomb, and degressive friction term

$$\tau_F(\dot{q}) = f_{asym} + f_v \dot{q} + f_c \tanh(s_f \dot{q}) + f_a \tanh(f_b \dot{q}) \quad (4.20)$$

with viscous friction f_v and Coulomb friction f_c as defined in section 2.2.4. The additional zero drift error of friction torque is $f_{asym} \in \mathbb{R}^{N_Q}$ and degressive friction coefficients $f_a \in \mathbb{R}^{N_Q}$ and $f_b \in \mathbb{R}^{N_Q}$. The degressive friction torque routes back to [GDH01] and models a saturation of the friction torque in the high-velocity range. Modeling this degressive friction behavior matches various measurements. Furthermore, the Coulomb friction is approximated by $\text{sign}(\cdot) \approx \tanh(s_f(\cdot))$ for differentiable friction in the trajectory and identification optimization algorithm. In contrast to f_b , the smoothness factor s_f is not a physically motivated model parameter, and $s_f \gg f_b$ is ensured.

Defining the extended friction model is one matter; the estimation of friction parameters is the other. Section 4.2 will solely exploit the identification of proper friction model parameters.

4.2 Friction Parameter Estimation

Depending on the temperature, the friction load can vary from 20 % to 80 % of the motor torque [Ku08]. Therefore, an accurate friction identification is essential for sophisticated feed-forward control. The standard dynamic robotic model is utilized to the greatest extent possible for improved comparability and integration with other research findings. Only two extensions necessary for the riding joint industrial robot are investigated: An advanced friction model and a model considering the HWC acting on joint 2. The friction identification approach consists of two distinct steps: A first step deals with trajectory optimization, e.g., finding the set of trajectories that excite the parameters well. The fundamental chicken-egg problem that a model-based trajectory optimization estimates the excitation trajectory, which is then utilized to identify the model's parameters arises. Therefore, a global optimization over a set of initial models in the entire parameter space is chosen rather than applying a single model based on prior knowledge. The second step estimates the parameters based on the measurements. This global optimization can be performed at run-time. The presented methods solve this second step computationally sufficiently on up-to-date hardware.

4.2.1 Identification Algorithm

A linearly constrained, nonlinear objective global optimization problem is solved. As the elastic joints are not a part of the identified parameters, the optimization problem is based on the rigid body model (2.1). Furthermore, the identification is enabled as an online procedure, meaning that the optimization problem is designed to be computed at run time. This led to several implications regarding an efficient implementation and further details on the algorithm. Online computation is entirely optional. The algorithm can be applied to an offline data set with minor modifications.

$$\begin{aligned}
\tilde{\Theta}^* = \underset{\tilde{\Theta}}{\operatorname{argmin}} & \left(\frac{\lambda_1}{K} \cdot \sum_{k=0}^{K-1} (q_{k,meas} - q_{k,sim}(\tilde{\Theta}))^2 \right. \\
& + \frac{\lambda_2}{K} \cdot \sum_{k=0}^{K-1} (\dot{q}_{k,meas} - \dot{q}_{k,sim}(\tilde{\Theta}))^2 \\
& + \frac{\lambda_3}{K} \cdot \sum_{k=0}^{K-1} (\tau_{k,meas} - \tau_{k,sim}(\tilde{\Theta}))^2 \\
& \left. + \frac{1}{V} \sum_{v=0}^{V-1} \lambda_{4,v} \cdot (\tilde{\Theta}_v - \tilde{\Theta}_{v,prev}^*)^2 \right) \\
s.t. & \quad \tilde{\Theta}_{lb} \leq \tilde{\Theta} \leq \tilde{\Theta}_{ub}.
\end{aligned} \tag{4.21}$$

The optimal parameter vector is $\tilde{\Theta}^*$, the search parameter vector is $\tilde{\Theta}$, upper and lower

parameter bounds are $\tilde{\Theta}_{ub}$ and $\tilde{\Theta}_{lb}$ respectively and the previously identified parameter vector is $\tilde{\Theta}_{prev}^*$. Each parameter vector $\tilde{\Theta}$ contains $V \in \mathbb{N}$ values. The dataset contains some $K \in \mathbb{N}$ time steps. The discrete-time index is $k \in \mathbb{N}$ and the parameter vector index is $v \in \mathbb{N}$. The four objective terms in (4.21) can be combined arbitrarily using the weight factors $\lambda_{(\cdot)}$. The objective terms feature the link position MSE, the link velocity MSE, the motor torque MSE, and a regularization term. The regularization reflects that the parameter change is slow compared to state variable change rates and clips the currently identified parameter set to the previous one. At the first iteration, clipping can be omitted by setting $\tilde{\Theta}_{prev}^* = 0$ if no a priori knowledge about the initial parameters is available. The nonlinear closed-loop formulation in (4.21) requires a global optimization problem but opens up some unique possibilities:

- **Separation of modeling and identification.** The model is defined consistently to other robotics applications, such as robot simulation or MPC in (4.21): Any model can be utilized to simulate q_{sim} , \dot{q}_{sim} and τ_{sim} . Unfortunately, additional engineering effort is necessary for designing a closed-loop model, even if only a single part of the model is identified. As a beneficial side effect, this leads to validation with measurements for every model component. For example, a good inertia identification is impossible without designing the proper friction model and feedback controller. Furthermore, it is not required to design models which are linear in inertial parameters. Moreover, nonlinear and discontinuous models are supported. Additionally, linear bounds can be enforced directly on the physical parameters, ensuring a physically feasible system.
- **Single measurement sufficient.** The optimization problem (4.21) features the advantage that a single measurement - position, velocity, or torque - is sufficient for parameter identification. This is enabled by the closed-loop formulation of the model, which utilizes the reference trajectory (q_R, \dot{q}_R) as the model input variable. The model output variable can be any combination of (including all) position, velocity, and torque measurements. All but one can be omitted in noisy, inaccurate, or unavailable position, velocity, or torque measurements. However, usually, the motor torque MSE has greater dynamic bandwidth than position and velocity MSE, thus containing more information and is recommendable.
- **Arbitrary parameter identification.** The parameters to be identified can be chosen arbitrarily. Parts of the friction, inertia, gravitational loads, feedback controller gains, HWC, or any other component can be identified in a single optimization problem. The choice of the parameter vectors leads to major consequences for the computational requirements of the algorithm. The more parameters are chosen, the more computational resources are necessary. In this work, all 30 friction parameters are identified at once. However, the choice of the identification parameters influ-

ences the condition number of the excitation trajectory for the parameters. It is not beneficial, although it is possible, to employ a single trajectory to identify all model parameters.

4.2.2 Design of Experiments

The sensitivity matrix of a trajectory of a nonlinear dynamic parametric model $\tau = f(q(t), \Theta)$ is denoted by the Jacobian

$$X_0(q(t), \Theta) = \left. \frac{\partial f(q(t), \Theta)}{\partial \Theta} \right|_{\Theta_j = \Theta_{0,j}}, \quad (4.22)$$

which is evaluated at a local point $\Theta_{0,j}$ and is, as a consequence, a local regressor at this point. The continuous parameter subset Θ_j includes all parameters of Θ that remain in X after differentiation. The values $\Theta_{0,j}$ are not known to be the optimal parameter values and are usually chosen to use assumptions. The fundamental problem is encountered that a nonlinear model-based trajectory optimization is employed to identify the model's parameters. At this point, the model's parameters cannot yet be known with certainty because the whole process is intended to identify them. Therefore, multiple points are chosen as the matrix $\Theta_{P,j}$ over a grid inside bounds for every $\Theta_j, \Theta_{j,lb} \leq \Theta_j \leq \Theta_{j,ub}$. This set of local regressors promises to be more informative than a single, arbitrary point. Collecting all the Jacobians for the points $\Theta_{P,j}$ in one matrix leads to $X_P(q(t), \Theta) = [X_1 \dots X_p]^T$.

Let the friction model (4.20) be the parametric nonlinear model to consider. The resulting Jacobian is defined as

$$X_{(P,i)}(q(t), \Theta_i) = [\dot{q}_i(t), \tanh(s_f \dot{q}_i(t)), 1, \tanh(f_b \dot{q}_i(t)), \xi_1, \xi_2] \Big|_{\Theta_{P,j}} \quad (4.23)$$

with the temporary variables $\xi_1 = -f_a \dot{q}_i(t)$ and $\xi_2 = \tanh(f_b \dot{q}_i(t))^2 - 1$. The parameter vector $\Theta_i = [f_{asym,i} f_{v,i} f_{c,i} f_{a,i} f_{b,i}]$ and the matrix $\Theta_{i,P,j} = [f_{(a,i,P)} f_{(b,i,P)}]$ are the collection of the grid values of the remaining parameters $f_{a,i}$ and $f_{b,i}$. Combining the regressors for all joints in one block diagonal matrix $W = \text{diag}(X_1, \dots, X_n)$ delivers a formulation for the condition number criterion for the trajectory of the whole system:

$$\dot{q}^*(t) = \underset{\dot{q}(t)}{\text{argmin}} (\text{cond}(W)) \quad (4.24)$$

$$\begin{aligned} \text{s.t. } \quad q_{lb}^{(D)}(t) \leq q^{(D)}(t) \leq q_{ub}^{(D)}(t) & \quad \forall t \in [0, T], \forall D \in \mathbb{N}_{\leq 2}^0 \\ q^{(D)}(t) = 0, & \quad \forall t \in \{0, T\}, \forall D \in \mathbb{N}_{\leq 4}^0. \end{aligned}$$

The identification is augmented with position, velocity, and acceleration limits, with the lower and upper bounds $q_{lb}^{(D)}(t), q_{ub}^{(D)}(t)$, of derivative order D . The second constraint guarantees that the trajectory up to the 4th derivative starts and ends at zero. By definition, the position is zero in the homing position, as pictured in Fig. 6.1. For identification, the trajectory is usually parameterized and, in most cases, as a Fourier series [SVS07]. Fourier series ensure the 4th-order joint differentiability necessary for the flatness-based control [WGR20]. The resulting formulation is a nonlinear optimization problem and demands global optimization approaches. As in [TUB20] a so-called *Memetic Algorithm* is used to get the optimal trajectory. The trajectories applied to the real robot are depicted in Fig. 4.5.

4.2.3 Experimental Setup

The experiments are carried out on a KUKA KR300 R2500 ultra SE robot as pictured in Fig. 6.1. It is an industrial robot featuring a maximum payload capacity of 300 kg and a reach of 3100 mm. A payload of 150 kg is attached in all measurements. The manipulator is designed by *KUKA* and not modified. The robot control, including the electrical hardware and the real-time operating system, is based on components by industrial supplier *B&B Automation*. This custom-made controller enables full access to low-level controllers and filters. However, the target compiler does not support extended capabilities, such as parallel computation or external software libraries. Consequently, the identification algorithm runs on an external computer communicating with the real-time target. As the real-time capabilities of this algorithm are essential, the algorithms are tested on two external computers. One is a standard *Microsoft Surface* from 2017, featuring *windows 10* and an *Intel(R) Core(TM) i7-7660U CPU @ 2.50GHz* and 8 GB RAM. The second external computer operates on *Linux Mint 20.1* with an *Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz* and 32 GB RAM. In both cases, all algorithms are executed in *MATLAB* and run only on Central Processing Unit (CPU). If not mentioned otherwise, the *Linux* desktop computer is applied.

Three trajectories are applied, as presented in Fig. 4.5. One is applied for robot identification (blue lines), and two more are employed for cross-validation only (red and green lines). All trajectories are created as explained in section 4.2.2. Due to the long lever, the experiments focus on the three main points, although, in all trajectories, all six joints are moved simultaneously so that all joints influence each other. As required for online iden-

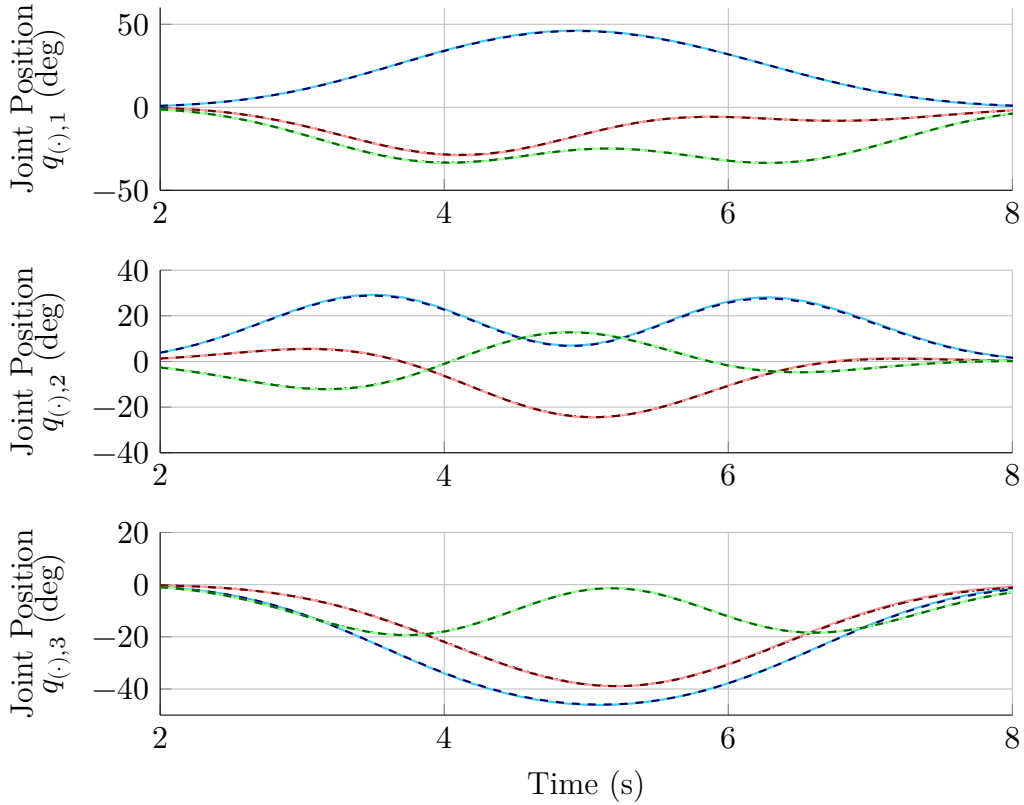


Figure 4.5: Measured and simulated trajectory for joint 1 (top), joint 2 (middle) and joint 3 (bottom). Dotted line: Measured trajectory. Solid line: Simulated trajectory. For clarity, the reference trajectories are not included. The difference between reference, measurement, and simulation is constantly less than ± 0.25 deg and cannot be observed without magnification. Colors correspond to the trajectory and torque figures. Blue: Identification trajectory, Red: 1st validation trajectory, Green: 2nd validation trajectory.

tification, the identification does not start in stand still but starts after 2 s when all joints are in movement. All movements are recorded in 4 ms. Every trajectory is performed only once for the single batch optimization. The presented algorithm does not require multiple repetitions of the measurements to mitigate sensor noise. For the real-time batch optimization, the identification trajectory (blue line) is repeated 30 times consecutively to obtain a total experiment duration of 5 minutes.

4.2.4 Single Batch Identification

Besides the baseline of the trajectories, Fig. 4.5 demonstrates the comparison between position measurements and simulations for a single batch identification. No deviation between model prediction and measurements can be observed for all trajectories without further magnification. The same holds for velocity measurements, which are not shown in this contribution due to space limitations. The raw, unfiltered torque measurements are

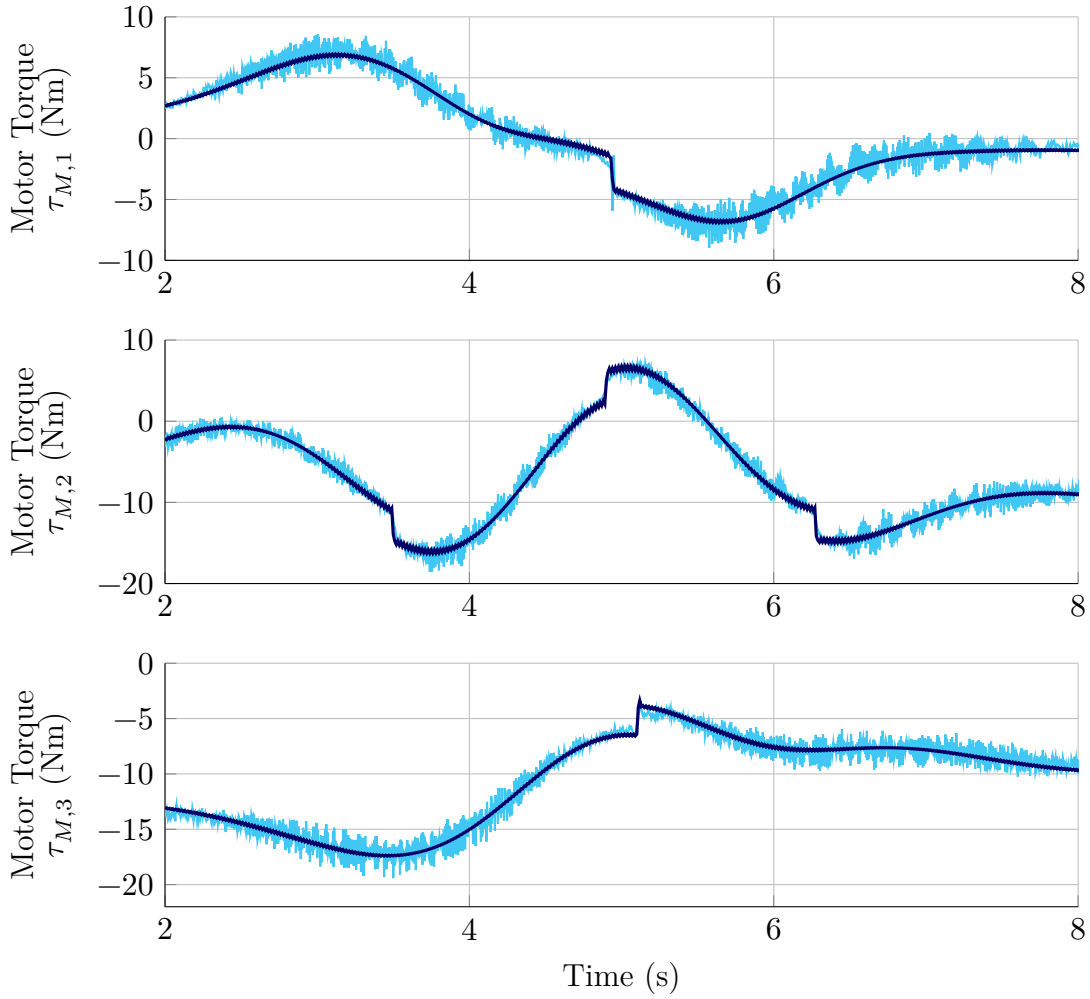


Figure 4.6: Motor torque of identification trajectory for joint 1 (top), joint 2 (middle) and joint 3 (bottom). Light blue: Measured torque. Dark blue: Simulated torque.

used as illustrated in Fig. 4.6 for the identification of the model. Applying raw data leads to a natural regularization.

The main results for all three joints and all three trajectories are shown in Tab. 4.2. To compare the results, the measured motor torque is filtered with a zero-phase digital low pass filter (passband frequency 30 Hz, filter order 8). The simulated torque remains unfiltered in all cases. The RMSE on the difference between the measured and simulated motor torque (*meas-sim*), filtered and simulated torque (*filt-sim*), and filtered and measured torque (*filt-meas*) are applied as indicators. Furthermore, the relative motor torque error is captured,

$$e_{rel} = 100 \cdot \|\tau_{meas} - \tau_{sim}\|_2 / \|\tau_{meas}\|_2 \quad (\text{error-rel}) \quad (4.25)$$

$$e_{rel, filt} = 100 \cdot \|\tau_{filt} - \tau_{sim}\|_2 / \|\tau_{filt}\|_2 \quad (\text{error-rel-filt}) \quad (4.26)$$

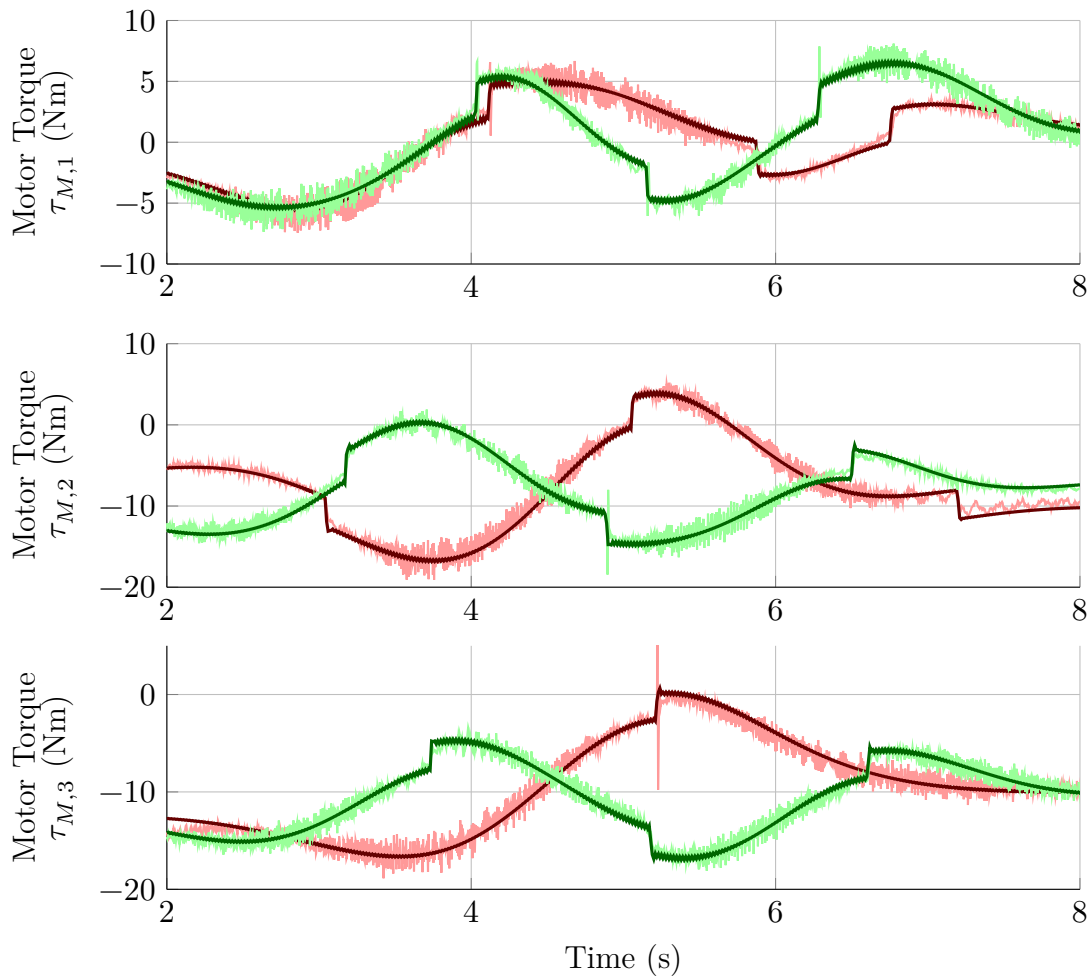


Figure 4.7: Motor torque of validation trajectories for joint 1 (top) joint 2 (middle) and joint 3 (bottom). Light red: Measured torque of 1st validation trajectory. Dark red: Simulated torque of 1st validation trajectory. Light green: Measured torque of 2nd validation trajectory. Dark green: Simulated torque of 2nd validation trajectory.

Table 4.2: RMSE and relative motor torque errors for every joint and for every trajectory. For definitions of indicators and filter settings see the main paragraph.

Joint	Indicator	Ident	1 st Val	2 nd Val	Unit
1	meas-sim	0.63	0.55	0.60	Nm
	filt-meas	0.56	0.45	0.51	Nm
	filt-sim	0.38	0.32	0.34	Nm
	rel-error	15.0	16.2	14.8	%
	rel-error-filt	9.5	10.2	8.9	%
2	meas-sim	0.75	0.71	0.68	Nm
	filt-meas	0.72	0.70	0.71	Nm
	filt-sim	0.61	0.84	0.73	Nm
	rel-error	8.4	7.8	7.5	%
	rel-error-filt	7.1	9.7	8.4	%
3	meas-sim	0.60	0.79	0.60	Nm
	filt-meas	0.80	0.73	1.00	Nm
	filt-sim	0.71	0.71	0.89	Nm
	rel-error	5.3	7.2	5.2	%
	rel-error-filt	6.4	6.8	8.2	%

with the L2-norm and the corresponding torque datasets.

Tab. 4.2 demonstrates that for all joints, every indicator remains the same for all trajectories. Furthermore, the model fit indicators outperform the cross-validation trajectories' identification trajectory. This indicates a good extrapolation capability and no overfitting of the identified parameters.

A relatively high RMSE between the filtered measurement and simulation of ≤ 0.89 Nm is observed. This is based on two reasons: First, the parameters are not identified on the filtered measurements but the raw data. Second, substantial oscillations in the filtered measurement below 30 Hz frequencies are observed, most likely due to elastic deformations of the gearboxes. The noteworthy oscillations in the filtered and unfiltered measurements are reflected in the relative error of 5.2 to 16.2%.

The RMSE on *filt-meas* is an indicator for the noise amplitude in the measured motor torque, comparing the raw motor torque with its filtered equivalent. In all cases, this indicator is between 0.45 and 1.0 Nm. The RMSE on simulation, *meas-sim* and *filt-sim*, is constantly less than the RMSE of the measurement noise of 1.0 Nm. As visualized in Fig. 4.6 and Fig. 4.7, the simulated motor torque stays in all cases within the motor measurement noise. Identified parameters are given in appendix B.4.

4.2.5 Computational Efficiency of Multi Batch Identification

The software is based on *Robotics Toolbox* [Co17]. The pose and payload dependent inertia matrix $M(q)$, centrifugal and Coriolis matrix $C(q, \dot{q})$ and the vector of gravitational torques $g(q)$ are derived as symbolic expressions using a modification of the *CodeGeneration()* function. The closed-loop simulator (2.1), (4.33) is compiled to executable C-code using the *MATLAB Coder Toolbox*, for two operating systems, Windows and Linux. The global surrogate optimizer is given in the *MATLAB Global Optimization Toolbox*. It is a general global nonlinear solver capable of mixed-integer problems, nonlinear constraints (which are not applied in this thesis), and stiff, non-differentiable objective functions. It is intended for extensive, objective functions and creates a surrogate, a radial-basis-function-based approximation of the objective function. Likewise, it is capable of a warm start so that evaluated points from previous computations can be reused. With the surrogate optimizer, the symbolic model, and the compiled code, the algorithm is already so fast that it slows down when utilizing *MATLAB Parallel Computation Toolbox* on one test hardware platform (*Linux* computer). However, parallel computation is applied on the second platform (*Windows* computer).

For the solution of the differential equations (4.1a), (4.1b), the 4-stage Runge-Kutta algorithm is applied. A fixed-time step solver is selected in order to minimize discontinuities in the optimization problem. Furthermore, a fixed-time step solver implies a deterministic computation time, which is important considering real-time systems. The simulation only requires the reference position q_R , the reference velocity \dot{q}_R and the initial conditions q_0, \dot{q}_0 . Suppose only a noisy, inaccurate, or no position or velocity measurement is available. In that case, the initial conditions can be set to the reference trajectory, $q_{R,0}, \dot{q}_{R,0}$, with minimal modeling error. The model calculation does not rely on further measurements than the initial state. Including the feedback controller in the closed-loop model leads to high sampling frequencies in the optimization problem. On the robot, the velocity controller is calculated each 0.2 ms and the position controller each 0.8 ms. The robot model must consider different sampling times of the feedback controller. Unfortunately, high sampling frequencies increase the computational effort. This is especially demanding for updating the inertia, Coriolis, and gravitational components. Therefore, the update of these components is only applied if a sufficient change in position or velocity occurred, e.g., $\|\Delta q\|_2 \geq 2 \text{ deg}$ or $\|\Delta \dot{q}\|_2 \geq 2 \text{ deg/s}$. This method is explained in the appendix E.1.

The identification runs consecutively for 5 minutes to test real-time capabilities. For clarity, the first 3 minutes are zoomed in Fig. 4.8 and in Fig. 4.9. The optimizer utilizes the last completed batch, identifies the parameters, and restarts after the computation is done using the currently completed batch. Discrete circles in Fig. 4.8 and in Fig. 4.9 refer to each completed identification. The optimization variables of the first batch are randomly sampled from the entire parameter space, e.g., $\tilde{\Theta}_{lb} \leq \tilde{\Theta} \leq \tilde{\Theta}_{ub}$. Subsequent

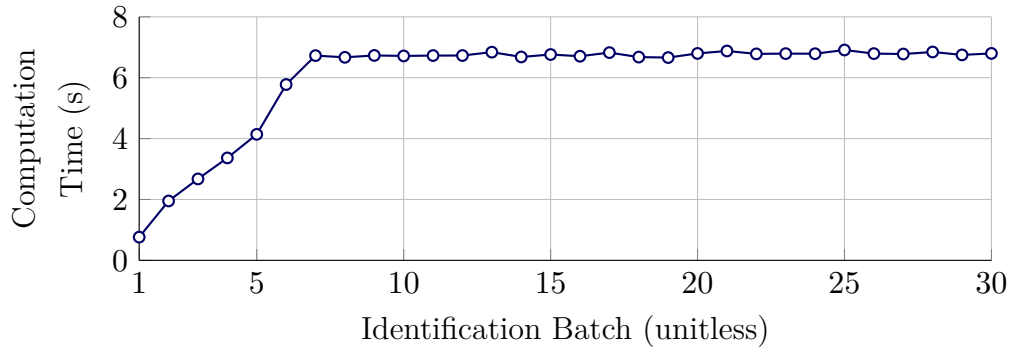


Figure 4.8: Computation time for each identification.

optimization performs a warm start to reuse previously evaluated function points. Fig. 4.8 presents the computation time required for each of the 48 optimizations done during the 5 minutes (figure clipped to 30 batches for visibility). The computation duration of the surrogate solver depends on the number of evaluation points per batch, e.g., 50, the maximum amount of evaluation points reused from previous iterations, e.g., 300, and the implementation of the cost function. Fig. 4.8 shows that the solver applies an increasing number of warm start points. Therefore, the solver slows down until the buffer size is fixed. A maximum evaluation time for the full robot model, including the 30 optimization variables, is achieved in less than 7s in all cases.

Fig. 4.9 presents the results for each batch identification. The RMSE motor torque is after 18.7s within the range of the torque measurement noise of 0.7Nm. The cost function values provide insight into the stable identification process. Except for the first 6 batches, the identification results of motor torque, link position, and link velocity using the real-time batch mode are similar to the single batch results presented in Fig. 4.5 and Fig. 4.6.

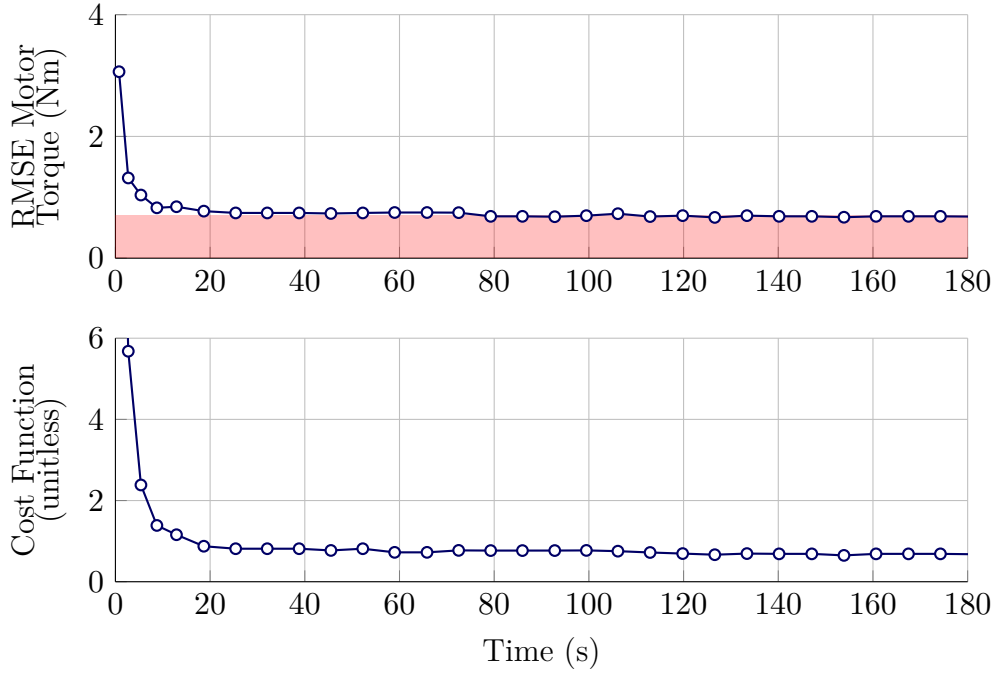


Figure 4.9: Identification results for the RMSE error of the motor torques of all joints (top), and the cost function (bottom). The light red background represents the motor torque noise level (top).

4.3 Robot Control

4.3.1 Control Structure

A model-based feed-forward controller to account for the dynamic system and a feedback control law to compensate for unknown disturbances and model errors are utilized. In Fig. 4.10 and Fig. 4.11 two different control architectures are presented. Both contain a feed-forward controller, a feedback controller, and trajectory planning.

The trajectory planning generates the reference signals, e.g., the reference link angle q_R of the desired trajectory and its derivatives. A motor torque τ_M is applied to the robot, and the link angle q and the motor velocity $\dot{\theta}$ are the control variables of the feedback controller. For the first three joints, the link angle q is measured directly using Secondary Encoders (SE) which are mounted on the hull of the robot structure. The motor velocity $\dot{\theta}$ is independently measured using the resolver mounted on the motor shaft. For both sensors, noise is addressed by implementing low-pass filters.

Regarding the feed-forward controller, there are two different architectures, as presented in Fig. 4.10 and Fig. 4.11. The design in Fig. 4.10 is referred to as the flatness-based feed-forward controller, whereas the structure in Fig. 4.11 is known as exact linearization. Depending on the literature, both are referred to as Computed Torque Control (CTC) [MH08; NSP08]. The flatness-based design is explained in detail in section 4.3.3 and computes the feed-forward torque independently of the current states. It relies exclusively on

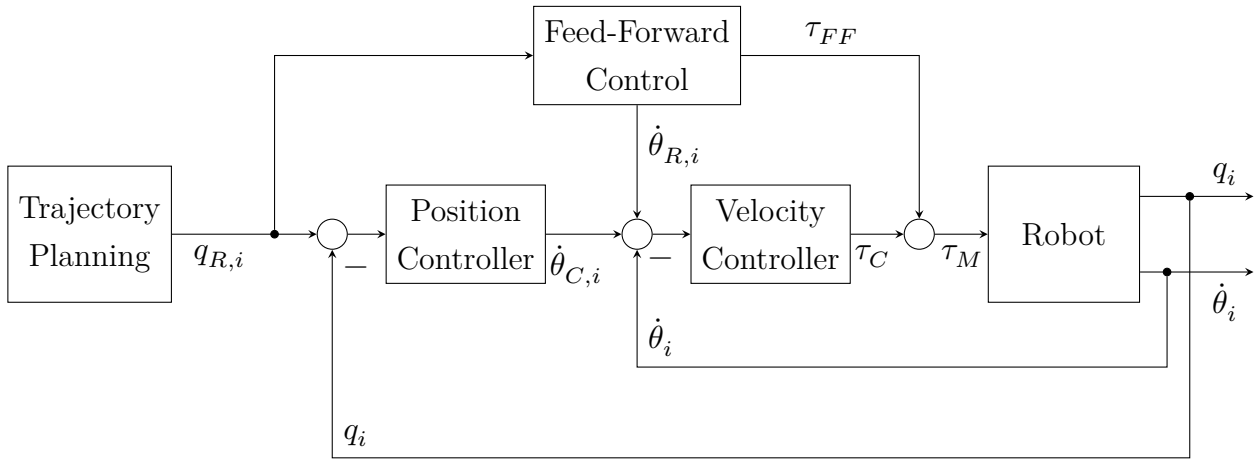


Figure 4.10: Controller design with feed-forward and feedback control.

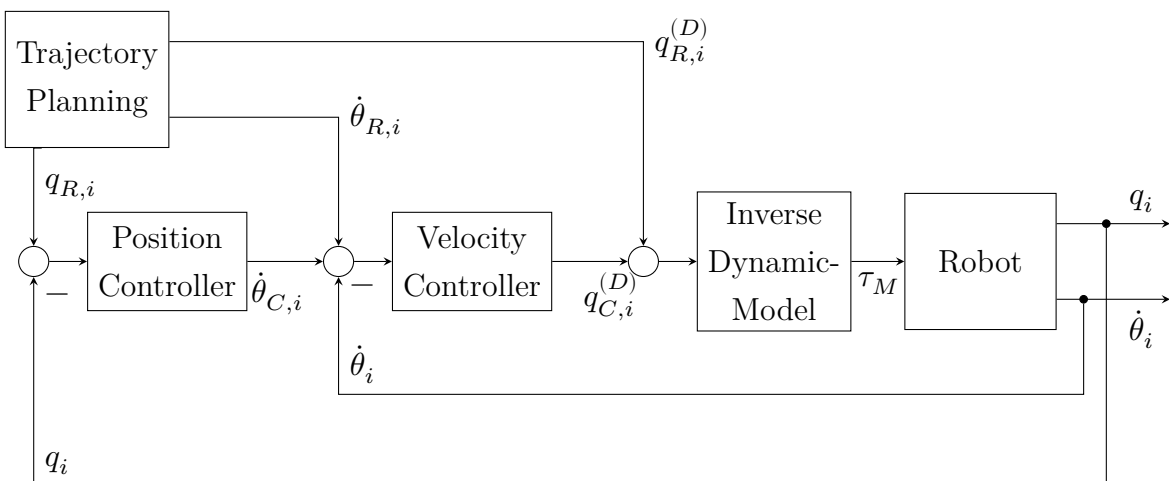


Figure 4.11: Controller design with exact linearization.

the reference trajectory and its derivatives, and therefore, it can potentially be computed offline. As a consequence, it is invulnerable to any measurement noise. The flatness-based architecture is independent of the feedback control. Consequently, the feedback control is not time delayed or modified. It contains the disadvantage of a potentially less precise model since it relies on the reference trajectory rather than the measured trajectory. For the robot demonstrator used in this work, the joint precision is less than 0.1 deg for all joints (see the final results in section 6.3.1) and therefore the error, using reference rather than measured joint angles, is negligible. The robot accuracy improvements gained by using a noise-free 4th-order differentiable trajectory prevail. Additionally and equally important, the feedback control can be faster computed if the feedback loop does not contain the dynamic model. However, for less accurate robot types, it is possible and useful to include online measurements to improve the model's accuracy partially. The inertia matrix, gravity, Coriolis, and centripetal terms can be updated at run time based on online measurements.

The exact linearization architecture, as presented in Fig. 4.11, attempts to eliminate the nonlinearity in robot dynamics in an inner loop and therefore control a simple, linear system with the feedback controller. A detailed explanation is given in [Sp87]. The feedback controller computes an angular acceleration for a dynamic inversion-based controller in order to compensate for nonlinear dynamics.

Its advantage is that the feedback controller is automatically adjusted to the current robot state. Disadvantages are a necessity of a fast computation of the inverse dynamic model in the inner loop and a potential time delay of the feedback control. In [NSP08], both designs are compared, and the flatness-based architecture outperforms the exact linearization structure with several feed-forward models regarding the achieved trajectory accuracy. The authors in [NSP08] argue that the main advantage of a feed-forward structure compared to exact linearization is the direct and non-delayed impact of the feedback control law. Based on these arguments, a flatness-based architecture rather than exact linearization is applied.

The exact linearization module does not account for joint elasticity; therefore, the inner loop reduces to a double integrator. The main idea of applying the model inverse remains the same in the flatness-based feed-forward and the exact linearization case. Assume the following form gives a model inverse of the robot

$$\tau_M = f_{FF}(q_U, \dot{q}_U, \dots, q_U^{(D)}) \quad (4.27)$$

with an input angle q_U and its derivatives. The derivative is defined as $q^{(D)} = \frac{d^D q}{dt^D}$. In the flatness-based architecture $q_U = q_R$. In the exact linearization formulation

$$q_U^{(D)} = q_R^{(D)} + q_C^{(D)} \quad (4.28)$$

with the feedback control variable $q_C^{(D)}$. With an elastic joint model, the derivative order is $D = 4$, whereas with the rigid joint model, the derivative order is $D = 2$ [MH08]. So, a rigid joint model is

$$\ddot{q}_U = \ddot{q}_R + \ddot{q}_C, \quad (4.29)$$

and a model with elastic joints is

$$q_U^{(4)} = q_R^{(4)} + q_C^{(4)}. \quad (4.30)$$

In the exact linearization case, applying the highest input derivative is beneficial. Note that the output of the velocity controller u_C depends on the architecture. For the flatness-based, it is equal to the motor control torque, e.g., $u_C = \tau_C$. In the exact linearization case, it computes the control input of the inverse dynamics model, e.g., $u_C = q_C^{(D)}$. In section 4.3.3, the flatness-based module is derived and thus (4.27) is estimated.

4.3.2 Feedback Controller using Secondary Encoders

A PD-controller with a link-side position controller using SE and a motor-side velocity controller based on the resolver signal is implemented. Considering elastic joints, it is difficult to precisely calculate the motor reference signals since the equation $\theta_R = u q_R$ is only valid for rigid joints. A precise calculation of the motor reference signals requires the full, nonlinear stiffness model (4.1a) and (4.1b). Therefore, the exact calculation of the motor reference signals is sensitive to modeling errors. In most previous works, this consideration is simply neglected, and the rigid link equation $\theta_R = u q_R$ is used for the motor-side position and velocity controller. Regarding position control, this problem can be solved by measuring q and implementing a link-side position control. Unfortunately, a link-side velocity control only partially solves this issue. A link velocity speed control utilizes the correct reference speed \dot{q}_R , but it causes two additional issues. First, a comparatively low sensor resolution due to the missing transmission factor requires significantly low pass filtering. Second, a soft velocity control parameter is necessary to avoid stability issues caused by the elastic joint. Both drawbacks can be addressed by implementing a motor-side velocity controller utilizing the full nonlinear elastic joint model for calculating the correct motor reference speed. The significance of this effect is presented in real world experiments.

Both the position and velocity controllers are realized as proportional controllers. The position controller is given by

$$\dot{\theta}_C = K_P(q_R - q) \quad (4.31)$$

with the proportional position gain K_P and the motor velocity $\dot{\theta}_C$. The velocity controller using the proportional speed gain K_V has the form

$$\tau_C = K_V(\dot{\theta}_C + \dot{\theta}_R - \dot{\theta}). \quad (4.32)$$

All in all, the control signal sums up to

$$u_{vel} = \tau_{FF} + \tau_C. \quad (4.33)$$

4.3.3 Flatness-based Feed-Forward Controller

The main task of the feed-forward control law is to achieve a good tracking behavior, whereby the feedback control law can be applied exclusively for compensating disturbances and model uncertainties. The inverse nonlinear model of the robot link is used to design the feed-forward control law. Therefore, a solution of the nonlinear model (4.1a) and (4.1b) for the motor torque τ_M is required,

$$\tau_M = J\ddot{\theta} + U^{-1}(M_L(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) + \tau_F(\dot{q}) + \tau_H(q)). \quad (4.34)$$

The following assumptions are made to estimate (4.34).

- The motor inertia matrix J is diagonal.
- All parameter changes during one interval are neglected, in the presented case for 0.8 ms. The parameters are updated each 0.8 ms according to the state trajectory but are not considered as differentiable variables in the flatness-based feed-forward control law.
- During each time interval, the dependency of the inertia matrix $M(q, \dot{q})$ on the position q and velocity \dot{q} is neglected.
- During each time interval, the dependency of the Coriolis and centripetal matrix $C(q, \dot{q})$ on the position q and velocity \dot{q} is neglected.
- During each time interval, the dependency of the gravitational load $g(q)$ on the position q is neglected.

- During each time interval, the dependency of the hydraulic load $\tau_H(q)$ on the position q is neglected.

Using the assumptions, (4.34) turns into

$$\tau_M = J\ddot{\theta} + U^{-1} (M_L\ddot{q} + C\dot{q} + g + \tau_F(\dot{q}) + \tau_H). \quad (4.35)$$

For good tracking behavior, e.g., $q = q_R$, the motor position is set to the motor reference position, e.g., $\theta = \theta_R$ and regarding the motor torque $\tau_M = \tau_{FF}$ is applied. It follows that

$$\tau_{FF} = J\ddot{\theta}_R + U^{-1} (M_L\ddot{q}_R + C\dot{q}_R + g + \tau_F(\dot{q}_R) + \tau_H). \quad (4.36)$$

The continuous differentiable friction torque $\tau_F(\dot{q}_R)$ is defined in (4.20). In order to solve (4.36), a substitution of $\ddot{\theta}_R$ with a function of q_R and its derivatives is required, which is given by the inverse nonlinear stiffness model. The torsion angle Δq as defined in (4.3) is utilized for the motor position,

$$\theta = U_R (\Delta q_R + q_R). \quad (4.37)$$

The torsion angle Δq_R is substituted using the inverse, continuously differentiable stiffness model (4.14) leading to

$$\theta_R = U \left(\frac{\tau_E}{c_{TR}} + \frac{2\phi_B}{1 + e^{-s_E \tau_E}} - \phi_B + q_R \right). \quad (4.38)$$

The division operator and the exponential function are defined element-wise in (4.38) and (4.39). The elastic torque τ_E is substituted using the sum of torques of the link side. This leads to

$$\begin{aligned} \theta_R = U & \left(\frac{M_L\ddot{q}_R + C\dot{q}_R + g + \tau_F(\dot{q}_R) + \tau_H}{c_{TR}} \right. \\ & \left. + \frac{2\phi_B}{1 + e^{-s_E (M_L\ddot{q}_R + C\dot{q}_R + g + \tau_F(\dot{q}_R) + \tau_H)}} - \phi_B + q_R \right). \end{aligned} \quad (4.39)$$

Obtaining the second derivative of (4.39), e.g., $\ddot{\theta}_R = \frac{d^2\theta_R}{dt^2}$, completes the solution for τ_{FF} in (4.36). The solution is too long to display as a compact equation. The source code containing the solution is given in E.2 and E.3.

Source code E.2 presents the feed-forward controller with the advanced friction model in the *MATLAB* programming language. The code is presented as created by the *MATLAB*

Symbolic Toolbox. As the toolbox can only solve for a single joint, source code E.2 must be extended for all joints before application. The code has been translated into *Python* (for the machine learning toolbox) and *C* (for the real-time robot control). The *C* code is given in code E.3.

Some minor modifications differ in code E.2 (*MATLAB*) and code E.3 (*C*). First, code E.2 applies the advanced nonlinear friction model (4.20) whereas code E.3 features the nonlinear friction model (2.3). Second, the $\exp(\cdot)$ functions are modified in code E.2 to formulations using $\tanh(\cdot)$, so that Not-A-Number issues can be handled directly without using a $\max(\cdot)$ function.

4.4 Originality and Background

To clarify intellectual property, background information is given in this section. This section is exclusively for background information and can be skipped for all readers only interested in the methods and results.

Besides the content presented in previous publications, the author wants to thank everyone who contributed to the industrial robot. These persons developed, implemented, and tested the hardware and the software, without which this work would have been impossible. Fig. 4.12 presents the robot in 2017 when this project started.

The extent of the work on the robot controller is reflected in the scope of its documentation. Full documentation of the controller would exceed the scope of this thesis. The major hardware components, main electrical concept, software architecture, kinematic robot parameters, and dynamic model parameters are captured in appendix B. Additionally, please refer to the 175 page software documentation given in [Ha22a]. Note that [Ha22a] only contains the final state of the robot control. It does not contain detailed explanations on the background of software design choices nor statements on developments not included in the version as of 2022. For the hardware documentation, please refer to [Gö19; Gö20] with 100 and 99 pages respectively.

The physical robot chapter 4 built up on [We22; WGR20]. In [WGR20], the robot model is derived and tested in simulation and on the real robot. Nigora Gafur contributed to the experiments, participated in the discussions, and wrote the state-of-the-art section. Jonas Weigand developed the model and control and is responsible for the writing of [WGR20]. The final robot model is unpublished work and was developed by Jonas Weigand, Jonas Ulmen, and Alexandre Janot. In particular, the parameter estimation based on the condition optimization criterion origins from Jonas Ulmen. Alexandre Janot and Martin Ruskowski served as supervisors and helped in all discussions. Finally, the model is validated on the public robot benchmark data presented in [We22]. The benchmark data was created by



Figure 4.12: Picture of the industrial robots in 2017 at the Chair of Machine Tools and Control Systems, RPTU Kaiserslautern-Landau.

Jonas Weigand, Jonas Ulmen, Julian Götz and Martin Ruskowski.

5 Continuous-Time Neural Networks

In this chapter, advances for continuous-time NN are presented. As the methods are capable of improvements beyond inverse models for six-joint industrial robots, additional real world benchmarks are introduced. The benchmarks are publicly available and the results of the presented methods can be directly compared to system identification methods beyond NN in literature.

In section 5.1, State Derivative Normalization (SDN) is introduced. The importance of data normalization in the value domain, e.g., using Min-Max scaling, is well known. For continuous-time models it is important, too, to normalize the hidden state and hidden state derivative. As this method improves continuous-time NN beyond the robot application, it is demonstrated on the public Cascaded Tank System (CTS) benchmark.

In section 5.2, model stability is discussed. Stability constraints can apply the knowledge from system identification to continuous-time NN. This is important for safety-critical tasks and in addition, it improves the long-term prediction accuracy of continuous-time models. The method enhances continuous-time NN beyond the robot use-case and is validated on the public Electro Mechanical Positioning System (EMPS) benchmark.

In section 5.3, memory efficiency is analyzed. Based on empirical evidence from the real robot, a direct comparison of discrete-time and continuous-time NN is presented. This demonstrates the superior generalization capabilities of continuous-time NN for robot identification.

5.1 State Derivative Normalization

The core idea of normalization is to map the input and output data in a numerically favorable range [BKH16; Og10; SS97]. That is, (I) to ensure normally distributed data for the parameter initialization (e.g., Xavier initialization [GB10]), (II) to scale all input and output channels in the same numerical range, and (III) to numerically improve the scope of the gradients for the nonlinear activation functions.

SDN can significantly improve the model performance. One novel promising solution

method is the introduction of a single normalization factor τ , which augments the state derivative equation

$$\frac{dx(t)}{dt} = \frac{1}{\tau} f_{NN}(x(t), u(t)). \quad (5.1)$$

5.1.1 Normalization Definition

One aspect in which dynamical systems can differ is their inherent scale of time. Normalizing means eliminating this aspect when dealing with different dynamical systems. A time constant of a linear system usually describes the decay rate, e.g., after an excitation. These constants refer to the decay rate of each dynamic mode, while the slowest mode determines the overall decay rate. Mathematically, the overall time constant of the linear system is the inverse of the largest eigenvalue of its system matrix.

For a general nonlinear system of the form

$$\dot{x} = f(x, t), \quad (5.2)$$

this analysis is not so straightforward. However, Contraction analysis [Lo99; LS97] concludes that such a time constant exists for the differential dynamics of the nonlinear system. This section concludes with three different analytic interpretations of the SDN (5.1), linking the state, the state derivative, and the time domain. Furthermore, the role of the explicit ODE solver is discussed and graphic insight into the method is given.

A tangent form or respectively the differential dynamics of a nonlinear system are given by [Lo99]

$$\delta\dot{x} = \frac{\partial f(x, t)}{\partial x} \delta x. \quad (5.3)$$

The Jacobian $\frac{\partial f(x, t)}{\partial x}$ is the linearization of the system, not at an operating point but everywhere. Now consider two trajectories of the system that originate from two different initial conditions. The distance between the two trajectories is denoted by δx . The squared distance between two trajectories can be defined by

$$\|\delta x\|_2^2 = \delta x^T \delta x. \quad (5.4)$$

This squared can be tracked over a distance over time to get the rate of change over time,

$$\frac{d}{dt}(\delta x^T \delta x) = 2\delta x^T \delta \dot{x} = 2\delta x^T \frac{\partial f(x, t)}{\partial x} \delta x. \quad (5.5)$$

If the Jacobian $\frac{\partial f(x, t)}{\partial x}$ is negative definite, the system is contracting, while the eigenvalues of its symmetric part $\frac{1}{2}(\frac{\partial f(x, t)}{\partial x} + \frac{\partial f(x, t)}{\partial x}^T)$ describe how fast any two trajectories of the system converge to each other.

The overall contraction rate is also bounded by the largest of these eigenvalues, which leads to

$$\frac{d}{dt}(\delta x^T \delta x) \leq 2\lambda_{\max} \delta x^T \delta x \quad (5.6)$$

$$\frac{1}{2} \frac{d}{dt}(\delta x^T \delta x) \leq \lambda_{\max} = \frac{1}{\tau}, \quad (5.7)$$

where τ is the time constant of the differential dynamics. Modifying (5.7) enables to choose $\hat{\tau}$ as an upper bound for the maximum dynamic constant

$$\tau = \frac{1}{\lambda_{\max}} \leq \hat{\tau}. \quad (5.8)$$

In the next section, it is shown that a constant is not necessarily interpreted as a measure of time. Yet, it is more universal.

State Derivative Normalization (SDN). A strictly positive, linear transformation without offset is introduced in (5.1). Any invertible nonlinear transformation is possible in principle, yet a strictly positive transformation is sufficient and remains physically plausible. In addition, a linear transformation can be used easily for inference and does not require a defined time grid in advance. Clearly, (5.1) compared to (3.1) scales the hidden state derivative. The linear transformation can also be interpreted as changing the weight initialization of the output layer of $f_{NN}(\cdot, \cdot)$.

The normalization can be implemented as a scalar normalization $\tau \in \mathbb{R}_{>0}$ or as a vector corresponding to each channel of the hidden state network, $\tau \in \mathbb{R}_{>0}^{N_X}$. In the latter case, the division and multiplication in (5.1) are defined element-wise.

Time Normalization. (5.1) can be rewritten as

$$\frac{dx(t)}{d(t/\tau)} = f_{NN}(x(t), u(t)) \quad (5.9a)$$

$$\tilde{t} = t/\tau \quad (5.9b)$$

$$\frac{dx(\tilde{t}\tau)}{d\tilde{t}} = f_{NN}(x(\tilde{t}\tau), u(\tilde{t}\tau)), \quad (5.9c)$$

which suggests that the normalization can be viewed as normalization of the time by factor τ .

The numerical discretization scheme reflects this interpretation as time normalization. The model integration step h_{model} is closely related to τ . The general S -stage explicit ODE solver can be defined by the Butcher Tableau [Bu16] which results in the discrete-time system

$$z_{k,s} = f_{NN} \left(x_k + h_{model} \sum_{j=1}^S a_{s,j} z_{k,j}, u_k \right) \quad (5.10a)$$

$$x_{k+1} = x_k + h_{model} \sum_{s=1}^S b_s z_{k,s}. \quad (5.10b)$$

Every explicit ODE scheme multiplies the network output $z_{k,s}$ with the integration step $h_{model} \in \mathbb{R}_{>0}$ (5.10). Normalizing with $1/\tau$ is equivalent to transforming h_{model} . The method applies to both fixed-step solvers and adaptive-step solvers. For fixed-step solvers, $h_{model} = h_{data}/\tau$ effectively normalizes in the time domain. For adaptive-step solvers, $h_{model,k} = h_{data,k}/\tau$ can be applied for each step.

Graphical intuition about the time normalization can be gained from output data of the CTS benchmark in Fig. 5.1 [Sc16b]. The measured data (black) of the water level is given in a sample rate of $T_s = 4.0$ s, the normalized data (red) is transferred to a sample time of $T_m = T_s/\tau = 1.0$, the number of time steps remains unchanged. The original data is transferred to the normalized time domain, where the model is trained and evaluated, and the results are transferred back to the original time grid.

State Normalization. It is also possible to rewrite (5.1) as

$$\frac{d(\tau x(t))}{dt} = f_{NN}(x(t), u(t)) \quad (5.11a)$$

$$\tilde{x}(t) = \tau x(t) \quad (5.11b)$$

$$\frac{d(\tilde{x}(t))}{dt} = f_{NN}(\tilde{x}(t)/\tau, u(t)). \quad (5.11c)$$

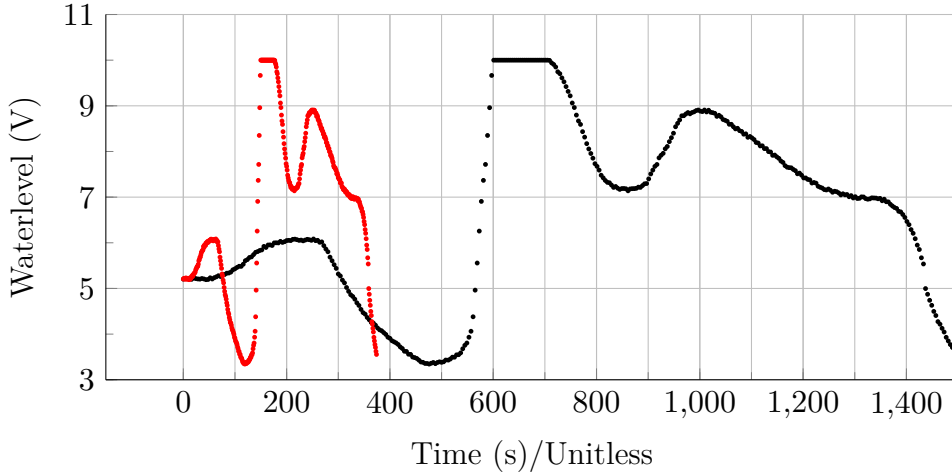


Figure 5.1: Time normalization of the output measurement of the Cascaded Tank System (CTS) benchmark [Sc16b]. Black: Original measurement (unit seconds). Red: Time domain normalized data (unitless).

In this way, normalization can be interpreted as state normalization.

The key takeaway of representations (5.1), (5.9) and (5.11) is that the normalization acts on all three domains simultaneously.

5.1.2 Normalization Factor Estimation

The tendencies of large and small normalization factors to define a proper normalization factor are discussed in the following. Considering continuous-time models *without* hidden state derivative normalization, the smaller the integration step h_{model} , the more accurate is the ODE solver, but also the more steps and the more computational resources are required. The normalization is depicted in Fig. 5.1. Using normalization, the number of integration steps K gets independent of the integration step h_{model} . Therefore, the ODE solver accuracy can be increased *without* additional computation steps. In its limit, very small integration steps will introduce large and stiff hidden state derivatives, which are challenging to estimate.

A good normalization factor can be defined by setting the variance of both, the hidden state $\text{var}(\bar{x})$ and the hidden state derivative $\text{var}(\dot{\bar{x}})$, in a proper range. The notation \bar{x} , $\dot{\bar{x}}$ refers to the hidden state (derivative) matrix with all state (derivative) vectors x_k , \dot{x}_k

over time¹.

Previous work [BST22; BST23] argued that a proper variance of the hidden state and hidden state derivative is one. Using the argument from section 5.1.1, that all three domains are inherently coupled, a formulation of a desirable normalization factor can be obtained,

$$\text{var}(\bar{x}) = \text{var}(f_{NN}(\bar{x}, \bar{u})) \quad (5.12a)$$

$$\text{var}(\bar{x}) = \tau^2 \text{var}(\dot{\bar{x}}). \quad (5.12b)$$

The estimation of the normalization factor τ in (5.12b) is analyzed in the following. The difficulty is that the normalization factor depends (I) on the hidden state and (II) the NN parameters, which are not known apriori.

- Make the normalization factor a trainable (hyper) parameter,
- estimate using cross-validation, or
- use an analytic approach based on the Best Linear Approximation BLA [ST20; VD94].

The first option is highly flexible concerning different data sets and NN structures. It should be ensured for physical plausibility that $\tau > 0$ holds. As most NN libraries do not provide constrained optimization [Pa19], one can implement

$$\hat{\tau} = \epsilon + \text{relu}(\tau) \quad (5.13)$$

with a small positive number $\epsilon \in \mathbb{R}_{>0}$ and the Rectified Linear Function (ReLU). Empirical pre-examinations revealed that the trainable normalization quickly reaches a stationary value. Therefore, it can be trained together with the network in the same loss function and does not require to be fitted as a hyperparameter.

The second option, data-driven estimation with cross-validation, utilizes a grid search to find a suitable normalization factor for a given data set. Results are presented in Fig. 5.3.

A third option is an analytical approach based on the BLA. Estimating a BLA of a potentially non-linear system is commonly done employing Prediction Error Minimization (PEM) with, for instance, a subspace identification approach using the N4SID

1

Note that this work to the variance over time, not the variance over the hidden state (derivative) channels.

Considering the dimensionality, $\text{var}(\bar{x})$ has the unit of (unit state)² and $\text{var}(\dot{\bar{x}})$ has the unit of (unit state)²/(unit time)². The normalization factor must hold for any unit of state and time. As a result, the unit of τ is time.

method [VD94]. Substituting the NN with the BLA in (5.12b), the unknown hidden state \bar{x} can be canceled out. This method is especially recommended for dynamic systems that can be reasonably represented by a linear model. As an additional advantage besides normalization, the BLA can be used as good NN weight initialization [ST20].

The BLA can be simulated in the time domain, obtaining \bar{x}_{BLA} and $\dot{\bar{x}}_{BLA}$. Rearranging (5.12b) leads to

$$\tau_{BLA} = \sqrt{\frac{\text{var}(\bar{x}_{BLA})}{\text{var}(\dot{\bar{x}}_{BLA})}} \quad (5.14)$$

for each component of \bar{x}_{BLA} and $\dot{\bar{x}}_{BLA}$. Optionally, estimating the mean can reduce the vector normalization to an approximate scalar normalization.

An alternative to simulate the BLA over time is to utilize the Discrete Fourier Transform (DFT). Considering that the input has a periodicity of L , the input can be composed into its Fourier components as

$$u(t) = \sum_{m=-\infty}^{\infty} U_m (e^{j\omega_m(t+\phi_m^u)} + e^{-j\omega_m(t+\phi_m^u)}) \quad (5.15a)$$

$$U_m = \left| \frac{1}{L} \int_0^L u(t) e^{-j\omega_m t} dt \right|; \phi_m^u = \arg\left(\int_0^L u(t) e^{-j\omega_m t} dt \right) \quad (5.15b)$$

$$x(t) = \sum_{m=-\infty}^{\infty} X_m (e^{j\omega_m(t+\phi_m^x)} + e^{-j\omega_m(t+\phi_m^x)}) \quad (5.15c)$$

$$X_m = \left| \frac{1}{L} \int_0^L x(t) e^{-j\omega_m t} dt \right|; \phi_m^x = \arg\left(\int_0^L x(t) e^{-j\omega_m t} dt \right) \quad (5.15d)$$

with $\omega_m = m2\pi/L$. Both the variance of the state and state-derivative can be expressed using DFT components as

$$\text{var}(x) = \frac{1}{L} \int_0^L \|x(t)\|_2^2 dt \sim \sum_{m=-\infty}^{\infty} \|X_m\|_2^2 \quad (5.16a)$$

$$\text{var}(\dot{x}) = \frac{1}{L} \int_0^L \|\dot{x}(t)\|_2^2 dt \sim \sum_{m=-\infty}^{\infty} \omega_m^2 \|X_m\|_2^2. \quad (5.16b)$$

The BLA the Fourier components of the state can be written in terms of the transfer function as $X_m = |G(j\omega_m)|U_m$ and $\phi_m^x = \phi_m^u + \arg(G(j\omega_m))$, assuming a single input system for simplicity. Hence, substituting this and 5.16 into 5.12b results in

$$\tau_{BLA} = \sqrt{\frac{\sum_{m=-\infty}^{\infty} U_m^T G(j\omega_m)^T G(j\omega_m) U_m}{\sum_{m=-\infty}^{\infty} \omega_m^2 U_m^T G(j\omega_m)^T G(j\omega_m) U_m}}. \quad (5.17)$$

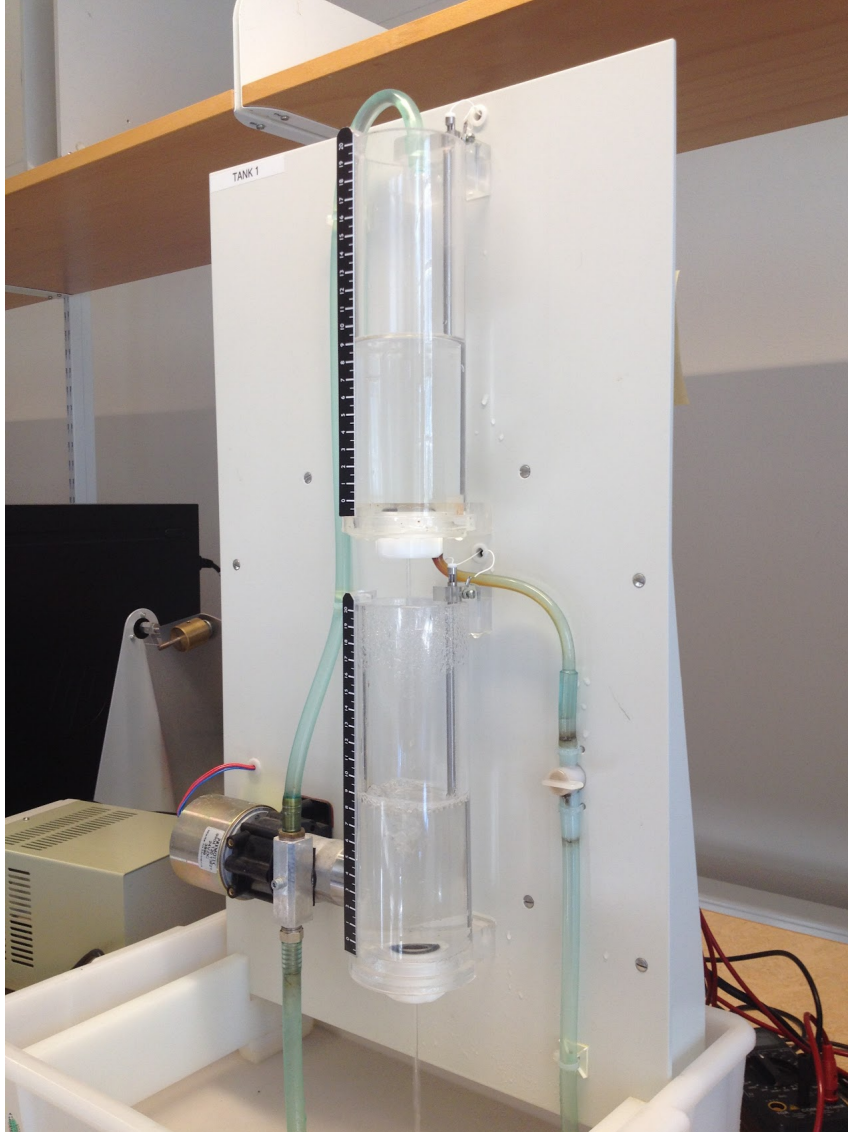


Figure 5.2: Picture of the Cascaded Tank System (CTS) [Sc16b].

This expression allows to derive some properties of the optimal τ_{BLA} . For example, a single sine wave (e.g., only one nonzero U_m) as input signal $u(t)$ results in $\tau_{BLA} = 1/\omega_m$. Furthermore, if $u(t)$ or $G(j\omega_m)$ has a finite bandwidth bounded by Ω than $\tau_{BLA} \geq 1/\Omega$.

5.1.3 Empirical Normalization Results

Benchmark. The proposed methods are applied to the CTS benchmark [Sc16b]. The setup is depicted in Fig. 5.2. It consists of two vertically mounted tanks, where the upper one is filled using a pump, and the water flows from the upper tank into the lower one. The task is to estimate the water level of the lower tank given a pump input sequence. The experiment incorporates an overflow of the tank, which introduces a hard saturation function.

Model Configuration. The fixed-step Runge-Kutta 4 ODE solver [Co84] is chosen. The

network $f_{NN}(\cdot, \cdot)$ consists of linear layers with matrices $A \in \mathbb{R}^{N_x \times N_x}$, $B \in \mathbb{R}^{N_x \times N_U}$ and two residual hidden layers with a Leaky ReLU activation function $\sigma(\cdot)$ and 64 hidden units for each layer. Weight matrices are denoted $W_{(\cdot)}$ and bias terms are $b_{(\cdot)}$ with appropriate dimensions. Bias terms for the linear layer are disabled to encourage model stationarity. An output network $g_{NN}(\cdot, \cdot)$ is applied with the same structure as the state network $f_{NN}(\cdot, \cdot)$. The overall architecture is

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t) \quad (5.18a)$$

$$+ W_{F1} \sigma \left(W_{F2} \sigma \left(W_{F3} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} + b_{F3} \right) + b_{F2} \right) \quad (5.18b)$$

$$y(t) = Cx(t) + Du(t)$$

$$+ W_{G1} \sigma \left(W_{G2} \sigma \left(W_{G3} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} + b_{G3} \right) + b_{G2} \right).$$

Input and output data are z-score normalized. Batch-Norm or Dropout are not applied. The implementation is written in Python, using *PyTorch* [Pa19]. The ADAM optimizer [KB14] is applied with unmodified configuration except for the learning rate, which is set to 0.003 for the first 1000 steps, to 0.0009 until step 3000, and to 0.00027 for all subsequent iterations. The maximum number of optimization steps is 20000. Regularization is obtained using weight decay of $\lambda_W = 10^{-8}$ and early stopping when the best validation error does not improve for 2000 iterations. As no explicit validation data set is given for CTS, the first 512 time steps of the test data are utilized for early stopping. Test data is not accessed for any other reason. Training is performed in 64 mini-batches with a sequence length of $J = 128$ steps using TSEM [FP21]. White box modeling of the CTS would lead to 2 states [Sc16b]. State augmentation, which is also called immersion, is applied with 4 additional states [DDT19]. The initial hidden states are obtained using a Deep Encoder Network [BST22] $e_{NN}(\cdot, \cdot)$ with $n_a = n_b = 5$. Weight matrices are initialized with a small random number $\mathcal{U}[-0.01, 0.01]$, and bias terms are initialized to zero. A barrier function L_S is applied to the linear layer A of the state network to encourage model stability [WDR21]. The barrier function ensures that all eigenvalues of the linear layer are negative, estimated using the differentiable Sylvester Criterion [Gi17]. Besides stability, this barrier minimizes drifts over long simulation horizons. Furthermore, a DAE network $d_{NN}(\cdot, \cdot)$ is utilized to account for the hard state saturation, which is trained using an additional penalty function L_D . It does not affect the forward model evaluation. The optimization problem is given by

$$\begin{aligned}
\min_{\theta} \quad & L_S + L_W + \sum_{n=\max(n_a, n_b)}^{N-J} \left(\sum_{j=0}^{J-1} \|y_{n+j} - \hat{y}_{n+j|n}\|_2^2 + L_{D,j} \right) \\
\text{s.t.} \quad & \hat{y}_{n+j|n} = g_{NN}(x_{n+j|n}) \\
& x_{n+j+1|n} = \text{ODE_Solve} \left(\frac{1}{\tau} f_{NN}(x_{n+j|n}, u_{n+j}), T_s \right) \\
& x_{n|n} = e_{NN}(u_{n-1}, \dots, u_{n-n_b}, y_{n-1}, \dots, y_{n-n_a}) \\
& L_{D,j} = \lambda_D \left(d_{NN}(x_{j|j}, u_{j|j}) \right)^2 \\
& L_S = \begin{cases} 0 & \text{if } A < 0 \\ \lambda_S & \text{otherwise} \end{cases} \\
& L_W = \lambda_W \|\Theta\|_2^2
\end{aligned} \tag{5.19}$$

with $\lambda_S = 10^{12}$ and $\lambda_D = 10^3$. The bar notation $x_{n+j|n}$ reads as ‘‘The simulated state at x_{n+j} starting at n with initial state $x_{n|n}$ ’’. The norm $\|\cdot\|_2^2$ is the mean squared error. The data sample time is T_s .

Effect of Normalization. The effect of the normalization factor τ given a fixed NN configuration is analyzed in simulation mode (hidden states are recursively estimated), and a fixed training pipeline throughout all experiments. Performance is measured in terms of RMSE

$$e_{RMSE} = \sqrt{\frac{1}{N - \max(n_a, n_b)} \sum_{n=\max(n_a, n_b)}^{N-1} (y_n - \hat{y}_n)^2}. \tag{5.20}$$

Fig. 5.3 displays a grid search for different scalar values of T_s/τ fixed prior to the experiment on a log scale ($T_s = 4$ s for CTS). Each experiment is repeated 20 times to account for and observe the effect of randomness in the initial weights. It can be observed in Fig. 5.3 that both the performance and variance of the results get worse for large and small values of τ . Furthermore, Fig. 5.3 indicates that there exists a desirable optimum. The optimum is different from the original sample time of the data, which would be at $T_s/\tau = 4$ for CTS. So without normalization, the expected RMSE lies between $T_s/\tau = 2.3$ and $T_s/\tau = 9.5$ in Fig. 5.3. While $T_s/\tau = 2.3$ reaches good results, the expected RMSE for $T_s/\tau = 9.5$ degrades to 0.65 V, 3 times as much as the results with normalization.

For the trainable normalization method, a normalization vector τ is defined as a trainable network parameter and implemented as in (5.13). It is not set with hyperparameter tuning but jointly trained with all NN weights, with the same learning rate. The normalization is initialized to $T_s/\tau = 0.1$. After training 20 models, an average normalized time $T_s/\tau = 0.072 \pm 0.048$ (mean \pm std) is obtained. For the cross-validation method, it can be observed in Fig. 5.3 that the optimum value is $T_s/\tau = 0.543$. Using BLA and (5.14), a normalization

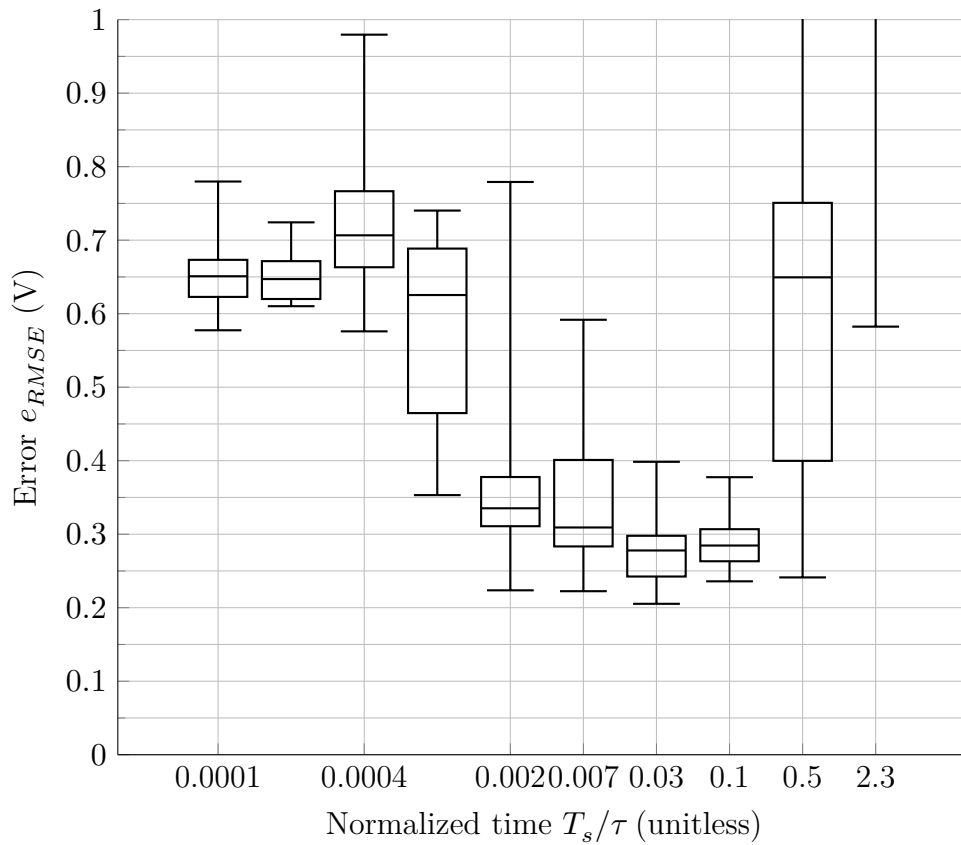


Figure 5.3: 200 independent experiments, each with the same configuration except for the random weight initialization and a fixed scalar normalization factor. 10 different normalization factors repeated 20 times each are tested. The box plot displays the median, lower quartile, upper quartile, minimum and maximum values. Experiments with a normalized time of $T_s/\tau = 40$ sometimes lead to unstable results (despite the weak stability penalty method), with $\text{RMSE} > 10^9$. Compare with [BST22; BST23].

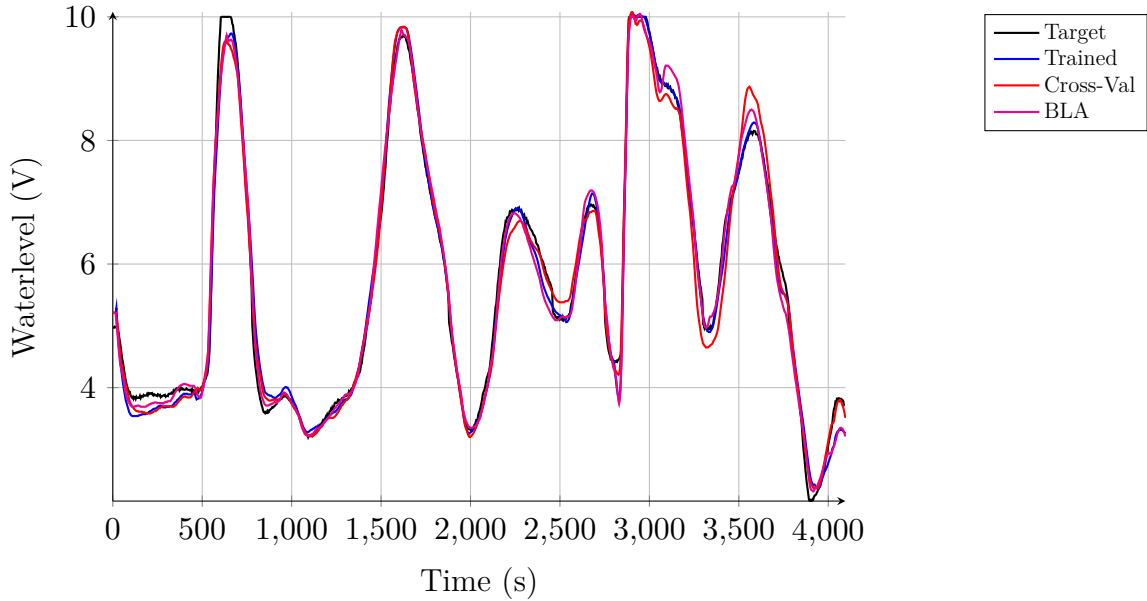


Figure 5.4: Simulation of the CTS setup [Sc16b]. Simulation of the best models gained with the trained normalization factor (Trained), the cross-validation method (Cross-Val), and the Best Linear Approximation (BLA).

factor $T_s/\tau = 0.054$ is estimated, which matches the results in Fig. 5.3.

Comparison to Literature. Since several methods have been applied to CTS, this work can only refer to the ones with the best performance (in terms of RMSE on the test data). The following methods beyond NN give a comparison of performance. [Re17] estimates a BLA and develops an unstructured Nonlinear State-Space Model (NLSS) with different initialization schemes. A nonparametric Volterra Series is designed in [BCS18]. A nonlinear state-space model connected to a Gaussian process is applied in [SS17]. A Tensor Network B-spline model is developed in [KB20]. As a direct comparison, continuous-time NN on this task has been applied in [BST22; FP21; MFP20; WDR21]. These works emphasize initial state estimation, fitting criteria, and model stability differently. Final results of the presented method are shown in Fig. 5.4 and Tab. 5.1.

Table 5.1: Results for the CTS Benchmark. Values with brackets correspond to the average over 5 runs, and values without brackets to the best model.

	test data $e_{RMSE} \text{ V}$
Best Linear Approximation [Re17]	0.75
Truncated Volterra Model [BCS18]	0.54
State-space with GP-inspired Prior [SS17]	0.45
Integrated Neural Networks [MFP20]	0.41
Soft-constrained Integration Method [FP21]	0.40
Stable Runge-Kutta Neural Network [WDR21]	0.39
Nonlinear State-Space Model [Re17]	0.34
Truncated Simulation Error Minimization [FP21]	0.33
Tensor B-splines [KB20]	0.30
Deep Subspace Encoder [BST22]	0.22
ours (SDN, Trained Parameter, best model)	0.2151
ours (SDN, Trained Parameter, mean \pm std)	0.2977 ± 0.1259
ours (SDN, Cross-Validation, best model)	0.2054
ours (SDN, Cross-Validation, mean \pm std)	0.2777 ± 0.052
ours (SDN, BLA, best model)	0.2253
ours (SDN, BLA, mean \pm std)	0.2633 ± 0.0284

5.2 Stability Constraints

This section applies the concept of ISS to Continuous-Time NN. The core result is constraints on the neural network parameters, which can guarantee stability. Stability is enforced for all forecast horizons and all potential noise state and input measurements, with remaining universal function approximation capabilities.

The results presented in this thesis originate from the Ph.D. Thesis of Michael Deflorian [De11b]. There is one error in the proof of the result in [De11b]. Specifically, the origin transformation does not hold. In a jointed work, we found an alternative proof which fortunately leads to the same core results [WDR21]. Besides the alternative proof, this chapter only contains minor differences compared to [De11b] and [WDR21].

A part of the results in [De11b] dropped, as it leads to a null-space solution in practical applications. As many modern NN implementations, such as *PyTorch* [Pa19] and *TensorFlow* [Ma15] do not support constraint optimization, we explain how to use barrier methods to be compliant with those libraries. Furthermore, we introduce weak stability methods, which are less conservative and yet lead to quantifying and improving stability measures of the model.

5.2.1 Empirical Stability Observation

Consider the EMPS benchmark [JGB19]. The EMPS is a standard component for robots or machine tools and is pictured in Fig. 5.5. It is complicated to identify since it suffers from nonlinear, asymmetric friction. The output is the horizontal car position. The input is the force acting on the car, created by the electric motor. The raw data is displayed in Fig. 5.6 and Fig. 5.7. As presented in Fig. 5.7, the input consists of several impulses and therefore requires a high-resolution measurement.

As a consequence of the high-resolution measurements, this benchmark requires a long simulation forecast of 24841 time steps. Applying such a long forecast horizon in the loss function is computationally infeasible with current hardware. Therefore, the data is pre-processed, taking every 10^{th} point and reducing the amount of data to 2485 time steps. Regarding this benchmark, the interpolation error applying this technique is not too

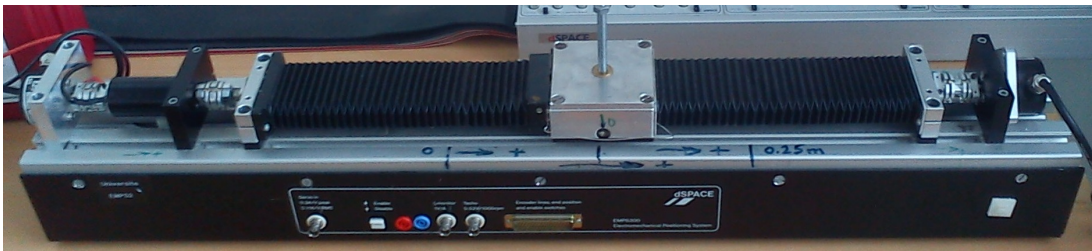


Figure 5.5: Electro Mechanical Positioning System (EMPS) Setup. [JGB19]

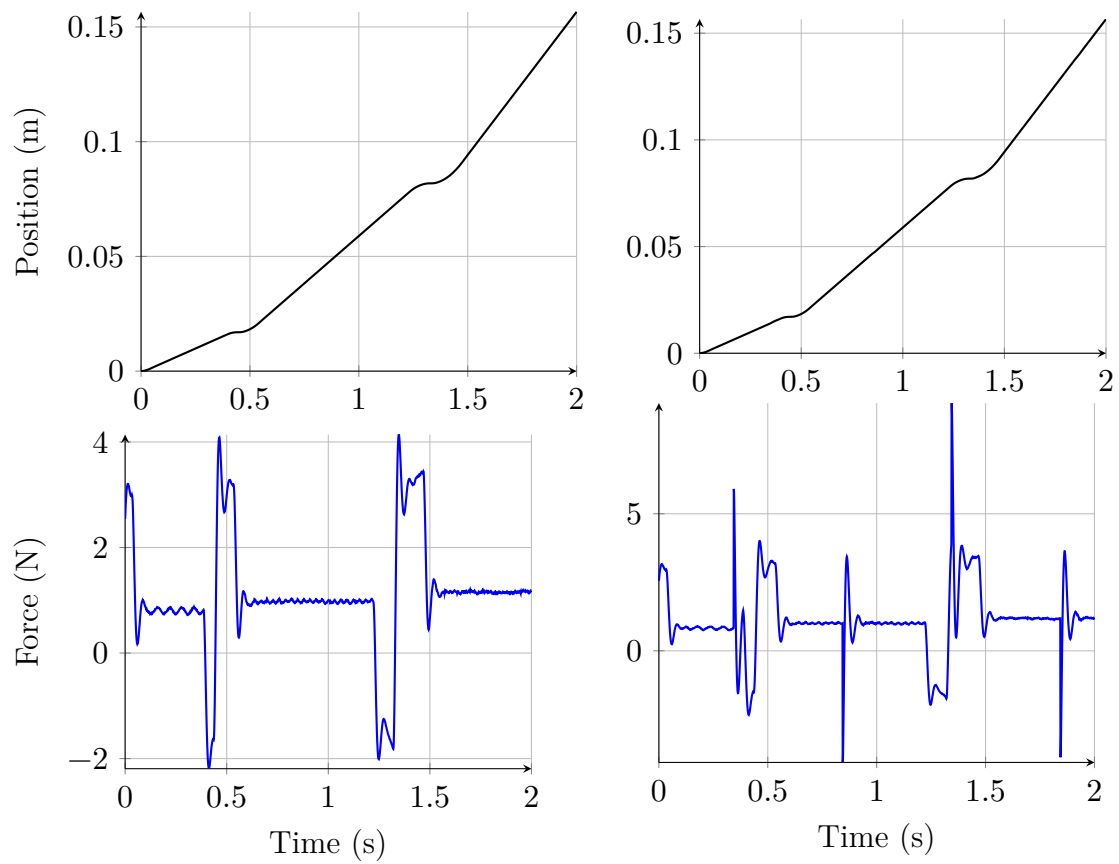


Figure 5.6: Data of the EMPS benchmark. Zoom on the first 2s with a sample time of 1 ms. Upper Row: Position output. Lower Row: Force input. Left side: Training data. Right side: Test data.

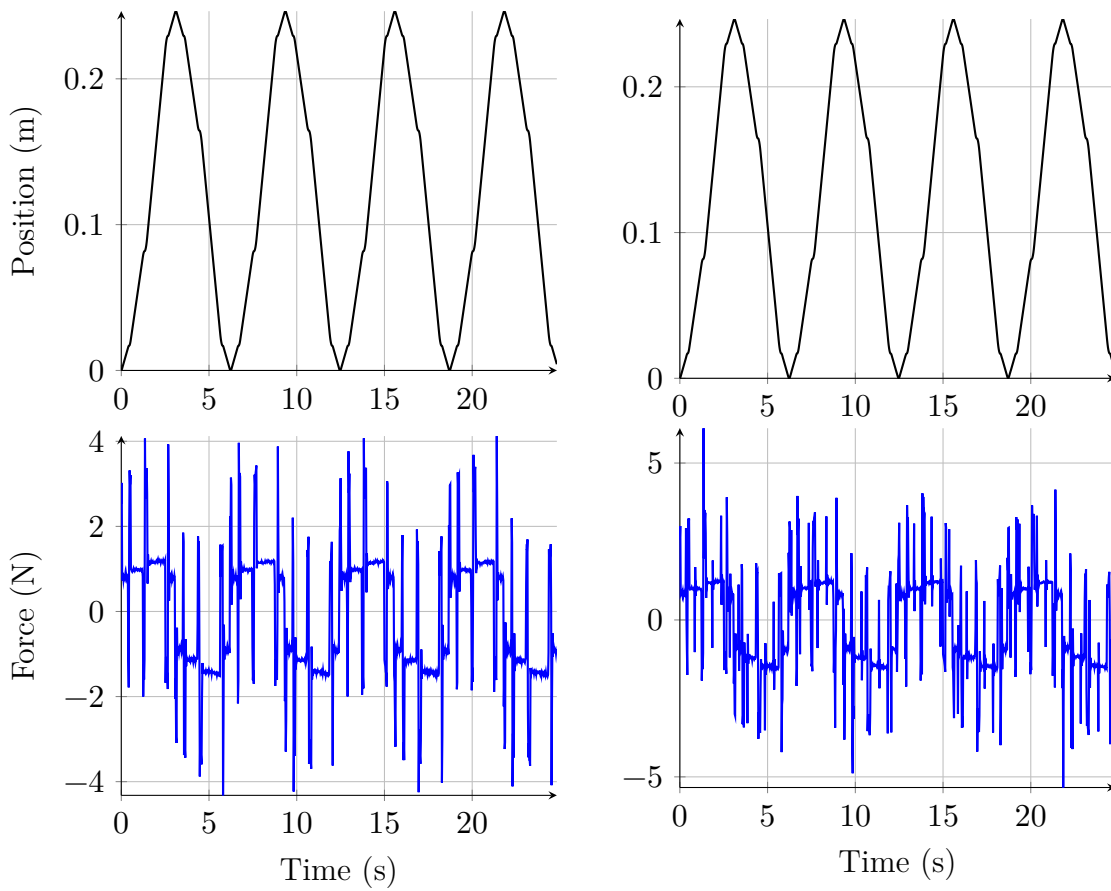


Figure 5.7: Data of the EMPS benchmark over the complete time horizon of 25 s. Upper Row: Position output. Lower Row: Force input. Left side: Training data. Right side: Test data.

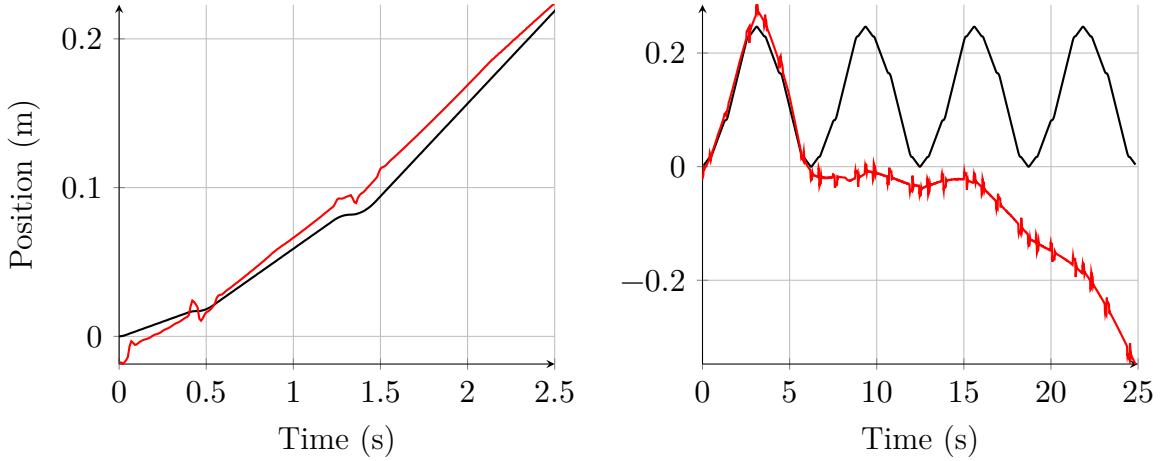


Figure 5.8: Simulation results without stability barrier. Left: First batch of training. Right: Simulation on test data. Black: Measured data. Red: Model prediction.

large. However, further interpolation leads to errors, as the input data are high-frequency impulses; compare Fig. 5.6 (bottom). Still, the model computationally requires to be trained on smaller batches, which is set to 264 time steps in the loss function each.

Fig. 5.8 shows a representative solution using a continuous-time NN. A single training batch with the trained model is displayed on the left side, with an absolute error magnitude of 0.001 m. On the right side, as requested by the benchmark, the same trained model is applied on the full trajectory of 2485 time steps in simulation mode. The absolute error increases by a factor of more than 200. The graph shows that the error might increase further over a longer horizon as the model appears unstable. At the end of this chapter, the exact same model with the presented stability constraints is trained as a direct comparison.

Considering stability with continuous-time NN in general can be illustrated using two simple examples, accounting for the continuous-time stability and the discretization method, as both aspects contribute to ISS. First, consider a learned continuous-time neural network which is GAS, but its states tend to infinity in the presence of bounded inputs. This example underlines the importance of ISS instead of GAS.

Consider a continuous-time neural network with one state $x(t) \in \mathbb{R}$, one input $u(t) \in \mathbb{R}$, continuous-time $t \in \mathbb{R}$, hyperbolic tangent activation function $\tanh(\cdot)$, no bias weights and two neurons in the hidden layer,

$$\dot{x}(t) = [-1 \quad 10] \tanh \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \right). \quad (5.21)$$

The network (5.21) is GAS as the autonomous system reduces to $\dot{x}(t) = -\tanh(x)$ and

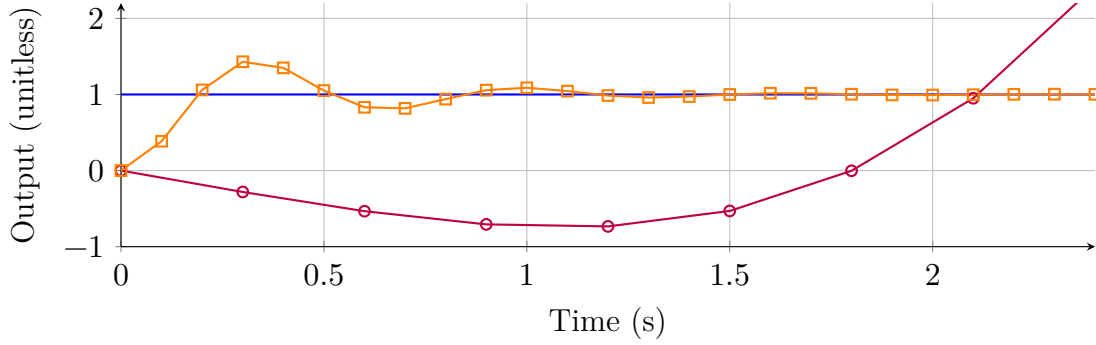


Figure 5.9: Step response of a stable and unstable time discretization. Circles and squares represent each discrete-time step. Blue: Step input. Orange: Stable discretization. Purple: unstable discretization.

converges to zero. But any given input $2 \leq u(t) \leq 10^6$ leads to

$$\dot{x}(t) = -\tanh(x(t)) + 10 \tanh(u(t)) \quad (5.22)$$

$$> -\tanh(x(t)) + 9 > 8 \quad (5.23)$$

and the system trajectory tends to infinity, although the input is bounded.

The second example shows that even an ISS neural network can become unstable if neural network constraints do not account for the sample time h (see section 3.3 for a definition of the sample time). Consider a standard state-space neural network with Eigenfrequency $\omega_0 = 10 \frac{1}{s}$, damping $D = 0.5$ and input gain $K = 1$. The network consists of two states, one input, no bias terms, and no hidden layers,

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -\omega_0^2 & -D\omega_0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ K\omega_0^2 \end{bmatrix} u(t). \quad (5.24)$$

Equation (5.24) is input-affine and the real parts of all eigenvalues are negative. As a consequence, it is GAS and ISS in the continuous domain. Assume (5.24) is time discretized by a 4th-order Runge-Kutta method and a sampling frequency much greater than the Eigenfrequencies. As shown exemplarily in Fig. 5.9 for a step response, the discrete-time trajectory for (5.24) tends to infinity for $h = 0.3$ s, although it is stable for $h = 0.1$ s. This trade-off between neural network weights and sample time generally is non-trivial.

To the best of the author's knowledge, the presented stability constraints are the only ones that can enforce ISS in the given examples.

5.2.2 Preliminaries for Stability

Consider a continuous-time and a discrete-time system

$$\dot{x}(t) = f_C(x(t), u(t)) \quad (5.25)$$

$$x_{n+1} = f_D(x_n, u_n). \quad (5.26)$$

The following ISS criterion for continuous-time neural networks builds upon previous works. The concept of ISS is introduced in [So08]. It extends Lyapunov stability to non-autonomous systems, e.g., $u(t) \neq 0$. Besides continuous systems, discrete-time ISS criterion for (5.26) was developed in [JW01]. Additionally, it is known that if a system is ISS and the point of equilibrium is the origin, it is also 0-GAS [Kh02].

For the sake of simplicity, the time dependency notation of state and input variables is neglected, e.g., $x(t) = x$, $u(t) = u$. Consider an ISS-Lyapunov function

$$V(x) = \frac{1}{2} \langle x, x \rangle_{P_1} \quad (5.27a)$$

$$\dot{V}(x) = \dot{x}^T P_1 x \leq -\beta \frac{1}{2} \langle x, x \rangle_{P_1} + \gamma \langle u, u \rangle \quad (5.27b)$$

with a positive definite matrix $P_1 > 0$, $\beta > 0$ and $\gamma \geq 0$. The scalar product for two vectors v_1, v_2 is defined as $\langle v_1, v_2 \rangle_V = v_1^T V v_2$. The identity matrix is applied for the scalar product if not mentioned otherwise.

Definition 5.2.1 (Weakly B-ISS RK method [DR14]) *A RK-method is called weakly B-input-to-state stable if for each (5.25) satisfying (5.27) the numerical approximation (3.7) is ISS for all sufficiently small h .*

The property is called weakly B-ISS because definition (5.2.1) depends on (5.25) and on the sample time h . In Butcher's original terminology, the B-stability is a property of the scheme (3.7), which is independent of (5.25) and h . In [DR14] and the extended German version [De11b], the following result is presented:

Theorem 5.2.2 (proof see [De11b; DR14]): *A (k, l, m) -algebraically stable RK method with $l < 0$, $m = 1$, $0 < k < 1$, $\gamma > 0$ and a positive definite matrix P_1 is weakly B-ISS, if for system (5.25)*

$$\langle x, f_C(x, u) \rangle_{P_1} + hm \langle f_C(x, u), f_C(x, u) \rangle_{P_1} < \gamma \langle u, u \rangle \quad (5.28)$$

holds at any time.

The coefficients k, l, m depend on the explicit RK method. Various (k, l, m) -algebraically stable RK methods are discussed in [Co84]. For example, the classical 4 step RK is $(k(l), l, m)$ -algebraically stable with $m = 1$ [Co84]. Two possibilities ensure B-ISS:

1. fix the sample interval h and choose the model weights so that (5.28) holds, or
2. choose the model weights so that $\langle x, f_C(x, u) \rangle_{P_1} < 0$ and then search for every sample interval the maximum h so that (5.28) holds.

This work concentrates on the first condition and views the sample time as a fixed constant within one data set.

5.2.3 Existence and Uniqueness of Equilibrium

To discuss the GAS and the ISS of the RKNN (3.6), the existence and uniqueness of the trajectories are prerequisites. A sufficient condition for this is to satisfy the Lipschitz condition [Kh02]

$$\|f_{NN}(x_1, \tilde{u}_1) - f_{NN}(x_2, \tilde{u}_1)\| \leq l_X \|x_1 - x_2\| \quad (5.29)$$

$$\|f_{NN}(x_1, \tilde{u}_1) - f_{NN}(x_1, \tilde{u}_2)\| \leq l_U \|\tilde{u}_1 - \tilde{u}_2\| \quad (5.30)$$

for any two states $x_1, x_2 \in \mathbb{R}^{N_X}$ and for any two augmented inputs $\tilde{u}_1, \tilde{u}_2 \in l_\infty^{N_U+1}$. The first condition can be estimated with

$$\|f_{NN}(x_1, \tilde{u}_1) - f_{NN}(x_2, \tilde{u}_1)\| \quad (5.31)$$

$$= \|W_x^l x_1 + W_u^l \tilde{u}_1 + W_a^o \sigma(W_x^h x_1 + W_u^h \tilde{u}_1) - W_x^l x_2 - W_u^l \tilde{u}_1 - W_a^o \sigma(W_x^h x_2 + W_u^h \tilde{u}_1)\| \quad (5.32)$$

$$\leq \|W_x^l\| \|x_1 - x_2\| + \|W_a^o\| \cdot \|\sigma(W_x^h x_1 + W_u^h \tilde{u}_1) - \sigma(W_x^h x_2 + W_u^h \tilde{u}_1)\| \quad (5.33)$$

$$\leq \|W_x^l\| \|x_1 - x_2\| + \|W_a^o\| \cdot \|W_x^h x_1 + W_u^h \tilde{u}_1 - W_x^h x_2 - W_u^h \tilde{u}_1\| \quad (5.34)$$

$$= \|W_x^l\| \|x_1 - x_2\| + \|W_a^o\| \|W_x^h x_1 - W_x^h x_2\| \quad (5.35)$$

$$\leq \|W_x^l\| \|x_1 - x_2\| + \|W_a^o\| \|W_x^h\| \|x_1 - x_2\| \quad (5.36)$$

$$= l_X \|x_1 - x_2\|. \quad (5.37)$$

Analogously, the equation

$$\|f_{NN}(x_1, \tilde{u}_1) - f_{NN}(x_1, \tilde{u}_2)\| \leq l_U \|\tilde{u}_1 - \tilde{u}_2\| \quad (5.38)$$

holds. By defining

$$l_X = \|W_x^l\| + \|W_a^o\| \|W_x^h\| \quad (5.39)$$

$$l_U = \|W_u^l\| + \|W_a^o\| \|W_u^h\| \quad (5.40)$$

the Lipschitz condition is satisfied. Thus, the existence and uniqueness of the trajectories of the RKNN (3.6) are ensured. It is a prerequisite that the action function $\sigma(\cdot)$ fulfills (5.29). This is the case for tanh and ReLU, among other activation functions.

$$0 \leq \frac{\tanh(v)}{v} = \frac{e^v - e^{-v}}{(e^v + e^{-v})v} \leq 1 \quad (5.41)$$

$$0 \leq \frac{\text{relu}(v)}{v} = z(v) \leq 1, \quad z(v) = \begin{cases} 0 & v < 0 \\ 1 & v \geq 0. \end{cases} \quad (5.42)$$

5.2.4 Constraints on the Neural Network Parameters

It is a prerequisite for GAS and ISS of the RKNN (3.6) that its point of equilibrium is the origin. This is the case for the state network, e.g., $f_{NN}(0, 0) = 0$. However, note that this prerequisite is not fulfilled if the bias terms of the network are not connected to an augmented input \tilde{u} . Defining the bias terms as additional input dimension fulfills the GAS and ISS prerequisites while remaining full nonlinear modeling capabilities. Otherwise, a complicated nonlinear coordinate transform is required if bias terms are defined as part of the state network rather than depending on the input.

To simplify the derivation of the ISS criterion, the state network is reformulated with a nonlinear Lipschitz matrix rather than nonlinear activation functions. Therefore, the condition that the activation function $\sigma(\cdot)$ is Lipschitz continuous and that $0 \leq \frac{\sigma(v)}{v} \leq 1$, $\forall v \in \mathbb{R}_{\neq 0}$ holds is utilized. This requirement is obligatory for the stability constraints. As a result the Lipschitz constant is $0 \leq l_i = \frac{\sigma(v_i)}{v_i} \leq 1$, $\forall i \in \{1, 2, \dots, N_N\}$. The diagonal Lipschitz matrix can be defined as $L(x, \tilde{u}) = \text{diag}(l_1(x, \tilde{u}), l_2(x, \tilde{u}), \dots, l_{N_N}(x, \tilde{u}))$ and get

$$\dot{x} = f_{NN}(\hat{x}, \tilde{u}) = W_x^l x + W_u^l \tilde{u} + W_a^o L(x, \tilde{u})(W_x^h x + W_u^h \tilde{u}). \quad (5.43)$$

Note that (5.43) is only used to derive the stabilization criterion. The prediction or

simulation properties of the RKNN are not changed in any way. For the sake of simplicity, the dependency of the Lipschitz matrix on the state and the augmented input is omitted in the following, e.g., $L = L(x, \tilde{u})$. Using Theorem 5.2.2 the weights have to be selected such that

$$\langle x, f_{\text{NN}}(\hat{x}, \tilde{u}) \rangle_{P_1} + hm \langle f_{\text{NN}}(\hat{x}, \tilde{u}), f_{\text{NN}}(\hat{x}, \tilde{u}) \rangle_{P_1} < \gamma \langle \tilde{u}, \tilde{u} \rangle \quad (5.44)$$

holds.

Theorem 5.2.3 (ISS of RKNN) *The RKNN (3.6) using a (k, l, m) -algebraically stable RK method is ISS, if there exist two positive definite matrices P_1, P_3 and weighting variables $\sum \lambda_i = 1, 0 \leq \lambda_i \leq 1, \forall i \in \{1, 2, \dots, N_N\}$ such that*

$$T_1 < 0 \quad (5.45)$$

$$T_{2,i} + \lambda_i T_1 \leq 0 \quad \forall i \in \{1, 2, \dots, N_N\} \quad (5.46)$$

holds. The i^{th} row vector of matrix W_x^h is R_i^{Wh} and the i^{th} column vector of matrix W_a^o is C_i^{Wo} . The matrices T_1 and $T_{2,i}$ are defined by

$$T_1 = P_3 + P_1 W_x^l + hm \left(W_x^l{}^T P_1 W_x^l + \|W_a^o{}^T P_1 W_a^o\| \|W_x^h\|^2 I \right) \quad (5.47)$$

$$T_{2,i} = \left(hm W_x^l{}^T + I \right) P_1 C_i^{Wo} R_i^{Wh} + hm \left(C_i^{Wo} R_i^{Wh} \right)^T P_1 W_x^l. \quad (5.48)$$

Proof: See appendix A.

Readers who are only interested in GAS of the RKNN and not in ISS can see from the proof that all terms associated with P_3 vanish (appendix A, (4) and (9)).

Theorem 5.2.4 (GAS of RKNN) *The RKNN (3.6) using a (k, l, m) -algebraically stable RK method is GAS, if there exists a positive definite matrix P_1 and weighting variables $\sum \lambda_i = 1, 0 \leq \lambda_i \leq 1, \forall i \in \{1, 2, \dots, N_N\}$ such that*

$$T_3 < 0 \quad (5.49)$$

$$T_{2,i} + \lambda_i T_3 \leq 0 \quad \forall i \in \{1, 2, \dots, N_N\} \quad (5.50)$$

holds. The i^{th} row vector of matrix W_x^h is R_i^{Wh} and the i^{th} column vector of matrix W_a^o is C_i^{Wo} . The matrices T_3 and $T_{2,i}$ are defined by

$$T_3 = P_1 W_x^l + hm \left(W_x^{lT} P_1 W_x^l + \|W_a^{oT} P_1 W_a^o\| \|W_x^h\|^2 I \right), \quad (5.51)$$

$$T_{2,i} = \left(hm W_x^{lT} + I \right) P_1 C_i^{Wo} R_i^{Wh} + hm \left(C_i^{Wo} R_i^{Wh} \right)^T P_1 W_x^l. \quad (5.52)$$

Remark: For $h = 0$, $\lambda_i = \frac{1}{N_N}$, $N_N = N_X$, $W_a^o = I$, $W_x^l = \text{diag}(d_1, \dots, d_{N_X})$, $d_i > 0$, $P_3 = 0$ and P_1 a diagonal matrix the constraints (5.2.3) and (5.2.4) reduce to the results presented in [HW02]. Therefore, the Theorems 5.2.3 and 5.2.4 generalize the result of GAS for continuous-time neural networks presented in [HW02] as different network structures, various sample times, more options in the stability criterion and ISS can be ensured.

5.2.5 Weak Stability Method

Recall the main idea of the provided ISS stability criterion. A linear model is introduced for which stability properties are well-known in literature and can be explicitly quantified. Then a nonlinear term is added to the model in parallel to ensure universal function approximation capabilities. Based on the Lipschitz continuity of the model, an upper bound regarding the stability of the nonlinear term can be computed. The algorithm ensures that the safety margin of the linear term is greater than the uncertainty about the nonlinear term.

Deriving the conditions on the nonlinear model terms is elaborate and requires to be derived for each model class. For instance, using $\text{relu}(\cdot)$ or $\text{tanh}(\cdot)$ activation functions is obligatory for the presented results. Relaxing the prerequisite regarding the activation function, that $0 \leq \sigma(v)/v \leq 1$ holds, is rather difficult. Furthermore, the proof only holds for one hidden layer. Extensions to additional hidden layers are possible, yet require considerable expenses.

From a practical point of view, a flexible approach for many model classes that still encourages stability would be desirable. For non-safety critical applications, one can achieve this by trading off guaranteed stability for constraints flexibility. The idea is to ensure the stability properties of the linear model terms without considering the bounded uncertainty of the nonlinear terms by simply implementing the constraint

$$W_x^l \leq 0. \quad (5.53)$$

Equation (5.53) ensures negative definiteness of the nonlinear kernel. The weak stability criterion can be applied to a wide range of nonlinear models without deriving the complete stability criterion for each specific nonlinear kernel. It can be included in the training pipeline using a barrier function (see section 3.5.2). Specifically,

$$L_{WeakStabilityBarrier}(\cdot) = \begin{cases} L_{Fit}(\cdot) & \text{if } \max(\text{eig}(W_x^l)) \leq \lambda_2 \\ L_{Fit}(\cdot) + L_{StabilityBarrier}(\cdot) & \text{if } \lambda_2 < \max(\text{eig}(W_x^l)) \leq 0 \\ \infty & \text{otherwise} \end{cases} \quad (5.54)$$

$$L_{StabilityBarrier}(\cdot) = \lambda_1 \text{relu}(\max(\text{eig}(W_x^l)) + \lambda_2) \quad (5.55)$$

Instead of the maximum eigenvalue, negative definiteness can be guaranteed using Sylvester’s criterion [Gi17]. Sylvester’s criterion, in contrast to formulations using an Eigenvalue function, is beneficial for the AD algorithm as it is fully differentiable. Some implementations of $\text{eig}(\cdot)$ are differentiable, yet some in libraries differentiation through $\text{eig}(\cdot)$ is challenging. The following Python code E.4 written for the *PyTorch* library implements the weak stability method with a barrier function using Sylvester’s criterion. The constants in the code refer to $\gamma_1 \approx \text{stability_scaling} = 10^6$, $\gamma_2 \approx \text{stability_offset} = 10^{-2}$ and $\infty \approx \text{stability_barrier} = 10^{12}$. Infinity is limited to 10^{12} for numerical reasons. The neural network weight matrix W_x^l is A. The constant stability_offset can be adapted to the specific application, the constants $\text{stability_scaling} = 10^6$ and $\text{stability_barrier} = 10^{12}$ are reasonable for a wide variety of data sets. The code is given in appendix E.4.

5.2.6 Empirical Stability Results

The proposed constraints are discussed using the EMPS benchmark in section 5.2.1. As the network consists of multiple hidden layers, only the weak stability method is applied. Using the *PyTorch* [Pa19] framework, the constraints are implemented as a progressive barrier in the loss function as recommended in section 3.5.2, with a scaling of $\gamma_2 = 10^6$ and stability offset of $\gamma_1 = 10^{-2}$. Besides the additional barrier, no changes have been applied. The network architecture, data preprocessing, and training configuration remain unchanged.

The empirical result using the progressive barrier method is depicted in Fig. 5.10. This result can be directly compared to the simulation without the stability constraints in Fig. 5.8.

The performance utilizing a maximum absolute error of 0.001 m is comparable for each training batch. For the simulation over the complete test data, the model’s error with

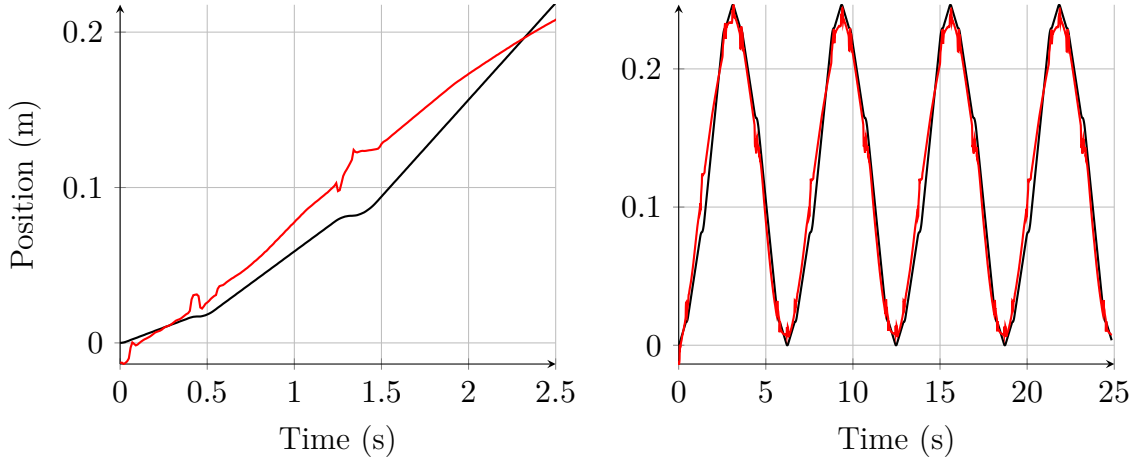


Figure 5.10: Results on the Electro-Mechanical Positioning System benchmark with stability barrier. Besides the additional barrier in the loss function, the model and training configuration are identical. Left: First batch of training. Right: Simulation on test data. Black: Measured data. Red: Model prediction.

stability constraints raises by a factor of 10 to 0.01 m. This might be due to an overfit to the training data. Note that the absolute error on the test data does not increase over time, comparing a first (0 s to 5 s) and a last (20 s to 25 s) interval.

The direct comparison for the model without a stability barrier is significant. On the test data, the error increases by a factor of 200, from 0.001 m to 0.2 m. The error increases over time, comparing the first and last intervals. For a longer experiment, likely, the absolute error will increase further.

5.2.7 Additional Methods to Improve Model Stability

Constraints on the NN weights have been presented, based on [De11b; WDR21]. These constraints can enforce ISS and GAS stability.

Other possibilities can also lead to high-performance, stable models. The presented weak stability constraints are only one possible indigent. Regarding safety-critical applications requiring a provable ISS stability, the other methods discussed in the following are *not* sufficient.

First, stability issues arise when inference requires extrapolation in the value or time domain. Therefore, the longer the forecast horizon in the loss function, the less extrapolation is required and the better the model performance. As a consequence, within the application’s demands and computational limitations, stability can be encouraged by decreasing the inference horizon and increasing the training horizon.

Enforcing the observable output as a subset of the hidden states improves stability, too. The main idea is to restrict the hidden state flexibility by explicitly enforcing prior knowl-

edge. This can be achieved by setting the linear output terms to constant, non-trainable parameters and disabling the nonlinear output function in (3.2).

Stability accounts for two aspects, model stability, and ODE solver stability. Regarding the ODE solver, there are three options:

- increasing the numerical order of the solver,
- using a variable-step solver with error control,
- or decreasing the sample time for a fixed-step solver.

5.3 Demonstration of Continuous-Time Memory Efficiency

In this section, we compare discrete-time and continuous-time NN on the same data. The theoretical outcome of this comparison is already known in literature [Ki22]. However, it is worth testing on the application of robot identification if the effort for a continuous-time NN is justified.

For direct comparability of discrete-time and continuous-time models, advanced techniques are dismissed for both variants. Two identical Multi Layer Perceptron (MLP) networks are applied, and the continuous-time solution only embeds the MLP in an Eulerscheme. Other than adding the Eulerscheme, no changes are incorporated. The data and data preprocessing are identical, as the network architecture, and the training pipeline.

Both models are tested on the complete working space. The design of experiments is described in section 6.1.

5.3.1 Discrete-Time Baseline

A discrete-time MLP network is developed for system identification, which is capable of universal function approximation. In theory, given sufficiently rich data, it is capable of approximating (2.2) with arbitrary accuracy. To improve the gradients and additionally to make the MLP network better comparable with the linear baseline, a linear and a nonlinear module are added,

$$\tau_n = B_{NN} \begin{bmatrix} q_n \\ \dot{q}_n \\ \ddot{q}_n \end{bmatrix} + W_1 \sigma \left(W_2 \sigma \left(W_3 \begin{bmatrix} q_n \\ \dot{q}_n \\ \ddot{q}_n \end{bmatrix} + b_3 \right) + b_2 \right) + b_1 \quad (5.56)$$

with the linear module $B_{NN} \in \mathbb{R}^{N_Q \times 3N_Q}$, output and two hidden weight matrices $W_1 \in$

$\mathbb{R}^{N_Q \times N_F}$, $W_2 \in \mathbb{R}^{N_F \times N_F}$, $W_3 \in \mathbb{R}^{N_F \times 3N_Q}$, and bias vectors $b_1 \in \mathbb{R}^{3N_Q}$, $b_2 \in \mathbb{R}^{N_F}$, $b_3 \in \mathbb{R}^{N_F}$, with $N_F \in \mathbb{N}$ hidden units.

Recall the complexity of the inertia, Coriolis, and centripetal matrices, $M(q)$, $C(q, \dot{q})$, as well as the gravitational load including HWC $g(q)$. As discussed in section 4.1.3, the symbolic representation of the physical model consists of more than ten thousand arithmetic operations:

- 2000 constants,
- 10000 multiplications,
- 2500 additions,
- 2000 subtractions,
- 50 cosine,
- 60 sine, and
- 30 square functions.

As a consequence, a relatively large MLP 5.56 is anticipated with a nonlinear kernel consisting of two hidden layers W_2 , W_3 , b_2 and b_3 . It features LeakyReLU activation functions, and $N_F = 128$ hidden units. The linear model core B_{NN} consists of 108, and the nonlinear core contains ≈ 59000 parameters. The performance of the MLP on the training data is presented in Fig. 5.11 and Fig. 5.12.

As to be expected for a flexible model such as a MLP with universal function approximation capabilities, the model can represent the training data accurately, see Fig. 5.11 and Fig. 5.12. The results for all joints are presented in appendix D. However, applying the trained model to the test data reveals severe drawbacks of the discrete-time model, see Fig. 13. Indeed, the performance differences of the models presented in this chapter are sufficiently large to be visible without any numerical validation performance metrics.

Note that only the sequence from 5 s to 45 s is relevant for training and test evaluation. Before and after this period, the robot is at a standstill, and positioning accuracy is not subject to dynamic effects. Feed-forward torque estimation in the standstill phases of the robot is not relevant to improve the robot's accuracy. However, the motor torques in the standstill phases of the robot are very difficult to estimate.

Furthermore, note that training on a sequence from 5 s to 45 s only contains 450 data points for each joint. This is a restriction of the benchmark data to test the model for data efficiency explicitly. The discussion of the implications of this restriction is postponed to section 5.3.3.

5.3.2 Continuous-Time Baseline

The expected robot model (2.2) is a *static* model in the sense that feed-forward torques do not depend on previous motor torques. The model does not contain any time-dependent memory. However, as a dynamical system including a feedback control loop, the torque time series is not completely independent but strongly auto-correlated.

It can be advantageous to apply continuous-time models, even in a context where a static model would be sufficient. In this section, the MLP model is transformed from a discrete-time to a continuous-time setting. Advanced continuous-time options, such as DAE and stability penalty methods, state derivative normalization, or higher-order ODE solvers, are not applied yet. This makes the simple discrete-time and continuous-time versions of the MLP directly comparable. In section 6.2.2, the full potential and all recommendations for continuous-time NN as given in chapter 3 are used.

From a physical point of view, as the expected model (2.2) is static, it must be lifted into a derivative formulation to apply a continuous-time model.

$$\frac{d\tau}{dt} = \frac{dU^{-1} (M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) + \tau_H(q) + \tau_F(\dot{q}))}{dt} \quad (5.57)$$

$$\frac{d\tau}{dt} = U^{-1} \left(\frac{dM(q)\ddot{q}}{dt} + \frac{dC(q, \dot{q})\dot{q}}{dt} + \frac{dg(q)}{dt} + \frac{d\tau_H(q)}{dt} + \frac{d\tau_F(\dot{q})}{dt} \right) \quad (5.58)$$

$$\begin{aligned} \frac{d\tau}{dt} = U^{-1} \left(\frac{dM(q)}{dt} \ddot{q} + M(q) \ddot{\ddot{q}} + \frac{dC(q, \dot{q})}{dt} \dot{q} + C(q, \dot{q}) \ddot{q} \right. \\ \left. + \frac{dg(q)}{dt} + \frac{d\tau_H(q)}{dt} + \frac{d\tau_F(q)}{dt} \right). \end{aligned} \quad (5.59)$$

Consequently, the continuous-time NN does *not* model the physical robot as represented in (4.36). Instead, the continuous-time NN represents the physical equations (5.60a). The derivative of the inertia matrix $\frac{dM(q)}{dt}$, Coriolis and centripetal $\frac{dC(q, \dot{q})}{dt}$, gravitational loads $\frac{dg(q)}{dt}$ and HWC $\frac{d\tau_H(q)}{dt}$ is complex as none of those depend linear on the link angle q or its derivatives \dot{q} , \ddot{q} . Only the inverse gearbox ratio U^{-1} is linear concerning the joint angle q . Furthermore, (5.60a) depends on the joint angle jerk $\ddot{\ddot{q}}$, information which is not directly supported using the input data of the benchmark. It could be estimated using Finite-Difference methods and appropriate filtering. As presented in the following, the continuous-time NN can derive this information, even in the non-advanced MLP configuration.

To demonstrate the capabilities of the most simple version of a continuous-time NN, an Euler-Backward as ODE solver and the constant time step $h = 1$ (without unit) is applied. An output network is not enabled. Indeed, the output is equal to the state, which is not

augmented. The resulting NN formulation is

$$\frac{d\tau_n}{dt} = A_{NN}\tau_n + B_{NN} \begin{bmatrix} q_n \\ \dot{q}_n \\ \ddot{q}_n \end{bmatrix} + \tilde{f}_{NN} \left(\tau_n, \begin{bmatrix} q_n \\ \dot{q}_n \\ \ddot{q}_n \end{bmatrix} \right) \quad (5.60a)$$

$$\tau_{k+1} = \tau_n + \frac{d\tau_n}{dt} \quad (5.60b)$$

$$\tau_{k+1} = (I + A_{NN}) \tau_n + B_{NN} \begin{bmatrix} q_n \\ \dot{q}_n \\ \ddot{q}_n \end{bmatrix} + \tilde{f}_{NN} \left(\tau_n, \begin{bmatrix} q_n \\ \dot{q}_n \\ \ddot{q}_n \end{bmatrix} \right). \quad (5.60c)$$

The test data results are depicted in Fig. 5.11 and Fig. 5.12. Additional experiments are presented in the appendix D. Results on the training data for the Euler model are not displayed as the performance on this data is satisfying. Otherwise, an accurate solution using the test data would be impossible.

The comparison of the Euler NN and the discrete-time MLP in Fig. 5.11 and Fig. 5.12 is decisive. The data-based model changed from a type that performs worse than applying no feed-forward model to one that can fairly represent the data. The only difference is the change of the formulation from a discrete-time to a continuous-time formulation. The parts with identical configurations are listed in the following.

- training pipeline (ADAM optimizer, learning rate, and learning rate scheduler, early stopping criteria, parameter weight decay, and MSE regression loss function),
- data set preprocessing (no filtering, data set reduction, coherent validation data definition, and z-score normalization with mean offset), and
- model architecture (number of layers, activation function type, number of hidden units, number of states, inputs and outputs, linear plus nonlinear kernel, omitted output network, weight initialization, initial state estimation).

Note that (5.60) depends on the previous motor torque τ_n . Therefore, the feed-forward control needs to be estimated at run time, in contrast to the static robot model (5.56). The static model only depends on the reference trajectory and can be computed online or offline. Computing the feed-forward torque in the trajectory planning phase enables the feedback control to be computed faster as less online computing is required. Furthermore, offline computation increases robot safety because the feed-forward torque can be extensively validated before execution. However, there are possibilities to compute the torque offline.

- Use a simulation model.
- Configure the NN model in simulation mode and estimate the torque recursively.

For initial torque estimation, see section 3.2.

- If the robot task is repeatable, torque measurements of previous runs can be applied.

5.3.3 Baseline Comparison

Memory efficiency is defined in this work as the parameter memory $m_p \in \mathbb{R}$ required to achieve comparable performance in terms of prediction accuracy. A number of $N_p \in \mathbb{N}$ model parameters with a size of $s \in \mathbb{N}$, depending on the data type, and a model accuracy $L \in \mathbb{R}$ leads to

$$m_p = \frac{L}{\sum_{i=1}^{N_p} s_i}. \quad (5.61)$$

For example, a single floating point number in C requires $s = 64$ bit. Typical loss functions L are given in Tab. 3.3 and Tab. 3.3. The definition (5.61) is broader than BIC or AIC, which only account for L and N_p . Memory efficiency in the definition (5.61) captures the information contained relative to the required disk storage. In other words, it indicates the efficiency of the model representation.

Memory efficiency is beneficial for four reasons. First, the required disk storage can be reduced and thus model deployment is facilitated. Second, model training is possibly computationally faster. Third, the model can be trained on fewer data, as demonstrated for example in this section. An optimization problem is simplified when a lower parameter accuracy of the objective variables is required. And finally, following the idea of BIC or AIC, model overfit is aggravated.

Recall the discrete-time MLP and compare the training data to the test data of the same NN in Fig. 5.11 and Fig. 5.12. Note that the training data is represented properly. For the robot feed-forward control application, only the performance on the test data is relevant and not the training. Nevertheless, the direct comparison of training and test appears to contain overfitting. A network with ≈ 59000 parameters is trained on $6 \cdot 450 = 2700$ target data points. Decreasing the number of model parameters does not improve the model performance, as the expected physical effects are too complicated, see section 2.2.

The question is whether more training data improves the performance of the discrete-time MLP. Therefore, the full training data set is applied, with $6 \cdot 39988 = 239928$ target data points. The result is presented in Fig. 5.11 and Fig. 5.12 for training data and test data respectively. For a direct comparison, only the number of training data points increased, from 2700 data points to 239928, and all other configurations were unmodified. Still, the discrete-time NN trained on 239928 data points perform worse on the test data than the continuous-time counterpart trained on 2700 data points. Nevertheless, the improvement

resulting only from additional data for the MLP is significant. While 2700 data points resulted in a non-applicable model, training on 239928 data points resulted in a useful model, see Fig. 5.11 and Fig. 5.12 respectively.

The continuous-time model performed even *better* using only the *few* data points. The continuous-time model consists of 124231 model parameters more due to the additional state information than the discrete-time counterpart with 59496 parameters. More model parameters increase the model's flexibility, leading to more model variance and increasing the risk of a bad generalization.

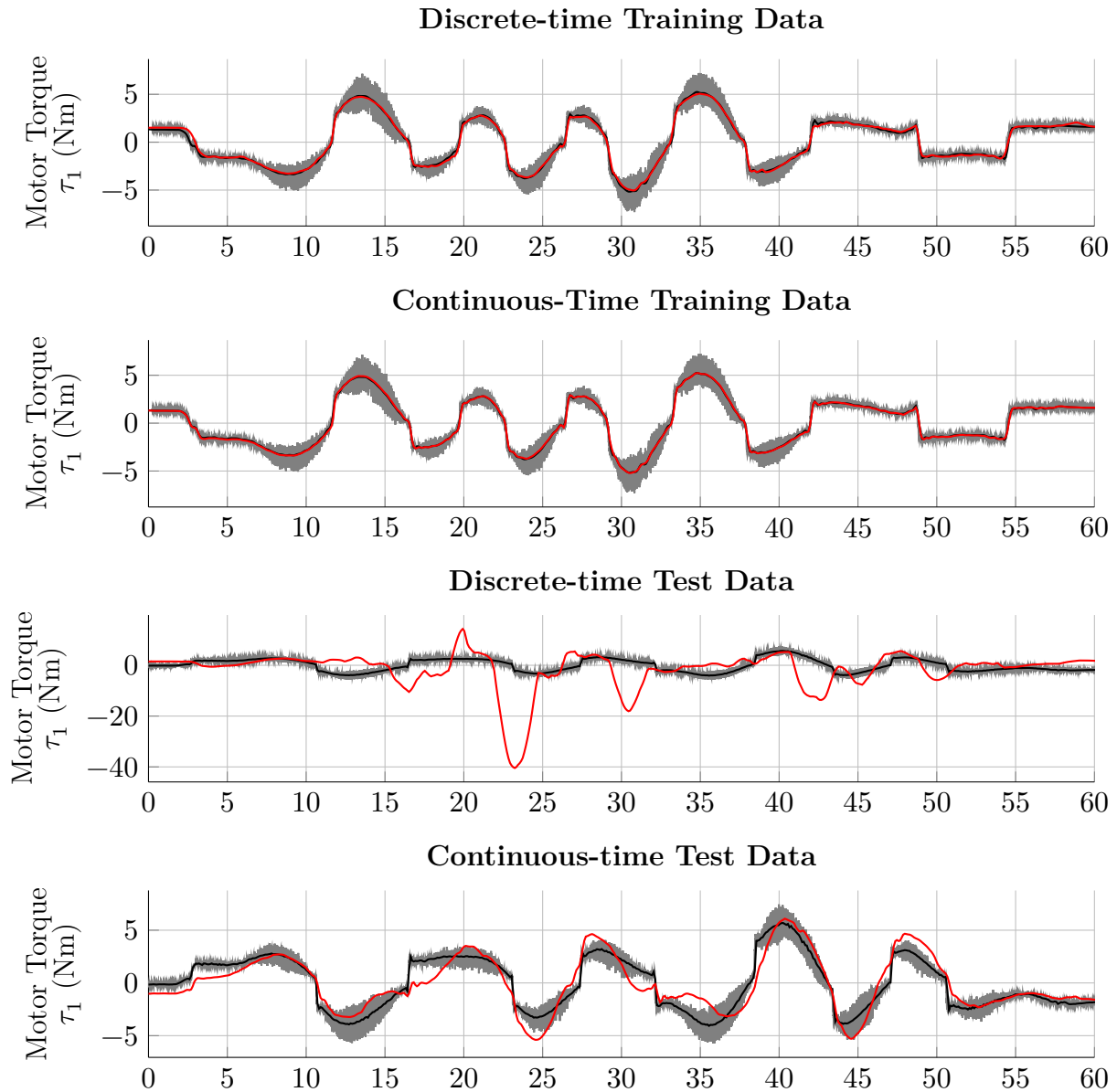


Figure 5.11: Empirical memory efficiency of joint 1. Training and test data are recorded and two models, a continuous-time and a discrete-time model are fitted. The upper figures present the model performance on the training data and the lower figures on the test data. Models and data are directly comparable, see the main paragraph. The continuous-time model can better generalize from training to test data.

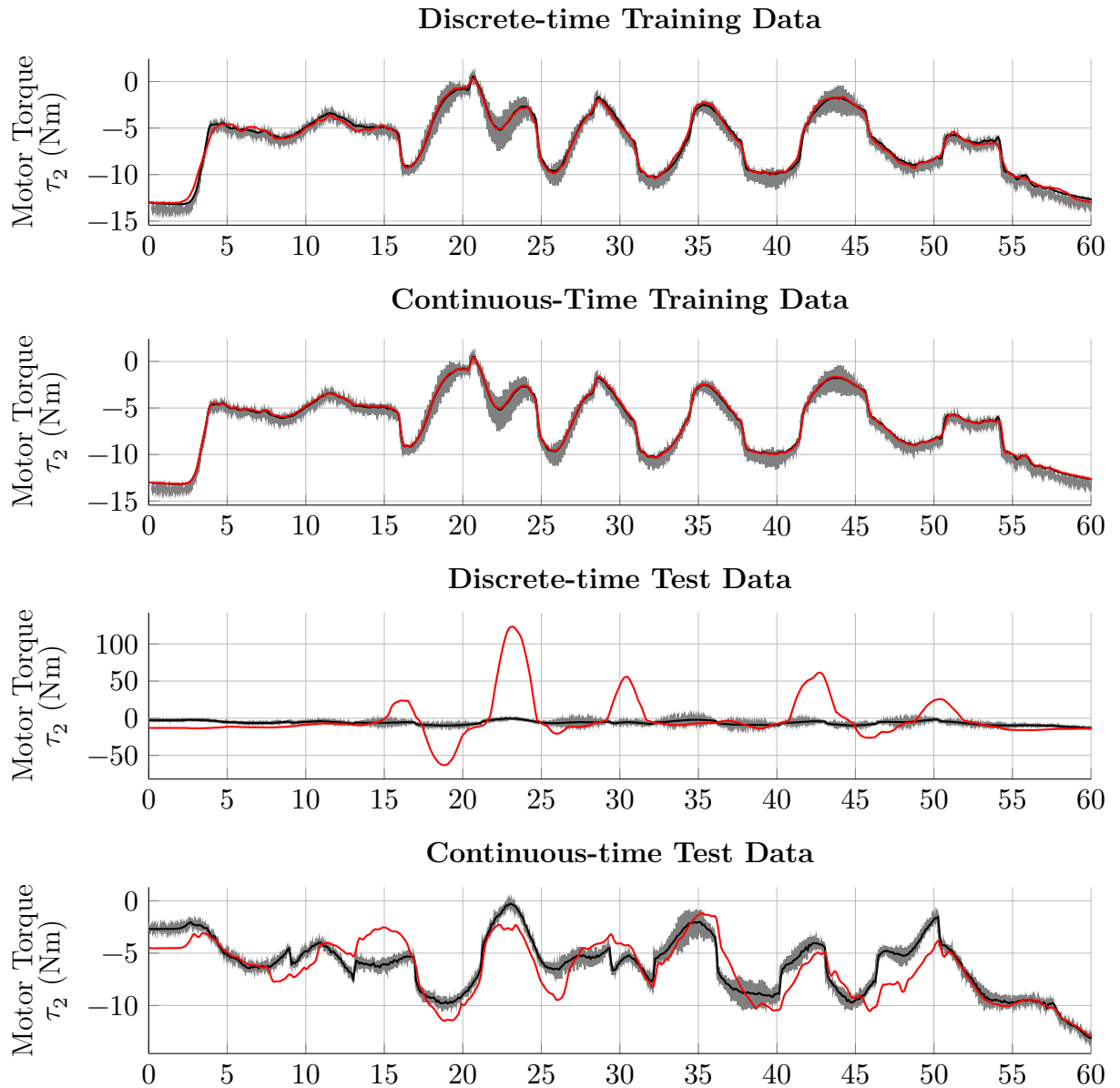


Figure 5.12: Empirical memory efficiency of joint 2. Training and test data are recorded and two models, a continuous-time and a discrete-time model are fitted. The upper figures present the model performance on the training data and the lower figures on the test data. Models and data are directly comparable, see the main paragraph. The continuous-time model can better generalize from training to test data.

The main statement is presented in Fig. 5.11 and Fig. 5.12. On the given example, the continuous-time model with 124231 parameters trained on only 2700 target data points performed better on the test data than the 59496 parameter discrete-time model. Besides embedding the same NN architecture in an Eulerscheme, both results are directly comparable.

Memory efficiency can be backed up from several perspectives beyond the field of system identification. Regarding NODE, both [Ki22] and [Ch18] directly comment on improved memory efficiency using a continuous-time representation. In the field of image recognition, ResNet [He16] is multiple times as efficient as AlexNet [KSH17]. ResNet performs better using a smaller NN, both applied to the same Image Net Benchmark.

5.4 Originality and Background

To clarify intellectual property, background information is given in this section. This section is exclusively for background information and can be skipped for all readers only interested in the methods and results.

In chapter 3, Foundations of Continuous-Time Neural Networks, the section 3.2 to section 3.5 represent state-of-the-art. Parts of the work are literally taken from or inspired by the publications [WDR21; We16; We17; WVR20a]. The bachelor thesis [We16] and master thesis [We17] are original works, including modeling, implementation, and writing. The concept and idea of the Bachelor's and Master's thesis are from Sebastian Stemmler. The publication [WVR20a] is conceived, modeled, implemented, and written by Jonas Weigand. Magnus Volkmann contributed to the physical robot model, proofread the publication, and presented the paper at the conference. Martin Ruskowski provided scientific guidance.

Section 5.1, State Derivative Normalization, is based on the publication [We23]. Gerben Beintema and Jonas Weigand independently and simultaneously identified that state normalization is possible and beneficial. Jonas Weigand et. al. published [WDR21], which was received by the International Journal of Control on 04.01.2021 and published online on 08.11.2021. It contains the idea of Time Domain Normalization, which states that it is possible and beneficial to estimate the model on a normalized time grid. Independently, Gerben Beintema et. al. uploaded the paper [BST22] on 20.04.2022 on arxiv. In searching for co-authors to give an analytic insight into the phenomena, contact with Maarten Schoukens lead to a collaboration. A third Ph.D. candidate has been acquired, Jonas Ulmen. Collaboration stated at the 6th Edition of the Workshop on Nonlinear System Identification Benchmarks, beginning on 25.04.2022. Gerben Beintema, Roland Tóth

and Maarten Schoukens published the paper [BST23] on the International Conference on Learning Systems (ICLR) 2023, and uploaded it on the 23.01.2023 to arxiv. The main publication the section 5.1 is based on the publication [We23], which was submitted to the IEEE Control Systems Letters (L-CSS) Journal on 31.01.2023. Gerben Beintema developed the BLA method and developed the DeepSI Toolbox [BTS21b]. The toolbox was not applied to the results presented in the paper, yet it was an excellent way to double-check the results with an independent code. For example, the observation of the analogy between state derivative normalization and hidden layer initialization methods was a result of comparing both source codes. Jonas Ulmen developed the link to the Contraction Theory. The data-driven cross-validation method, the optimization-based method, implementation, and experiments were provided by Jonas Weigand. [We23] was written by Jonas Weigand, Gerben Beintema, and Jonas Ulmen. Publication supervision is given by Maarten Schoukens, Roland Tóth, Daniel Görjes, and Martin Ruskowski. All authors contributed to the discussions and proofread the publication.

Section 5.2, Stability Constraints for continuous-time Neural Networks, is based on the publication [WDR21]. The publication itself is based on the Ph.D. thesis of Michael Deflorian [De11b]. The Ph.D. thesis of Michael Deflorian was a major source of the master thesis [We17], and out of this, a collaboration between Michael Deflorian and Jonas Weigand emerged. Michael Deflorian already provided the stability constraints method, a code base written in *MATLAB*, and a complete publication draft. However, the proof of the constraints was incorrect. It was based on a coordinate transform, in which an inequality equation using the Lipschitz Norm was incorrectly substituted in an equality equation. Vassilios Yfantis recognized this. Jonas Weigand developed the correct proof. The coordinate transform was a requirement in the version of [De11b] and could not be fixed, as a nonlinear part remaining that could not be estimated. However, lifting the complete proof into a higher dimension resolves the requirement for a coordinate transform, as the higher dimensional space already provides the correct equilibrium point. Besides correcting the proof, Jonas Weigand developed a new code base meeting recent software development requirements and conducted new experiments. The draft was completely rewritten and extended from 8 pages to 32. Jonas Weigand made two further add-ons. First, introducing a penalty and barrier method enables modern deep learning frameworks such as *TensorFlow* [Ma15] or *PyTorch* [Pa19] to incorporate the stability constraints even without a constraint optimization interface. Secondly, weak stability methods are introduced. Weak stability methods preserve the main idea of the stability constraints, yet they are enabled without adaption to a much more flexible range of NN models. However, it should be emphasized that this contribution would not have been possible without the outstanding work of Michael Deflorian, who developed and implemented the stability constraints for continuous-time NN in the first place. Martin Ruskowski gave scientific supervision of the publication. Furthermore, two students' theses contributed to

the continuous-time NN topic, namely Julian Raible and Nico Zantopp.

Section 5.3 presents unpublished work.

6 Comparison of Physical, Data-based, and Hybrid Models

In this chapter, the physical, data-based, and hybrid models are compared on the task of robot identification and control. First, the design of experiments is explained in section 6.1. Then, specific model configurations are presented in section 6.2. Finally, experimental results for all three models are presented and directly compared in section 6.3.

6.1 Design of Experiments

The presented methods are applied on a public benchmark dataset of the industrial robot, see Fig. 6.1. It is a robot with a nominal payload capacity of 300 kg, a weight of 1120 kg, and a reach of 2500 mm. It exhibits 12 states accounting for position and velocity for each of the 6 joints. The number of states increases to 24 states when elastic joints are considered. Joint elasticity models deformations of the gearboxes and introduces an independent position and velocity of each side of the robot joint. The robot encounters backlash in all joints, pose-dependent inertia, pose-dependent gravitational loads, pose-dependent hydraulic forces, pose- and velocity-dependent centripetal and Coriolis forces as well as nonlinear friction, which is temperature-dependent and therefore potentially time-varying.

The task of the development of a sophisticated control is considered. To compare this work with the literature, the training and test trajectories have already been published and described in detail in [We22]. The same public data set is applied for the physical, data-based, and hybrid models.

A sophisticated model-based design of experiments is utilized for parameter identification of the physical model. This specialized data is explained in detail in section 4.2. However, the public data set defined in the section does not rely on a model-based design of experiments. To test for various applications, more general experiment design principles have been utilized, such as consideration of the frequency and the amplitude spectrum.

The experiments are designed as 36 different trajectories with a duration of 60.6 s each. All movements start and end in homing position, see Fig. 6.1. Each trajectory is executed

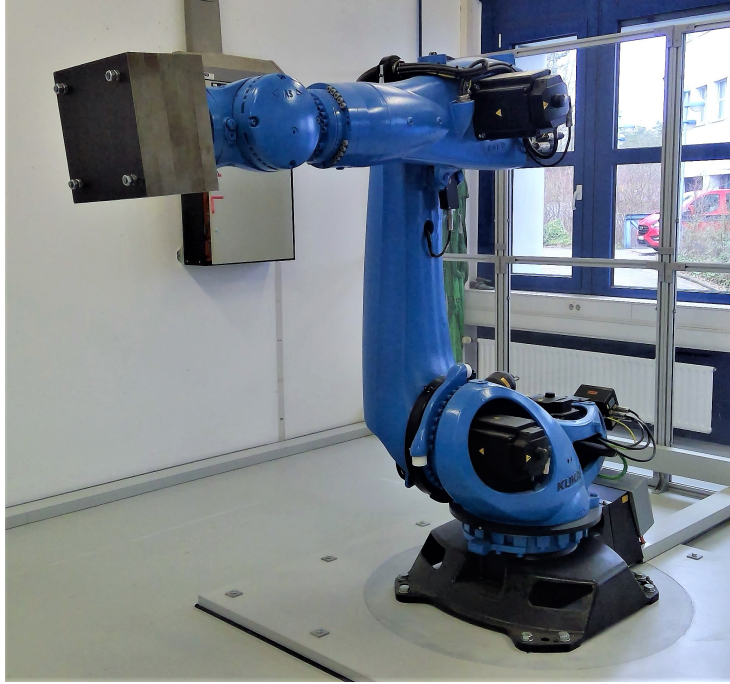


Figure 6.1: Picture of the industrial robot KUKA KR300 R2500 ultra SE at the Chair of Machine Tools and Control Systems, RPTU Kaiserslautern-Landau.

twice so that repeatability is explicitly tested.

The design of experiments for each trajectory is formulated as an optimization problem to incorporate all physical constraints such as position, velocity, acceleration, and jerk limits. Each trajectory is formulated as the sum of sine and cosine functions to ensure continuous differentiability. Furthermore, optimizing for the sine and cosine coefficients $\mathbf{A} \in \mathbb{R}^{V \times N_Q}$, $\mathbf{B} \in \mathbb{R}^{V \times N_Q}$ reduces the number of optimization variables compared to a discrete-time formulation by multiple orders of magnitude. The sum of sine and cosine functions is defined as

$$q_{i,n} = \sum_{v=1}^V \left(\frac{A_{v,i}}{\omega v} \sin(\omega v n) - \frac{B_{v,i}}{\omega v} \cos(\omega v n) \right) \quad (6.1)$$

with a base frequency $\omega \in \mathbb{R}$, a number of $v \in [1, \dots, V]$ sine and cosine functions, and $n \in [1, \dots, N]$ discrete time steps for each joint $i \in [1, \dots, N_Q]$.

For each frequency, $V = 10$ coefficients and $N = 60$ equally spaced time steps are chosen for computing the sine and cosine coefficients. The frequency grid in the design of experiments ranges from the base frequency $\omega = 1/60 \approx 0.0166$ Hz to a maximum frequency of 1 Hz. The sample rate $\Delta t = t_{n+1} - t_n$ and therefore the number of time steps N vary depending on the task, as the trajectory time remains unchanged at $t_N = 60.6$ s. For estimating the sine and cosine coefficients are set to $\Delta t \approx 1$ s, $N = 60$ for fast computation. The high-frequency reference trajectory is set to $\Delta t = 4$ ms, $N = 15147$ to match the feedback controller.

The optimization problem is implemented in MATLAB using *CasADi* with the *IPOpt* solver [An19; MA21]. The optimization problem is given by

$$\mathbf{A}^*, \mathbf{B}^* = \underset{\mathbf{A}, \mathbf{B}}{\operatorname{argmin}} J(\mathbf{q}_n) \quad (6.2a)$$

$$\text{s.t. } \mathbf{q}_{\text{lb},n}^{(m)} \leq \mathbf{q}_n^{(m)} \leq \mathbf{q}_{\text{ub},n}^{(m)} \quad \forall n \in [1, 2, \dots, N], \forall m \in \{0, 1, 2, 3\} \quad (6.2b)$$

$$\mathbf{q}_n^{(m)} = \mathbf{0}, \quad \forall n \in \{1, N\}, \forall m \in \{0, 1, 2, 3\} \quad (6.2c)$$

with the joint upper and lower joints limits $\mathbf{q}_{\text{ub},n}^{(m)}$, $\mathbf{q}_{\text{lb},n}^{(m)}$ for each time step n and each derivative m . Applied limits are documented in the appendix in Tab. 1. The applied objective criterion is empty, e.g., $J(\cdot) = 0$. The optimizer mainly ensures that all trajectory derivatives start and end at zero (6.2c). Each new trajectory is different as the initial values of the sine and cosine coefficients are created randomly. Different objective criteria have been tested, such as maximizing velocity or acceleration, e.g., $J(\mathbf{q}_n) = \sum_{n=1}^N \dot{\mathbf{q}}_n^2$, or a following discontinuous reference, e.g., $J(\mathbf{q}_n) = \sum_{n=1}^N (\mathbf{q}_n - \mathbf{q}_{R,n})^2$. It is easier, to identify a robot model in the high-velocity range only, as some stationary nonlinear effects can almost be omitted. Preliminary examinations revealed that the empty objective function captures both stationary and high-velocity effects well.

The first constraint (6.2b) ensures that limits of the trajectory including the first three derivatives (velocity, acceleration, and jerk) are ensured for all time steps n . The second constraint (6.2c) ensures that all derivatives are zero at the start and the end of each trajectory. Choosing the same initial and final pose for all experiments is optional, yet it ensures that the data can be shuffled and combined without the loss of physical feasibility.

For robot feed-forward controllers, an inverse model of the robot is required, which takes the desired or actual position and its derivatives as input and outputs the feed-forward torque. A time-shift of 4 ms of the input signals is required to compensate for the feedback controller computation time. The input consists of the time-shifted position, velocity, and acceleration, and the outputs are the measured motor torques, as summarized in Tab. 6.1.

The data set is collected in single trajectories, raw data is sampled in 4 ms. For black-box model identification, the data is low pass filtered with a cut-off frequency of 2 Hz and is upsampled to 100 ms.

Three trajectories are provided as test data, each repeated twice to measure repeatability directly. A part of the test data, the robot position, and the motor torques are depicted in Fig. 6.2 and in Fig. 6.3 respectively. The short duration of the test data with only 6 min is chosen to make model effects better visible. In parts of this work, only the first 60 s of the test data are presented for even better visibility.

The training data, which is required for the data-based and hybrid model, consists of 35

Table 6.1: Summary for the inverse model identification datasets.

Inverse Model	
	Joint Position q (deg)
	Joint Velocity \dot{q} (deg/s)
Input	Joint Acceleration \ddot{q} (deg/s ²) (All time-shifted)
	18 Channels
Output	Motor torque τ (Nm)
	6 Channels

trajectories. Each trajectory is repeated twice to account for repeatability. The training data is based on independent reference trajectories. Besides the reference trajectories, the training and test data's definition and post-processing are identical. Extrapolation is tested implicitly, as the training data cannot cover the whole state-space. The training data is presented in Fig. 6.4 and in Fig. 6.5.

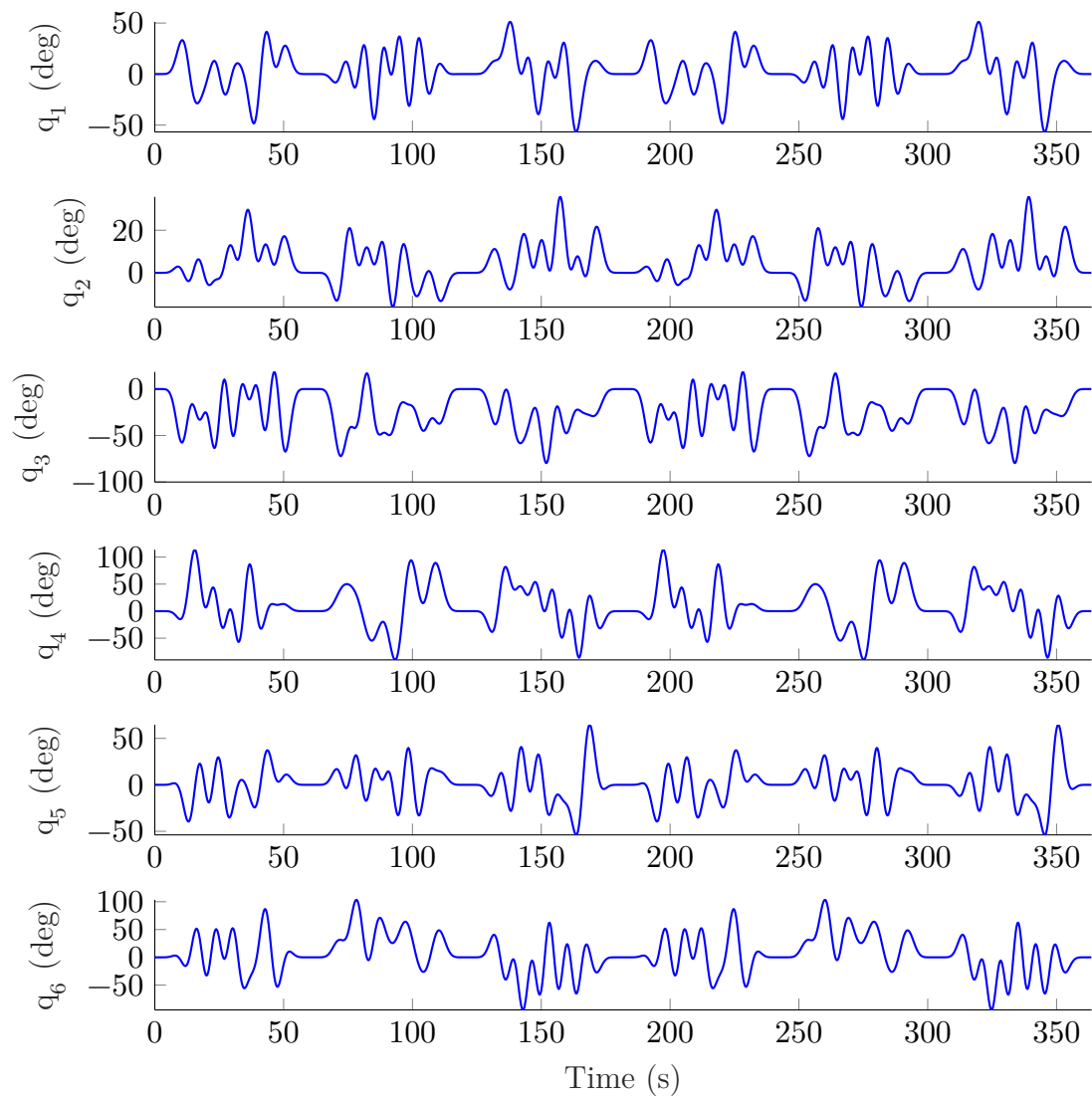


Figure 6.2: Measured joint positions as test data. Each row corresponds to one joint.

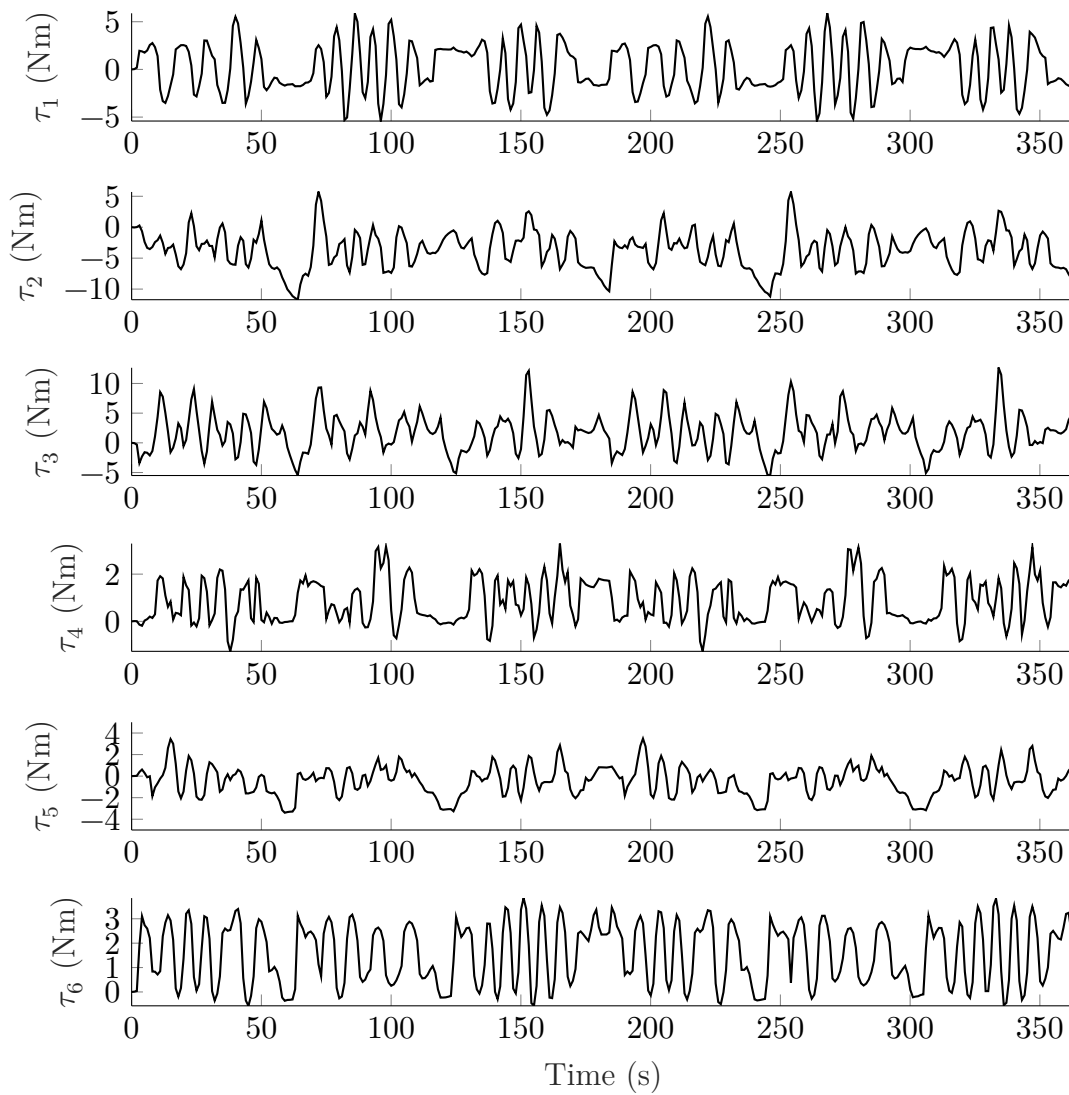


Figure 6.3: Measured motor torques as benchmark test data. Each row corresponds to one joint.

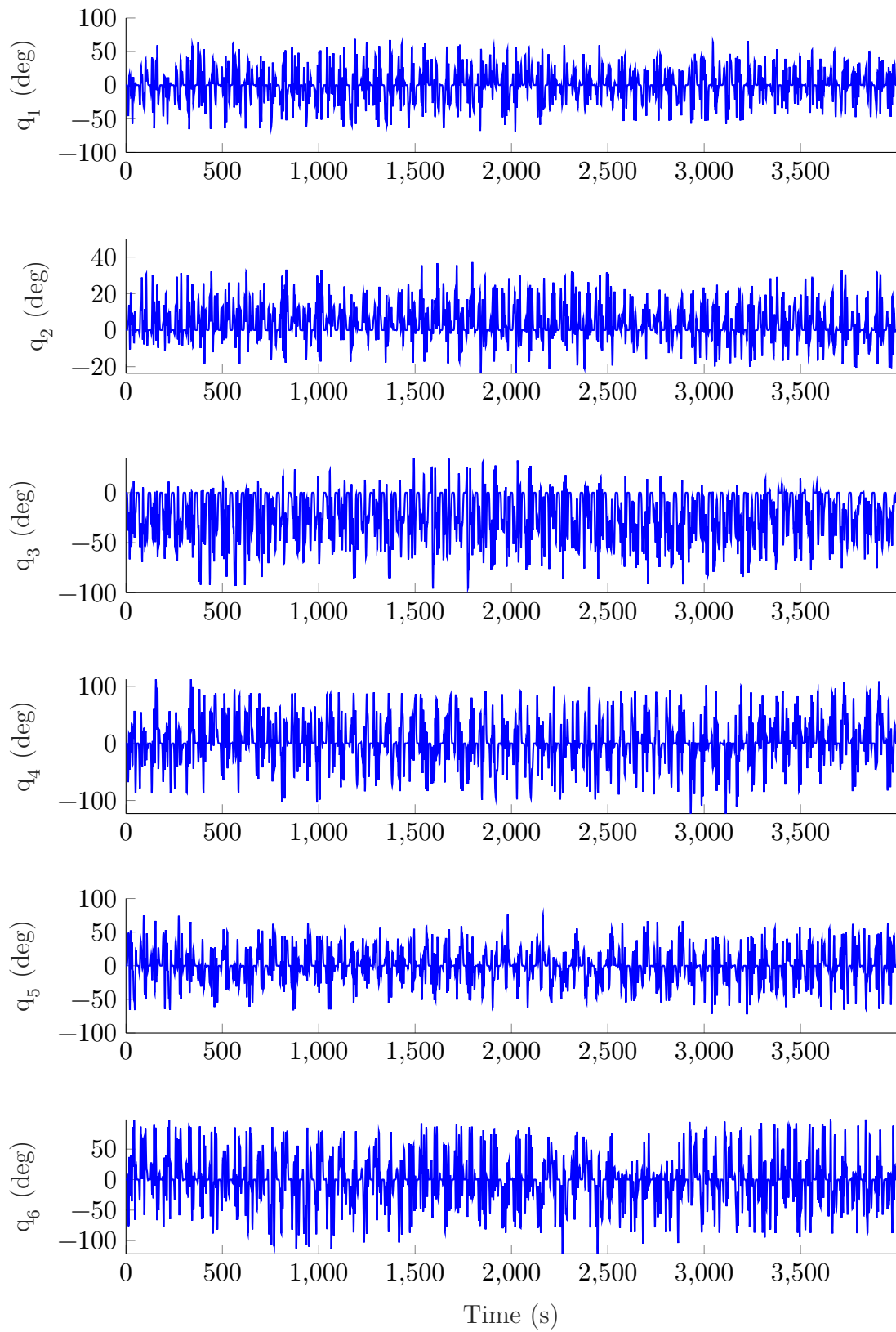


Figure 6.4: Measured joint positions as benchmark training data in deg. Each row corresponds to one joint.

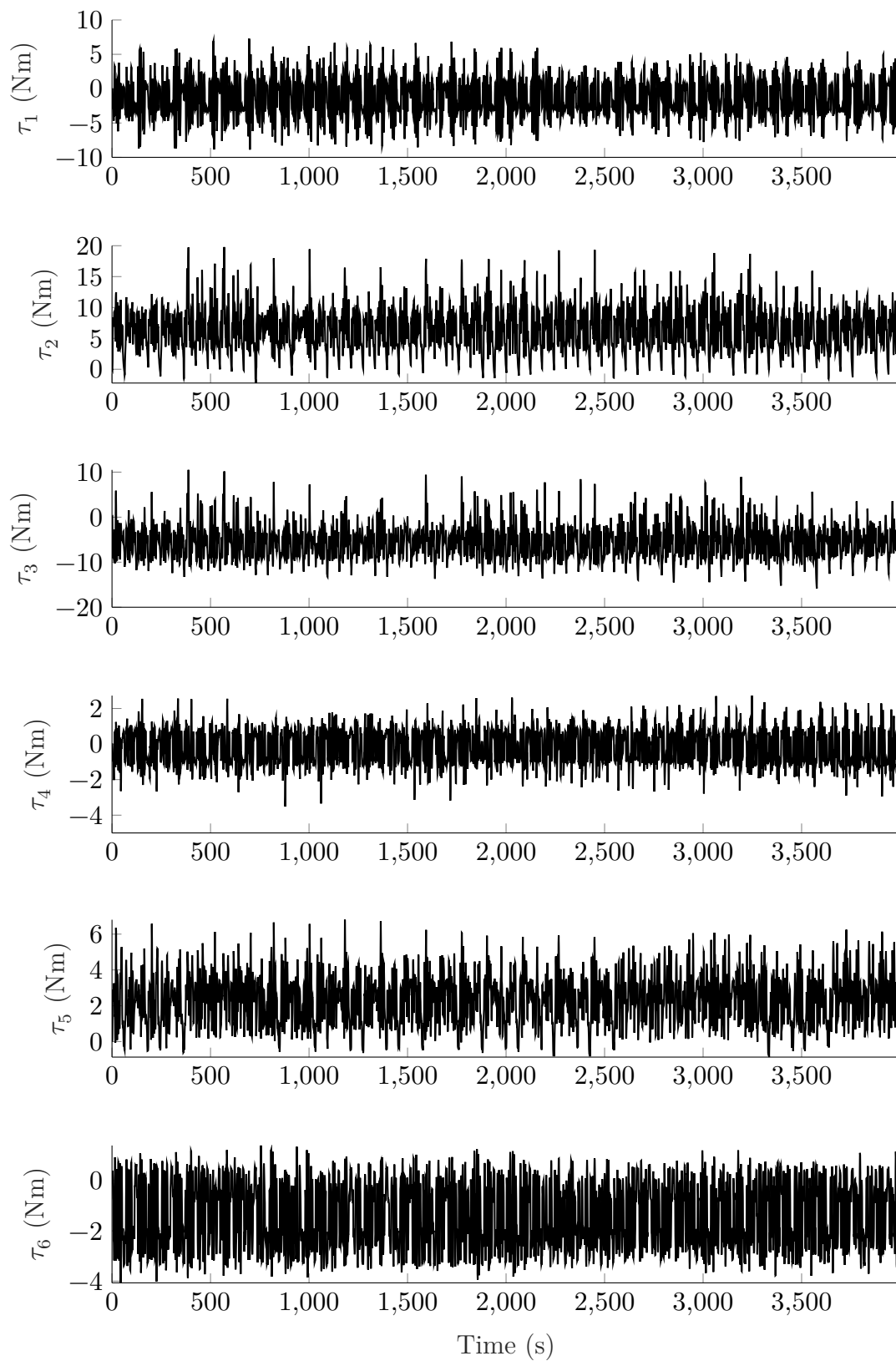


Figure 6.5: Measured motor torques as benchmark training data in Nm. Each row corresponds to one joint.

6.2 Model Configuration

Technical details of the robot, used for all three models, are given in appendix B. In particular, appendix B.1 describes the geometry of the robot as well as the position and velocity limits used in all cases. Appendix B.2 documents the robot controller layout, including hardware details and software architecture. Robot sensor details, including the motor resolvers and SE, are presented in appendix B.3.

6.2.1 Physical Model Configuration

The physical model is defined in chapter 4. The identified robot parameters are given in appendix B.4.

6.2.2 Data-based Model Configuration

The general recommendations for the data-based model are discussed in chapter 3. A summary of the concrete design choices for the best-performing continuous-time NN is given in the following.

A fixed-step Runge-Kutta ODE solver is applied. It delivers a trade-off between accuracy and computational complexity. Furthermore, as a fixed-step solver, the backpropagation algorithm does not require coping with failed steps. The fixed-step solver further encourages a smooth gradient of the loss function for parameters. The nonlinear state-space network is utilized with an enforced linear output. The output torque is an explicit part of the augmented hidden state. The nonlinear network consists of two hidden layers, with LeakyReLU activation functions and 128 hidden units each, to account for the complexity of the physical model.

The model input is extended with the 3rd and 4th derivative of the robot position, $\frac{d^3 q(t)}{dt^3}$ and $\frac{d^4 q(t)}{dt^4}$. This input extension increases the number of model parameters and their model flexibility. The main argument for this extension is the direct comparability of the data-based and physical models. As the flatness-based feed-forward model requires the 3rd and 4th derivative to estimate the joint elasticity, the same input data is provided for the NN. Prior experiments did not present any detrimental model accuracy when prohibiting access for the NN model to this data. Furthermore, the model evaluation does not require many computational resources, especially as the model is continuous-time and in a one-step-ahead prediction setting. Therefore, the additional number of model parameters is not a significant disadvantage. It makes the NN and physical model better comparable. The complete derivative estimation is given by

$$u(t) = \left[q(t), \frac{dq(t)}{dt}, \frac{d^2q(t)}{dt^2}, \frac{d^3q(t)}{dt^3}, \frac{d^4q(t)}{dt^4} \right]^T \quad (6.3a)$$

$$\begin{bmatrix} \frac{d\tau(t)}{dt} \\ \frac{dx_A(t)}{dt} \end{bmatrix} = A_{NN} \begin{bmatrix} \tau(t) \\ x_A(t) \end{bmatrix} + B_{NN} u(t) + \tilde{f}_{NN} \left(\begin{bmatrix} \tau(t) \\ x_A(t) \\ u(t) \end{bmatrix} \right) \quad (6.3b)$$

$$\tau(t) = C_{NN} \begin{bmatrix} \tau(t) \\ x_A(t) \end{bmatrix} + D_{NN} u(t) + \tilde{h}_{NN} \left(\begin{bmatrix} \tau(t) \\ x_A(t) \\ u(t) \end{bmatrix} \right) \quad (6.3c)$$

$$\tilde{f}_{NN} = W_1 \sigma \left(W_2 \sigma \left(W_3 \begin{bmatrix} \tau(t) \\ x_A(t) \\ u(t) \end{bmatrix} + b_3 \right) + b_2 \right) + b_1 \quad (6.3d)$$

$$\tilde{h}_{NN} = W_4 \sigma \left(W_5 \sigma \left(W_6 \begin{bmatrix} \tau(t) \\ x_A(t) \\ u(t) \end{bmatrix} + b_6 \right) + b_5 \right) + b_4 \quad (6.3e)$$

Training data is coherently split into a training and a validation set. Early stopping is engaged if the validation loss cannot be improved for a given number of solver iterations, accounting for high-variance loss functions. Furthermore, the user interface allows for a manual training stop, and a maximum number of overall iterations is defined, too. Training is accelerated using a multistep learning rate scheduler. To ensure appropriate gradients and to use the model's relatively few parameters, Disc-Opt is used instead of Opt-Disc.

A model ensemble with 10 submodels is applied to reduce outliers and decrease output variance. It also improves the repeatability of the experiments. Output is composed using the mean function.

The data is directly applied for initial hidden state estimation. Augmented states are initialized as zero. As a result of the one-step-ahead prediction setting, augmented states do not develop a dynamic over time.

MSE is applied as the main loss function. The evaluation metric is RMSE. Neither a stability penalty function nor a DAE penalty function is utilized.

6.2.3 Hybrid Model Configuration

According to [Sj95], three types of models can be distinguished in system identification (quoted literally):

- **White-Box Models:** This is the case when a model is perfectly known; it has been possible to construct it entirely from prior knowledge and physical insight.
- **Gray-Box Models:** This is the case when some physical insight is available but several parameters remain to be determined from observed data. It is useful to consider two subcases:
 - **Physical Modeling:** A model structure can be built on physical grounds, which has a certain number of parameters to be estimated from data. This could, e.g., be a state-space model of given order and structure.
 - **Semi-physical Modeling:** Physical insight is used to suggest certain nonlinear combinations of measured data signal. These new signals are then subjected to model structures of black box character.
- **Black-Box:** No physical insight is available or used but the chosen model structure belongs to families that are known to have good flexibility and have been “successful in the past”.

Using this definition, the physical robot model explained in chapter 4 is a gray-box physical model, as its parameters are derived from data. The classification of the data-based model depends on the application. Regarding the CTS and the EMPS benchmark, the NN’s ODE structure itself is related to the expected physical models. Physical insight is used for the choice of the input and output signals. Therefore, it can be classified as a gray-box semi-physical model. Nevertheless, the continuous-time NN is a very flexible structure. As explained in section 5.3.2, the feed-forward robot benchmark is *not* expected to be a ODE structure. Regarding the robot benchmark, the data-based model can be classified as a black-box model, using a ODE structure which contradicts the expected physical model.

This work defines hybrid models as models which combine different submodel types. Another formulation is that hybrid models combine expert knowledge with data-driven methods, inserting different degrees of physical knowledge. For example, a gray-box physical model and a black-box model are combined in the robot case.

As a further distinction, hybrid models can be separated into three block structures. Note that all the following structures are capable of a multi-output, multi-model, or recursive formulation as discussed in section 3.5.5. Furthermore, all structures are capable of model ensembles, see section 3.4.2. In both cases, a submodel of the hybrid model can contain several layers of subsequent submodels. Finally, for advanced readers, an excellent intro-

duction to several structures of nonlinear, block-oriented models is given in [ST17]. A discussion of parallel, serial, and submodel structures is presented in the following.

- **Parallel structure.** The structure is depicted in Fig. 6.6. The physical and data-based models are configured in parallel, receiving the same input $u(t)$, combined at the output $y(t)$. Full universal function approximation capabilities are given using a sufficiently flexible data-based model. Any NN structure with at least one hidden layer and a nonlinear activation function fulfills this requirement [Ha19; ST98]. No parameters are shared. Consequently, full independence of both models can be acquired, depending on the mode. Both models can be designed independently in the development process, possibly using different software packages. For example, the robot's physical inverse model parameters are identified in *MATLAB* using a global optimization toolbox. In contrast, the data-based model is written in Python using the *PyTorch* library. For software developers, this modularity is a major advantage. Validation and testing are facilitated as both models share identical input and output definitions. Two training modes can be distinguished.

→ In a **surrogate mode**, the data-based model is trained to predict the measured output $y(t)$. Full independence of the data-based and physical model is gained.

→ In an **error correction mode**, the data-based model is trained to reduce the remaining prediction error of the physical model, $y(t) - y_{phys}(t)$. Error correction mode requires a sequential software development strategy. The physical model is a prerequisite for training the data-based model. Furthermore, the error signal $y(t) - y_{phys}(t)$ can be more complicated to predict than the output signal $y(t)$, as it contains additional model errors. The amplitude of $y(t) - y_{phys}(t)$ is typically smaller than $y(t)$, and thus data normalization is required. Nevertheless, the error correction mode's advantage is obtaining an even better hybrid model performance. Errors in the physical model can be diminished using the universal function approximation.

Furthermore, a parallel structure extends safety features. For example, the physical model extrapolation is often explainable, based on first principles. On the other hand, a safe extrapolation is cumbersome to guarantee in the data-based model. In the parallel structure, extrapolation of the data-based model can be explicitly quantified by comparing both models. Furthermore, the contribution of uncertain extrapolation can be reduced. Therefore, the parallel structure in a surrogate mode is recommended and applied.

- **Serial structure.** The serial structure is pictured in Fig. 6.7. An expert model computes the input for a data-based model or vice versa. The parameters and structure of both models are not shared. Designing the physical model downstream

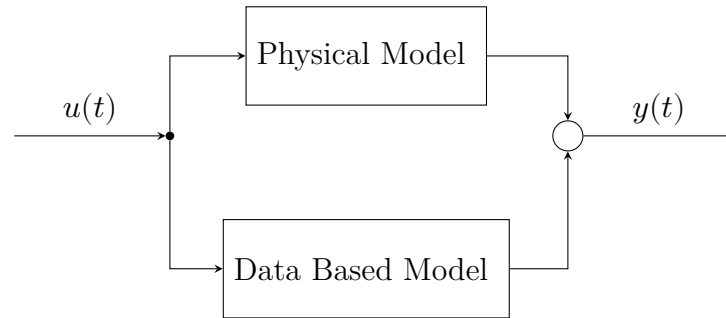


Figure 6.6: Hybrid model in parallel structure.

often increases the model safety concerning extrapolation so that all outputs can be explicitly checked and limited by the developer. The expense of development and validation depends on the intermediate state $z(t)$ definition and whether it is human-interpretable. On the one hand, it is often a requirement for physical models to utilize human-interpretable inputs. Interpretable inputs enable independent development and validation of both models. On the other hand, human-interpretable states can create an information bottleneck, making universal function approximation capabilities cumbersome. The universal function approximation of the data-based submodel cannot guarantee the universal function approximation capabilities of the hybrid model. The internal estimations of the physical model can restrict the representation capabilities. For example, suppose a dominant physical effect is not modeled, and the required information is not included in the intermediate state $z(t)$. In that case, the hybrid model cannot represent the impact of this physical effect. As another example, assume the physical model is downstream, and its output range and change rate are limited. This case prohibits universal function approximation capabilities.

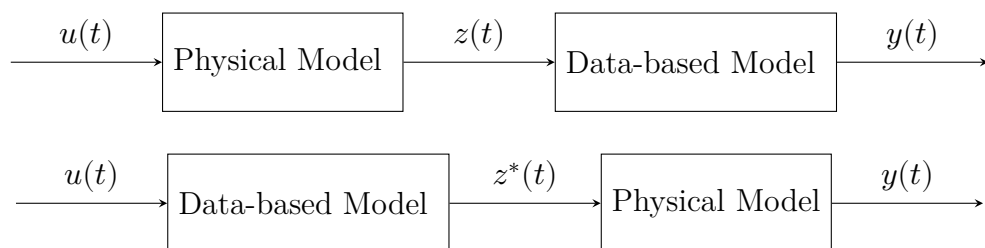


Figure 6.7: Hybrid model in serial structure. Above: Physical first, then data-based model. Below: Data-based first, then physical model.

- **Submodel structure.** This structure is presented in Fig. 6.8. The data-based model is a submodel and is embedded in a larger, physical model. The structure of both models is directly coupled. For example, the data-based model can represent a friction model embedded in a larger dynamic process model. As another example, the data-based model can be applied to perform object detection and passes this information to a controller for autonomous driving. Universal function approximation capabilities are not given per se, as the universal data-based model is strictly

embedded. Independent validation and testing are not possible in general. Validation of the data-based model requires internal knowledge (or even measurements) of internal states. Independent software development is cumbersome. Training of the submodel contains the risk of unintentional modeling. For example, assume the data-based model is contemplated to model the friction of a dynamical process and not modeled elastic behavior leads to significant deviations from the model predictions. If the loss function only accounts for the overall model prediction accuracy, it will encourage the data-based model, intended for friction, to model elasticity. Additionally, it is difficult to predict how the failure of the submodel will influence overall model performance. This structure is only recommended if the data-based model can be trained and validated independently of its embedding.

Regarding real world applications, many algorithms are “hidden” submodel structures. For example, data acquisition, command execution, database access, web interface, and software error management are typically human-developed, pure white-box algorithms. The model scope is decisive. It depends on the definition of the input and output signals whether the superordinate submodel structure is explicitly included (“visible”) or excluded (“hidden”).

The opposite that the physical model is a submodel of the data-based model is rare. This infusion of expert knowledge can strengthen the training process regarding training time, memory and parameter efficiency, or data efficiency. Nevertheless, this case is challenging for analytical development and validation. The impact of the embedded physical model is hard to predict. Additionally, the physical model requires additional development effort.

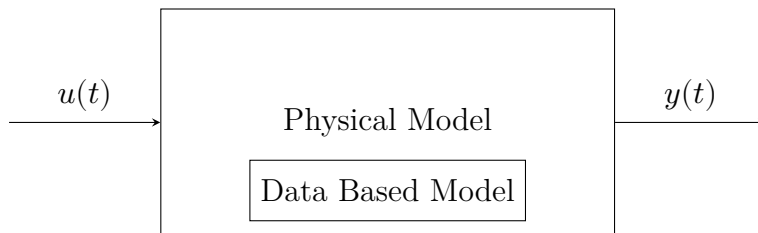


Figure 6.8: Hybrid submodel structure.

The implemented hybrid structure is defined as follows. A parallel structure is chosen, as it enables universal function approximation capabilities and direct comparison of the data-based and physical model. The data-based model is the continuous-time model defined in section 6.2.2 and the physical model is applied as derived in section 4.1. A surrogate mode rather than an error-correction mode is utilized. Both models contribute to the feed-forward torque $\tau_{FF}(t)$ as defined by a blending parameter $\gamma_B \in \mathbb{R}$, with $0 \leq \gamma_B \leq 1$. In the case of $\gamma_B = 1$, the hybrid model reduces to the flatness-based control. In the case of $\gamma_B = 0$, it simplifies to the data-based feed-forward control. An equal contribution with

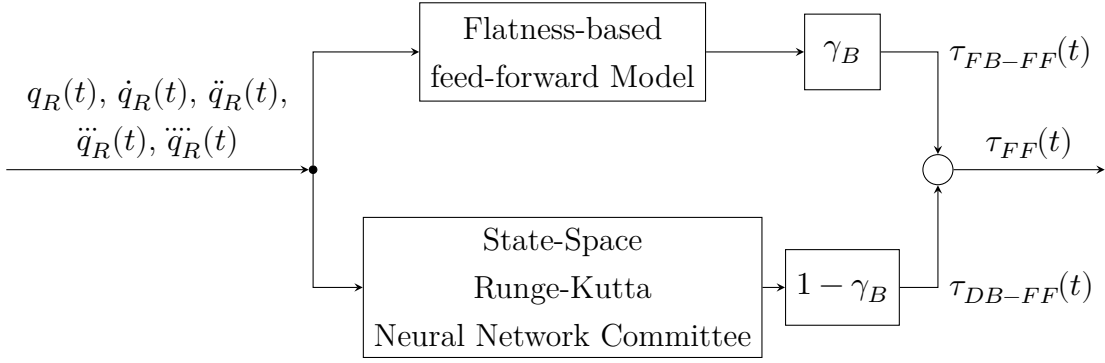


Figure 6.9: Implemented hybrid model in parallel structure.

$\gamma_B = 0.5$ is applied in the succeeding experiments. The implemented hybrid structure is presented in Fig. 6.9.

6.3 Experimental Results

6.3.1 Comparison of Robot Model Accuracy

The advanced physical model is applied to the benchmark. Fig. 6.10 shows the motor torque from the feed-forward control, compared to the measured torque. The measured torque is given in high-resolution measurement (gray line) and as a low-pass filtered signal (black line). The filter is a butterworth filter with order 4 and cut-off-frequency of 5 Hz [MA21]. Especially during stand-still or low-velocity periods of the trajectory (less than ≈ 7.5 s and more than ≈ 52.5 s) model errors are visible. As the robot is not or only slowly moving, these do not lead to significant position errors. From an application point of view, stand-still or low-velocity periods are seldom relevant for dynamic path accuracy.

Fig. 6.11 presents the final model accuracy on the test data of the data-based model. The torque prediction is, in most cases, smaller than the high-frequency bandwidth. Even the standstill phases of the robot, although not required for robot control, can be predicted accurately. The torque prediction accuracy of the hybrid surrogate model is depicted in Fig. 6.12. The performance is in between the pure physical and the pure data-based model, see Fig. 6.10 and Fig. 6.11 respectively.

All in all, the data-based model delivers the best torque prediction accuracy on test data. This indicates that overfitting of the model to the training data is limited to a reasonable level and the underlying continuous-time dynamics are accurately identified. The performance of the data-based model is followed by the hybrid model, and the lowest torque prediction accuracy of the three models is generated by the physical model. Yet, the improvements of the physical model, comparing [WGR20] and this thesis, are significant.

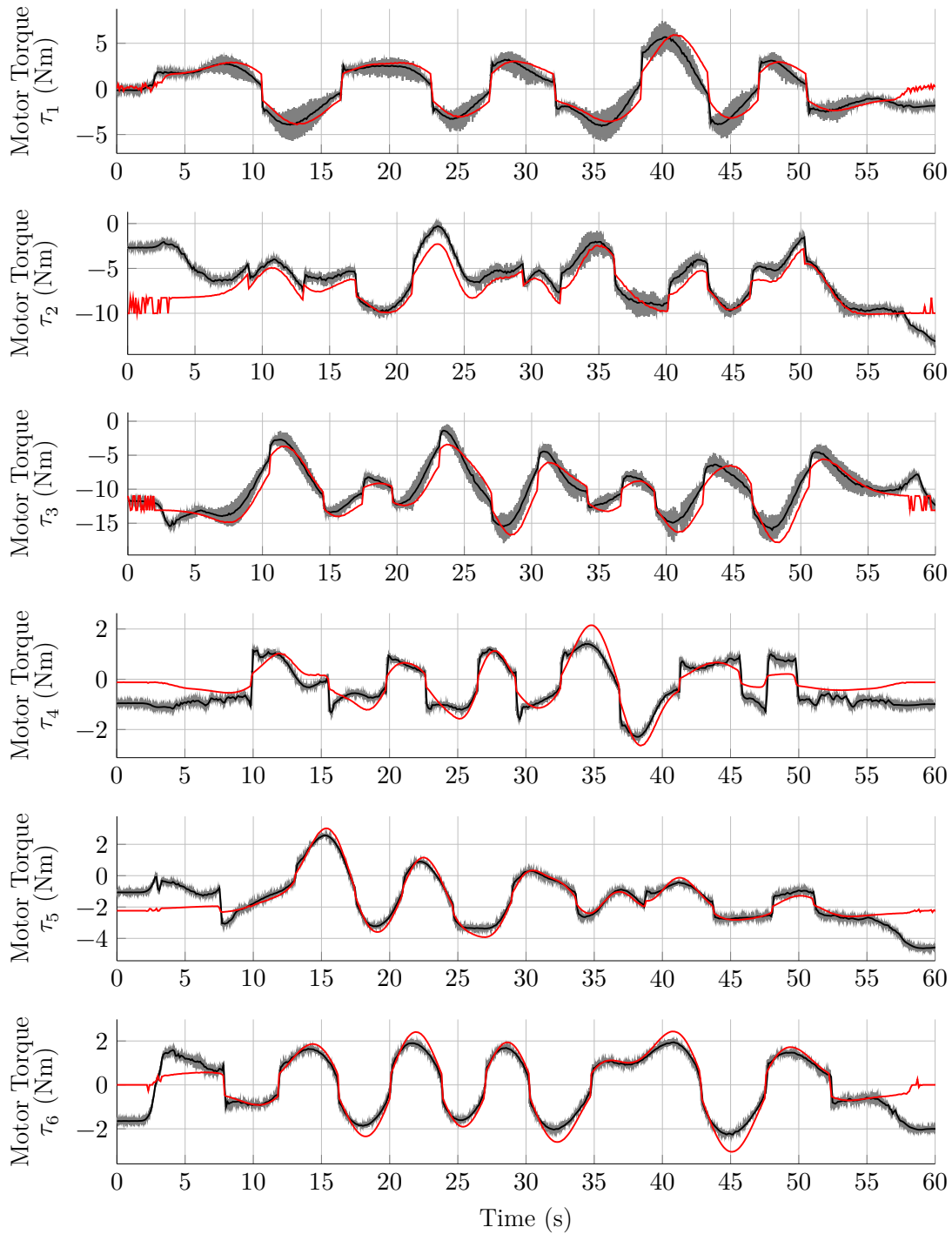


Figure 6.10: Performance of the linear baseline on the benchmark test data. Physical model parameters have been adapted to independent experiments. The torque estimation is based on the actual robot position. Red: Flatness-based model prediction. Gray: Motor torque measurements. Black: Filtered motor torque measurements.

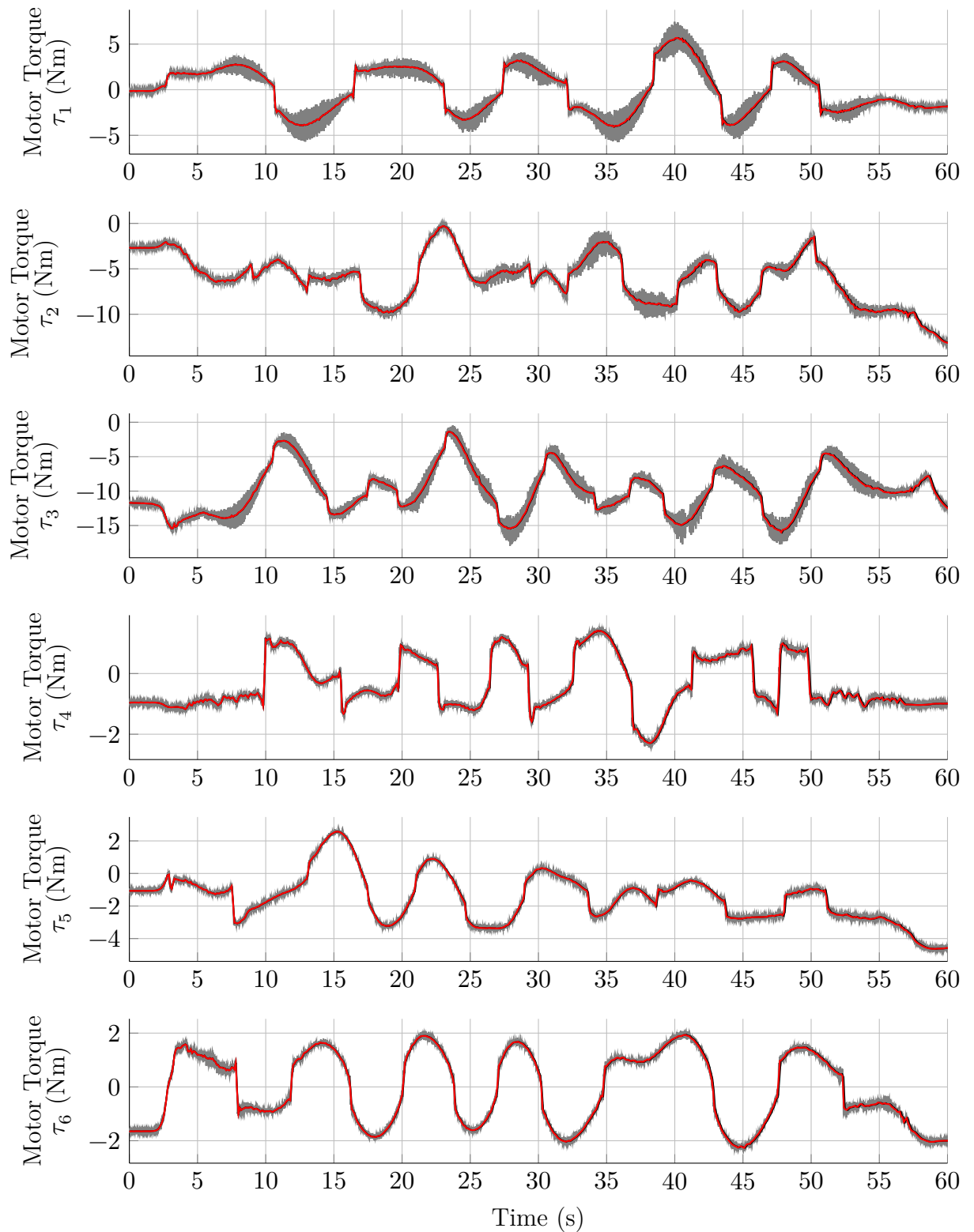


Figure 6.11: Performance of the continuous-time advanced RKNN model on the benchmark test data. Trained on many data points. Red: Model prediction. Gray: Motor torque measurements. Black: Filtered motor torque measurements.

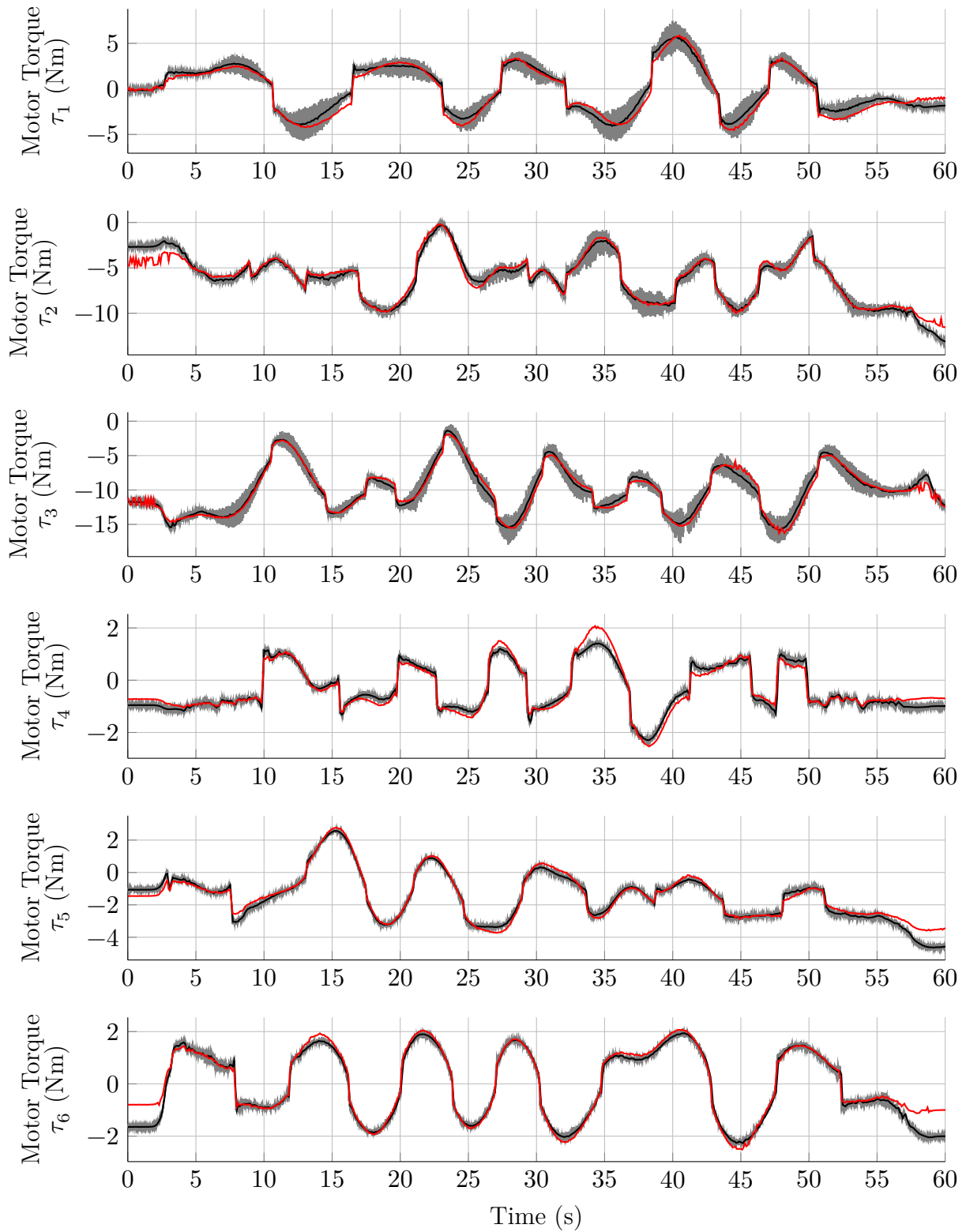


Figure 6.12: Performance of the hybrid model on the benchmark test data. The data-based model is trained on many data points. Red: Model prediction. Gray: Motor torque measurements. Black: Filtered motor torque measurements.

6.3.2 Comparison of Robot Path Accuracy

The robot feed-forward control task is to improve the robot path accuracy. Figures 6.14, 6.13 and 6.15 present the results on the test data for the physical, data-based, and hybrid models respectively. On the left side, the reference and actual robot position is displayed, and on the right side, the absolute position error is presented. On all joints and in most parts of the presented trajectory, the final path accuracy, in terms of joint accuracy, is less than ± 0.1 deg. In direct comparison, figures 6.16 and 6.17 present the absolute position and absolute position error of all three methods in the same figure respectively.

The results indicate a high similarity between the three methods regarding path accuracy. The main difference occurs on joint 2, possibly due to the HWC modeling. The experiments further indicate that the path deviation is systematic and deterministic. Given the difference of the methods in the model identification, section 6.3.1, the feed-forward method is an unlikely source of this similarity. It follows that the remaining path accuracy deviation is a consequence of other components in the robot control architecture. This suggests, that the closed-loop position error cannot be improved by models, e.g., the source of error is only a small fraction in the feed-forward model with a large fraction in the feedback structure, or it depends on other data acquisition effects.

All in all, no significant difference in path accuracy using the data-based, physical, and hybrid model can be observed. Given the different torque prediction accuracies in section 6.3.1, this outcome indicates that all three model torque predictions are sufficiently precise to be no significant source of path deviations. The remaining path deviations are out of scope of this thesis.

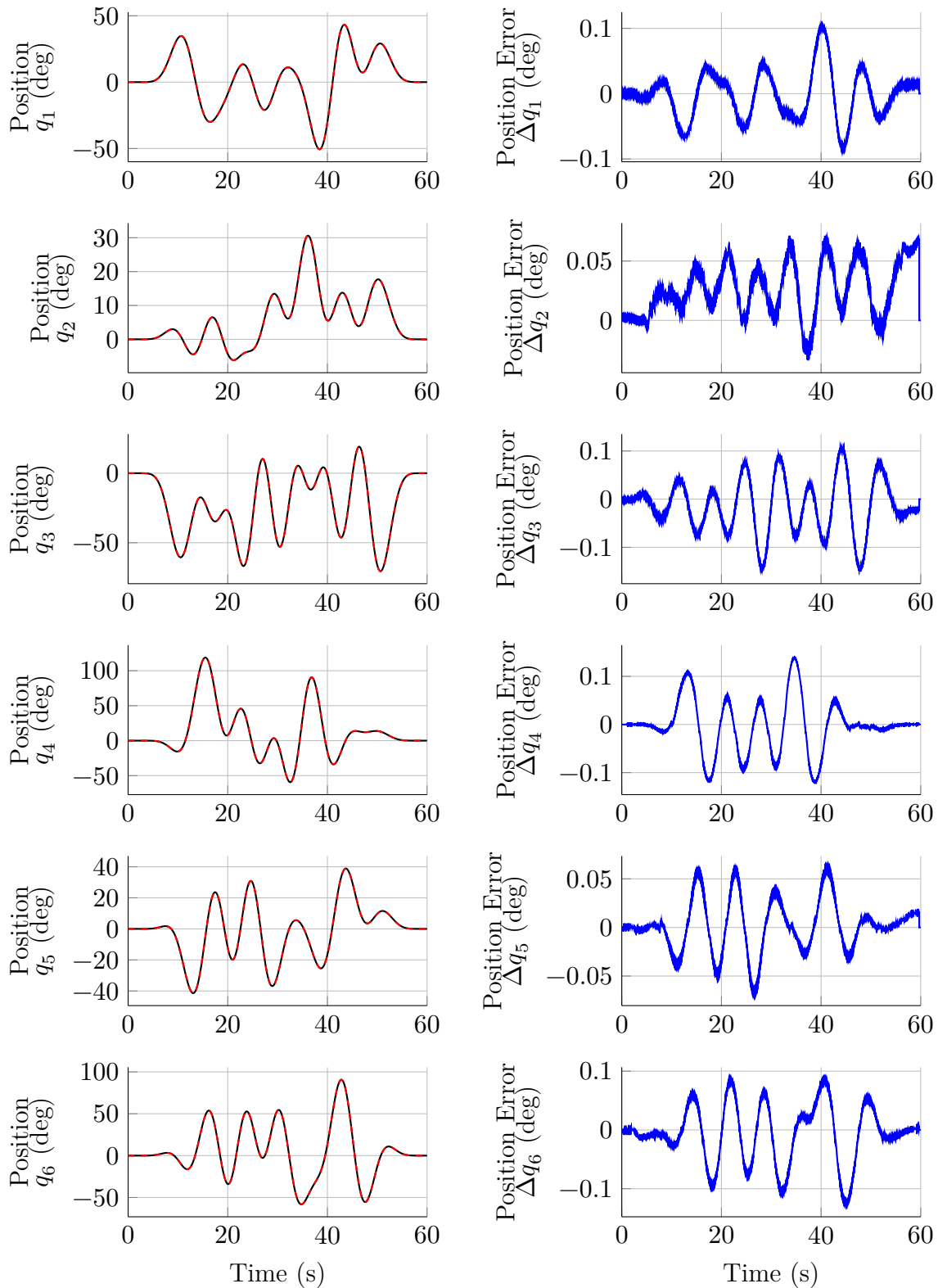


Figure 6.13: Robot accuracy using the continuous-time State-Space Runge-Kutta. It is trained on the 5 s to 45 s of the public training data. Each row represents one joint. Left: Reference position (black line) and measured position (red dotted line). Right: position error of each joint (blue line).

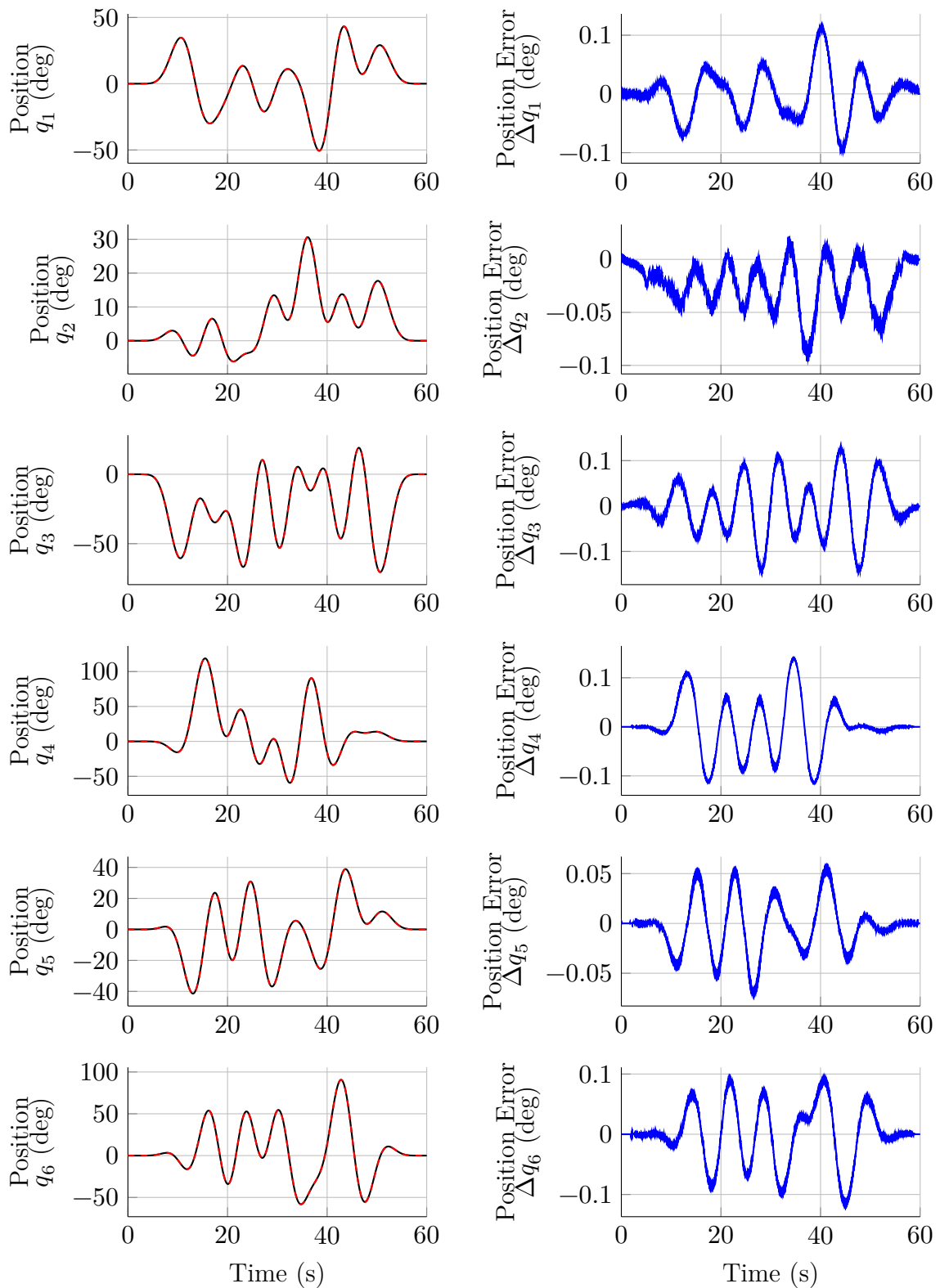


Figure 6.14: Robot accuracy using the flatness-based feed-forward control. Each row represents one joint. Left: Reference position (black line) and measured position (red dotted line). Right: position error of each joint (blue line).

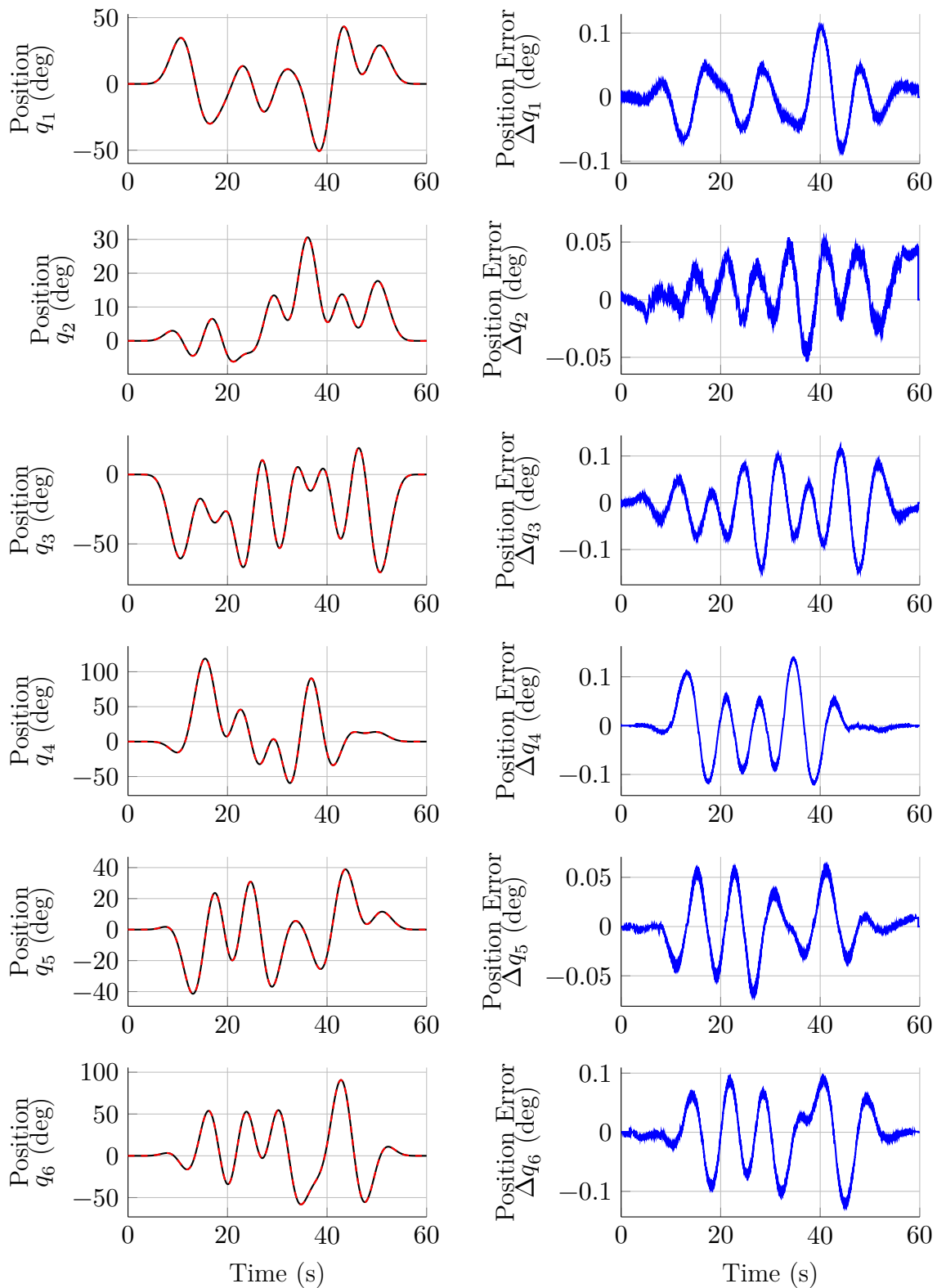


Figure 6.15: Robot accuracy using the hybrid model. The data-based submodel is trained on the 5s to 45s of the public training data. Each row represents one joint. Left: Reference position (black line) and measured position (red dotted line). Right: position error of each joint (blue line).

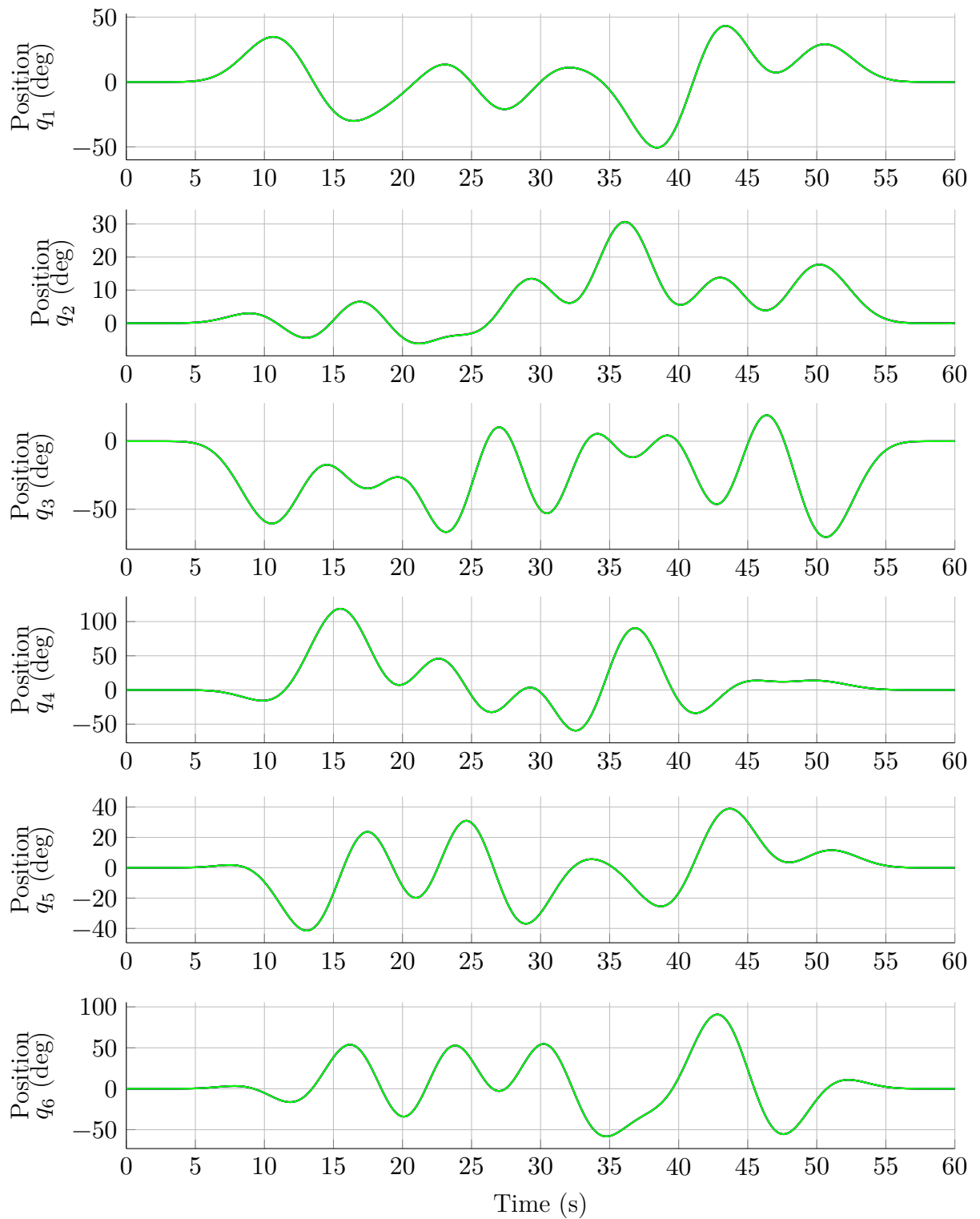


Figure 6.16: Robot accuracy comparison of the physical, data-based, and hybrid model. The data-based submodel is trained on the 5 s to 45 s of the public training data. Each row represents one joint. Red: physical model, green: data-based model, blue: hybrid model.

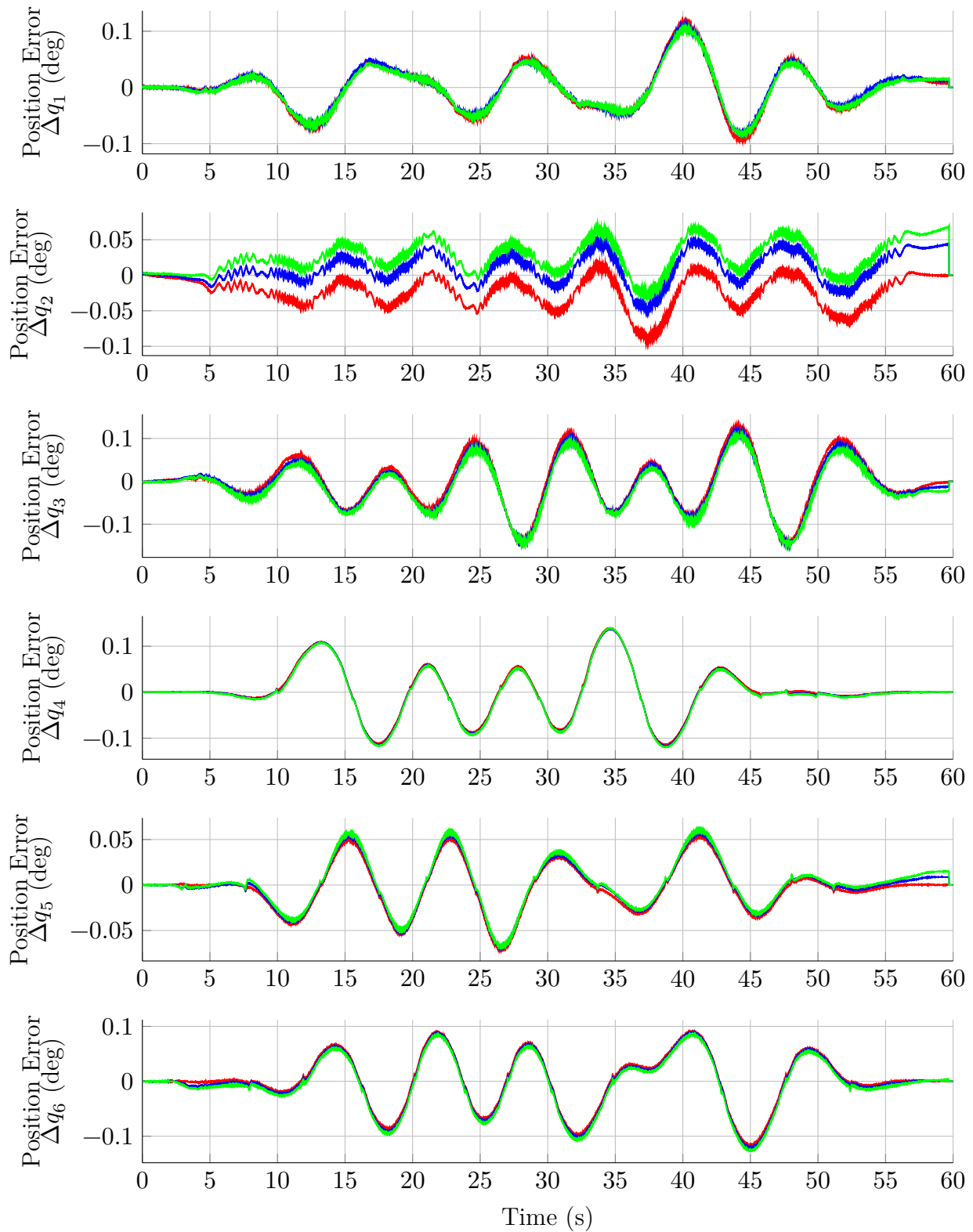


Figure 6.17: Robot accuracy error comparison of the physical, data-based, and hybrid model. The data-based submodel is trained on the 5 s to 45 s of the public training data. Each row represents one joint. Red: physical model, green: data-based model, blue: hybrid model.

6.3.3 Comparison of Robot Model Safety

This work defines robot safety as the risk of damaging the industrial robot itself, the surrounding inventory, or the injury of persons. Most safety issues are solved with additional measures, for example, the risk of injury of persons is excluded using a fence and locking the robot cell door when the robot is energized. The robot cell is depicted in Fig. 6.18. Robot cell management is exclusively handled by safety-certified Programmable Logic Controller (PLC).

Damaging surroundings are ruled out by the kinematic pose limitations of the robot. Damage to the robot is, in the last instance, handled by an error monitoring module, which can trigger an emergency stop in critical cases. This error monitoring module permanently checks for

- joint pose, velocity, torque, and torque derivative limitations,
- redundancy checks, for example, using the motor resolver and SE,
- redundancy of modules, for example, estimation of the feed-forward torque using different models,
- smoothness and derivatives of all sensor signals, feed-forward channels, and reference trajectory,
- error messages of all submodules,
- external client connection (OPC-UA), manual movement interface, and state machines of all joints,
- and messages from safety PLC.

This list amounts to 37 error cases for each joint and 222 in total. The cases account only for errors that trigger an emergency stop, not warnings. Operation technology warnings and errors are additional. The error monitoring is designed to avoid false negatives strictly. It is not prioritized for a detailed root-cause analysis. For instance, a model error would likely trigger several error messages, such as model redundancy check fail, torque derivative overstep, and lack of feed-forward smoothness.

However, it is desired that these hard emergency stop limits are not enabled in the first case. Consequently, robot model safety is discussed in the following, defined as the risk of damaging the robot and its surroundings or injure of persons, explicitly caused by the generalization capabilities of the feed-forward control. The following discussion is about preventing critical model predictions in the first place without triggering the error monitoring module.

From an engineering perspective, the model safety of the physical feed-forward control

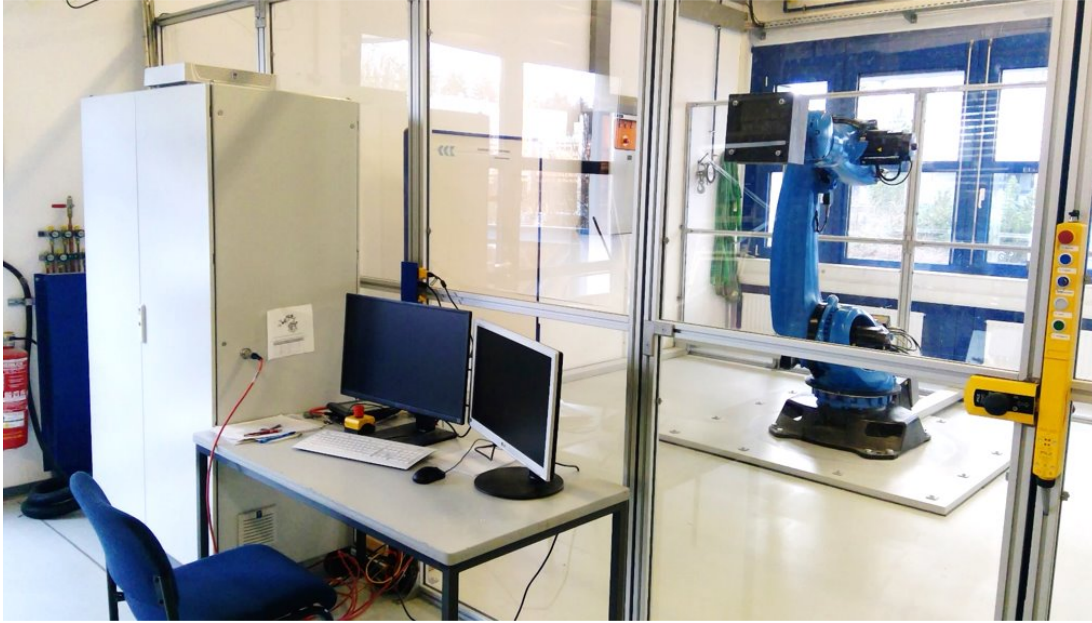


Figure 6.18: Picture of the robot machining cell.

is beneficial, as the model is completely explainable. Caution is required for handling non-flat effects, such as Coulomb friction of backlash. The data-based module, as trained NN is critical. Recall the very good performance of the tapped-delay MLP network on the training data and the critical performance on the test data in Fig. 5.11 and Fig. 5.12. Applying such a data-based module would very likely trigger error monitoring as the model is overfitted to the training data and generalizes poorly. A proper training pipeline validation and test are essential for data-based models.

Additionally, for hybrid models, the contributions of the data-based module and the physical module can be explicitly quantified using a parallel hybrid model architecture. Using the assumption that the physical module is quantifiable entirely, the less the overall hybrid output is contributed by the data-based module, the better the model safety assessment.

However, the data-based module can significantly reduce the required engineering effort to achieve the presented accuracy. A very similar model architecture was applied in chapter 5 on two completely different real world applications, the CTS and the EMPS benchmark, achieving results comparable to literature beyond NN. In conclusion, given data of similar industrial robots, the NN can easily adapt to other model variants.

In combination with that argument, the less the data-based module contributes to the prediction, the better the safety assessment. It can be desirable to constrain the data-based model output. For discrete-time models, only an output constraint can be applied. Using continuous-time NN, the dynamics of the hidden state and its derivative can be limited directly in a continuously differentiable manner

$$\frac{dx(t)}{dt} = \lambda_B \tanh(f_{NN}(x(t), u(t))) \quad (6.4)$$

with a proper boundary vector $\lambda_B \in \mathbb{R}^{N_x}$.

All in all, the physical model offers the best model safety as it is entirely explainable. The hybrid model encounters a reduced safety evaluation, depending on the fraction of torque contribution of the data-based model. The data-based model is not completely physically explainable, therefore it requires additional caution. However, the torque output range of the data-based model can be explicitly limited using (6.4). In addition, as presented in section 5.3, continuous-time NN can better generalize on robot identification tasks than discrete-time NN, as the continuous-time model is closer to the underlying continuous-time dynamics.

As explained in this section, all three models are embedded in a complete safety concept. Therefore, even if the models would cause critical predictions, the impact of model predictions will be caught and limited by the downstream safety concept. As a result, all three model types are safe to operate. Yet, it is important to discuss which model type can potentially produce critical predictions in the first place.

7 Summary and Outlook

A custom robot controller enabled the implementation of sophisticated feedback and feed-forward control. To engage full access to the robot control, from the low-level motor control and sensor integration to high-level path planning and simulation, a new robot control based on components by *B&R Automation* has been developed from scratch, see appendix B.

To account for gearbox deformations in the feedback control, high accuracy Secondary Encoders have been applied and integrated into the control loop. Subsequently, the robot path accuracy could be improved significantly using this controller. For additional material on this topic, please refer to the publication [WGR20] and student thesis [Be19; Di19; He19].

A decent feed-forward control improves path accuracy in two ways: First, it can compensate for deterministic physical path deviations. Second, it enables the developers to specialize and focus the feedback control on the few path deviations remaining. Specialized feedback control can tackle a few path errors much better than a general-purpose feedback controller.

Several model choices for the feed-forward control are possible. This work develops not only different model types but also applies different development methodologies. The physical model is developed in an iterative methodology, whereas the data-based model utilized a theoretical, general-purpose approach.

On the one hand, the physical model is designed with an iterative approach. A nonlinear model is developed based on physical knowledge and evaluated in simulation. Conclusions from simulations refine the model. The next iteration is tested on the real robot. Again, conclusions are made, resulting in another model improvement. This iterative approach uses the full modularity and explainability of the model. It is an analytical approach, bringing the simulated model and the real world closer together.

On the other hand, the data-based model starts with theory and a general understanding of the model class; only in the last stage is it applied to the robot. For example, the robot benchmark is utilized to demonstrate memory efficiency. Nevertheless, this could have been almost any continuous-time benchmark. Two main advances, state derivative normalization, and Input to State Stability stability are evaluated on the Cascaded Tank

System and Electro Mechanical Positioning System data, not on the robot. This methodology highlights one of the main benefits of data-based models: They are highly flexible, enabling fast adoption across different physical domains. This methodology is reflected in the development of the continuous-time NN model, as a general model, based on a literature review and applied to different domains. General recommendations for system identification tasks with continuous-time Neural Network are given in section 3.2 to section 3.5. This work might facilitate the knowledge of engineers and researchers new to the field of system identification with Neural Network. However, this methodology is quite in contrast to the iterative method applied for the physical robot model.

A model for flexible joints in a novel, continuously differentiable formulation has been derived. The continuous differentiability enables the model to integrate into downstream applications: Fast model execution using Automatic Differentiation tools by generating symbolic and optimized code representations E.3. Automatic Differentiation tools are at the core of many gradient-based algorithms; it facilitates global optimization for parameter identification. Furthermore, the flexible joint model enables advanced simulation, see section C.1, which can analyze high-order oscillations of the gearboxes.

Hydraulic Weight Counterbalance modeling is rare in the robotic's community. However, experiments in the second iteration of the robot presented severe deviations on the predicted and measured torques for joint 2. With the Hydraulic Weight Counterbalance model, the robot model accuracy could be significantly improved.

Experiments of the second iteration of the robot indicated friction parameter errors. Therefore, the nonlinear friction model is extended to increase its flexibility. Global optimization has been applied to design friction parameter experiments in the first step. In the second step, the friction has been optimized parameters using these measurements.

Finally, developing a flatness-based feed-forward control enables the advanced model to improve the robot path accuracy. This incorporates the complete, nonlinear model as presented above. Symbolic code representation is enabled by a continuously differentiable formulation of all terms and Automatic Differentiation tools.

The properties of continuous-time Neural Network for system identification have been analyzed and discussed. This discussion includes the Ordinary Differential Equations configuration required for continuous-time models, the Neural Network configuration, and the training pipeline configuration. Recommendations and design choices are given for engineers and researchers new to the field of system identification with Neural Network. The final performance of the Neural Network model applied to the robot control task is based primarily on these recommendations; the continuous-time model has not been designed in several iterations.

Data normalization is a key ingredient for machine learning algorithms. However, normal-

ization is applied exclusively to the discrete-time Neural Network properties in literature. It was shown that for continuous-time models, a normalization of the time dimension, state, and state derivative is possible in general. Concrete methods are presented to normalize a given application in the time, state, and state derivative domain. The method has been compared to system identification methods beyond Neural Network and achieved some of the best results reported yet.

Using constraint optimization, and based on [De11b; WDR21], a method to guarantee Input to State Stability stability has been developed. Especially for industrial applications, it is important to prove that all predicted states remain bounded in every potential use case and under all possible circumstances. The main improvements compared to [De11b] are threefold: The proof of the method has been corrected. A weak stability method has been developed, capturing a much broader model class. It applies to a wide range of continuous-time Neural Network with linear and *any* nonlinear terms. Furthermore, to integrate into modern deep learning frameworks, such as *TensorFlow* or *PyTorch*, both the Input to State Stability and the weak stability method have been enhanced with penalty and barrier functions. This way, constraint optimization problems can be cast into unconstrained optimization frameworks like *TensorFlow* or *PyTorch*. The performance results have been shown on the Electro Mechanical Positioning System benchmark.

The memory efficiency of continuous-time Neural Network has been demonstrated. Many sources already recognized this property in the system identification community. In the machine learning community, recognition improved over the last years, for example, in [Ch18; He16; Ki22]. Memory efficiency is a core advantage of continuous-time Neural Network compared to discrete-time Neural Network. This property was demonstrated in real experiments on the robot, directly comparing discrete-time and continuous-time models. It was further analyzed on the model parameter level.

Hybrid models, combining data-based and expert models, have been discussed. Different structures have been analyzed with their respective advantages and disadvantages. A particular focus has been given to model safety. Hybrid models can explicitly trade off exceptional data-based prediction accuracy for model safety.

This work addressed two communities, robotics and machine learning. Two possible future research directions are discussed.

The continuously differentiable flexible joint model enables Automatic Differentiation tools, which are prerequisites for many gradient-based algorithms. Automatic Differentiation tools have been used to derive high-performance code, improve the feed-forward control's real-time capabilities, and facilitate global parameter identification. However, the differentiability of the model combined with Automatic Differentiation tools supports integration into Model Predictive Control using flexible joint models. Model Predictive

Control defines a control task as an optimization problem, explicitly including a model, constraints on actuation, state, and other (nonlinear) considerations. It optimizes the actuation to a reference trajectory, incorporating future events in advance. Future research might address integrating a continuously-differentiable flexible joint model within a Model Predictive Control.

The robot software is deployed on the *B&R Automation* real-time system. High-level tasks, such as parameter identification, are implemented via an Open Platform Communications Unified Architecture server connection to external, non-real-time clients. However, to better integrate the research results into the robot community, it is worth considering using Robot Operating System as middleware. At the time of publication, no robot control hardware in this electrical power class with native Robot Operating System support was available. However, those components could be integrated in the future, making the results comparable and openly transferable to the robotics community. Even without Robot Operating System native hardware, a server-client connection via Open Platform Communications Unified Architecture or *B&R Powerlink* is possible. In this case, the high-level software components would be implemented in Robot Operating System and sent to the real-time system. Especially regarding the flexible parameter identification procedure, this is worth considering.

Regarding machine learning, not all applications can be solved by prediction only. Some require control. In this work, a model was trained as an inverse controller, achieving exceptional path accuracy results. However, not all applications can be solved by inverting the dynamic system. For advanced optimization applications, approaches such as Model Predictive Control and Reinforcement Learning already show promising results. It would be interesting to follow the path of [Uç19; Uç20; WVR20a] and integrate continuous-time Neural Network models with control applications. In [Uç19; Uç20], the combination of Model Predictive Control and a continuous-time Neural Network is investigated. In [WVR20a], a Runge-Kutta Neural Networks model is applied in a model-based Reinforcement Learning setting training a second network intended for robot control.

Inspired by [Ch18; He16; Ki22; Sm20], an application of continuous-time Neural Network to other, non-technical domains is very promising. [He16], the ResNet architecture, is applied to image classification and does not mention dynamical systems or even Ordinary Differential Equations configurations once. The mechanisms applied are called skip connections. Nevertheless, skip connections are mathematical equivalent to an Euler discretization method [HR18]. [Ch18], Neural Ordinary Differential Equations, gained much attention in the machine learning community. Nevertheless, it does not mention dynamical systems at all. [Sm20], the winning concept of the M4 competition, achieves outstanding results on various time series forecasting applications. Dynamic systems are not mentioned. It does not directly apply an explicit Ordinary Differential Equations scheme. It

refers to a hybrid model consisting of Long Short Term Memory network and an exponential smoothing method. Exponential smoothing methods are mathematically equivalent to continuous-time low-pass filters of first-order. Therefore, the approach in [Sm20] is similar to a continuous-time state-space Neural Network. These examples illustrate the effectiveness of the continuous-time Neural Network, even in other domains. It will be interesting what other exciting applications will emerge in the future with continuous-time Neural Network.

Appendix

A Proof of Stability Theorem

This proof is presented in [WDR21]. It is similar to the proof presented in [De11b], with the difference that an augmented model input is defined. This eliminates the requirement of a coordinate transformation in [De11b]. For details of the contributions, see appendix 5.4.

The idea of the proof is to find an estimation for (5.44), such that Linear Matrix Inequalities (LMI) only depend on the network weights and fixed parameters. First, the terms X and U are estimated. Then, L is eliminated. For the sake of simplicity, $X = W_x^l + W_a^o L W_x^h$ and $U = W_u^l + W_a^o L W_u^h$ is set to define $f_{\text{NN}}(\hat{x}, \tilde{u}) = Xx + U\tilde{u}$. The left-hand side of (5.44) can be estimated by

$$\langle x, f_{\text{NN}}(\hat{x}, \tilde{u}) \rangle_{P_1} + hm \langle f_{\text{NN}}(\hat{x}, \tilde{u}), f_{\text{NN}}(\hat{x}, \tilde{u}) \rangle_{P_1} \quad (1)$$

$$= x^T P_1 (Xx + U\tilde{u}) + hm \left((Xx + U\tilde{u})^T P_1 (Xx + U\tilde{u}) \right) \quad (2)$$

$$= x^T (P_1 X + hm X^T P_1 X) x + x^T (P_1 U + 2hm X^T P_1 U) \tilde{u} + \tilde{u}^T (hm U^T P_1 U) \tilde{u} \quad (3)$$

with a positive definite matrix $P_1 \in \mathbb{R}^{N_x \times N_x}$. To estimate the term $x^T (P_1 U + 2hm X^T P_1 U) \tilde{u}$ general relation is applied

$$2z_1^T z_2 \leq z_1^T P_2 z_1 + z_2^T P_2^{-1} z_2, \quad (4)$$

which is valid for any vectors $z_1, z_2 \in \mathbb{R}^{N_x}$ and for any positive definite matrix $P_2 > 0$, $P_2 \in \mathbb{R}^{N_x \times N_x}$ [HJ12].

$$x^T(P_1U + 2hmX^TP_1U)\tilde{u} \quad (5)$$

$$= 2\left(\frac{1}{2}x\right)^T(P_1U\tilde{u}) + 2(x)^T(hmX^TP_1U\tilde{u}) \quad (6)$$

$$\leq \left(\frac{1}{2}x\right)^TP_2\left(\frac{1}{2}x\right) + (P_1U\tilde{u})^TP_2^{-1}(P_1U\tilde{u}) \quad (7)$$

$$+ x^TP_2x + (hmX^TP_1U\tilde{u})^TP_2^{-1}(hmX^TP_1U\tilde{u}) \quad (8)$$

$$= \frac{5}{4}x^TP_2x$$

$$+ \tilde{u}^T(U^TP_1^T \cdot (P_2^{-1} + h^2m^2XP_2^{-1}X^T)P_1U)\tilde{u}.$$

Substituting (9) in (4) leads to

$$\langle x, f_{\text{NN}}(\hat{x}, \tilde{u}) \rangle_{P_1} + hm \langle f_{\text{NN}}(\hat{x}, \tilde{u}), f_{\text{NN}}(\hat{x}, \tilde{u}) \rangle_{P_1} \quad (9)$$

$$\leq x^T \left(\frac{5}{4}P_2 + P_1X + hmX^TP_1X \right) x \quad (10)$$

$$+ \tilde{u}^T(U^T(hmP_1 + P_1^TP_2^{-1}P_1$$

$$+ h^2m^2P_1^TXP_2^{-1}X^TP_1)U)\tilde{u}.$$

$$< \gamma \langle \tilde{u}, \tilde{u} \rangle. \quad (11)$$

Equation (11) is fulfilled and thereby Theorem 5.2.2 (proof see [De11b; DR14]) in two steps. First, it is shown that all terms in (11) associated with \tilde{u} are lower bounded by a function $\gamma \langle \tilde{u}, \tilde{u} \rangle \geq 0$. Second, it must be ensured that the terms associated with x are strictly negative. The positive function for any input \tilde{u} and any state x is given by

$$\gamma \langle \tilde{u}, \tilde{u} \rangle = \min_{x, \tilde{u}} \|(U(x, \tilde{u}))^T(hmP_1 + P_1^TP_2^{-1}P_1$$

$$+ h^2m^2P_1^TX(x, \tilde{u})P_2^{-1}(X(x, \tilde{u}))^TP_1)U(x, \tilde{u})\| \geq 0. \quad (12)$$

Equation (12) always holds as the norm is by definition greater than or equal to zero. To ensure that the terms in (11) associated with the state x are strictly less than zero,

$$Z_1 = \frac{5}{4}P_2 + P_1X + hmX^TP_1X. \quad (13)$$

Re-substitution of X and calculation leads to

$$X^T P_1 X = (W_x^l + W_a^o L W_x^h)^T P_1 (W_x^l + W_a^o L W_x^h) \quad (14)$$

$$\begin{aligned} &= W_x^l{}^T P_1 W_x^l + W_x^l{}^T P_1 W_a^o L W_x^h \\ &\quad + (W_a^o L W_x^h)^T P_1 W_x^l \\ &\quad + (W_a^o L W_x^h)^T P_1 W_a^o L W_x^h. \end{aligned} \quad (15)$$

Next, L is eliminated in two steps. It is ensured that $\|L\| \leq 1$ holds and the positive definite term

$$x^T (W_a^o L W_x^h)^T P_1 W_a^o L W_x^h x \leq x^T Q x, \quad (16)$$

$$Q = \|W_a^o{}^T P_1 W_a^o\| \|W_x^h\|^2 I. \quad (17)$$

can be estimated. Using (17) in (16) leads to

$$\begin{aligned} X^T P_1 X &\leq W_x^l{}^T P_1 W_x^l + W_x^l{}^T P_1 W_a^o L W_x^h \\ &\quad + (W_a^o L W_x^h)^T P_1 W_x^l + Q. \end{aligned} \quad (18)$$

Substitution of (19) in (13) results in

$$x^T Z_1 x \quad (19)$$

$$\leq x^T (P_3 + P_1 W_x^l + P_1 W_a^o L W_x^h) \quad (20)$$

$$\begin{aligned} &\quad + hm \left(W_x^l{}^T P_1 W_x^l + W_x^l{}^T P_1 W_a^o L W_x^h \right. \\ &\quad \left. + (W_a^o L W_x^h)^T P_1 W_x^l + Q \right) x \\ &= x^T \left(T_1 + (hm W_x^l{}^T + I) P_1 W_a^o L W_x^h \right. \end{aligned} \quad (21)$$

$$\begin{aligned} &\quad \left. + hm (W_a^o L W_x^h)^T P_1 W_x^l \right) x \\ &= x^T Z_2 x \end{aligned} \quad (22)$$

with

$$T_1 = P_3 + P_1 W_x^l + hm \left(W_x^l{}^T P_1 W_x^l + Q \right). \quad (23)$$

The approach in [HW02] is utilized to estimate the other terms independently of L . The i^{th} row vector of matrix W_x^h is defined as R_i^{Wh} and the i^{th} column vector of matrix W_a^o is defined as C_i^{Wo} . This results in

$$W_a^o L W_x^h = \sum_{i=1}^{N_N} l_i C_i^{W^o} R_i^{W^h}. \quad (24)$$

Consequently, this leads to

$$x^T Z_2 x \quad (25)$$

$$= x^T \left(T_1 + (hm W_x^l{}^T + I) P_1 \sum_{i=1}^{N_N} l_i C_i^{W^o} R_i^{W^h} \right. \\ \left. + hm \left(\sum_{i=1}^{N_N} l_i C_i^{W^o} R_i^{W^h} \right)^T P_1 W_x^l \right) x \quad (26)$$

$$= x^T \left(T_1 + \sum_{i=1}^{N_N} l_i \left((hm W_x^l{}^T + I) P_1 C_i^{W^o} R_i^{W^h} \right. \right. \\ \left. \left. + hm (C_i^{W^o} R_i^{W^h})^T P_1 W_x^l \right) \right) x \quad (27)$$

$$= x^T Z_3 x. \quad (28)$$

Introducing the weighting variables λ , with $\sum_{i=1}^{N_N} \lambda_i = 1$, $0 \leq \lambda_i \leq 1$, $\forall i \in \{1, \dots, N_N\}$, reformulation of T_1 results in

$$T_1 = \left(1 - \sum_{i=1}^{N_N} l_i \lambda_i \right) T_1 + \left(\sum_{i=1}^{N_N} l_i \lambda_i \right) T_1. \quad (29)$$

Incorporating (29) into the sum leads to

$$x^T Z_3 x \quad (30)$$

$$= x^T \left(1 - \sum_{i=1}^{N_N} l_i \lambda_i \right) T_1 x \quad (31)$$

$$+ \sum_{i=1}^{N_N} l_i x^T \left((hm W_x^l{}^T + I) P_1 C_i^{W^o} R_i^{W^h} \right. \\ \left. + hm (C_i^{W^o} R_i^{W^h})^T P_1 W_x^l + T_1 \lambda_i \right) x \\ = x^T \left(1 - \sum_{i=1}^{N_N} l_i \lambda_i \right) T_1 x + \sum_{i=1}^{N_N} l_i x^T (T_{2,i} + T_1 \lambda_i) x \quad (32)$$

with

$$\begin{aligned}
T_{2,i} = & \left(hmW_x^l{}^T + I \right) P_1 C_i^{Wo} R_i^{Wh} \\
& + hm \left(C_i^{Wo} R_i^{Wh} \right)^T P_1 W_x^l.
\end{aligned} \tag{33}$$

By the definition of the weight variables λ_i and the fact that $0 \leq l_i \leq 1$ holds, it is ensured that

$$\left(1 - \sum_{i=1}^{N_N} l_i \lambda_i \right) \geq 0 \tag{34}$$

holds, and the proof is completed.

B Technical Robot Details

B.1 Dimensions and Limits

The robot workspace is presented in Fig. 1 and Fig. 2. All lengths are given in mm and angles are given in deg. Please note that Fig. 1 and Fig. 2 applies a different zero-pose notation than the measurements. The angular difference is given by

$$\mathbf{q} = [0, -90, 90, 0, 0, 0] \text{ deg.} \tag{35}$$

The position and velocity limits are presented in Tab. 1, using the measurement position convention. As the velocity limits are all symmetrical, only the upper limits are presented, as $\dot{q}_{lb,n} = -\dot{q}_{ub,n}$ holds.

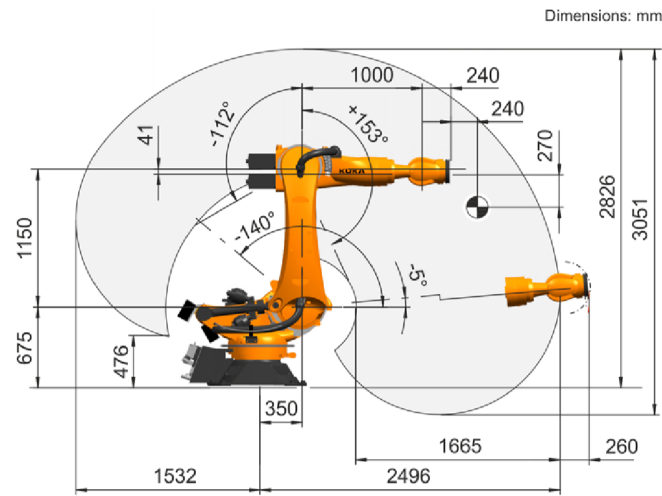


Figure 1: KUKA KR300 R2500 ultra SE robot dimensions (top view) in mm and absolute angular limits in deg. Different zero angle definitions. Copyright by [KU21].

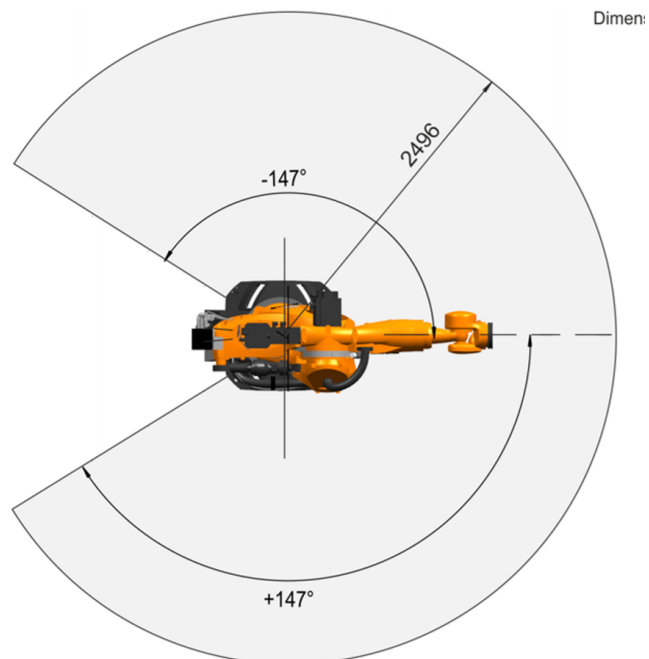


Figure 2: KUKA KR300 R2500 ultra SE robot dimensions (side view) in mm and absolute angular limits in deg. Different zero angle definitions. Copyright by [KU21].

Table 1: Joint position and velocity limits. Velocity limits are symmetrical. Absolute limits are caused by electrical and mechanical constraints [KU21]. In the experiments, a reduced subset is applied.

	Absolute Limits			Applied Limits		
	q_{lb}	q_{ub}	\dot{q}_{ub}	q_{lb}	q_{ub}	\dot{q}_{ub}
	deg	deg	deg/s	deg	deg	deg/s
Joint 1	-147	147	84.6	-90	90	63.4
Joint 2	-50	85	82.3	-30	40	61.7
Joint 3	-202	63	79.3	-110	40	59.5
Joint 4	-350	350	122	-180	180	91.5
Joint 5	-122.5	122.5	113	-90	90	84.8
Joint 6	-350	350	175	-180	180	131.3

B.2 Robot Controller Layout

Figure 3 presents the basic robot controller layout. Starting at the top right, an industrial real-time computer performs the main robot tasks: the process state machine, the measurement handling, the diagnose system, the web-based human-machine-interface, the hand-held operating device, parts of the control loop, as well as error-detection and machine supervision. The real-time computer hosts an Open Platform Communications Unified Architecture (OPC-UA) server, which communicates bidirectionally with a *Matlab* OPC-UA client. The *Matlab* code performs non-real-time tasks such as trajectory generation, robot model identification, or machine learning applications. *Matlab* is also the interface for measurement post-processing and file management. The industrial computer also hosts a second non-real-time operating system, Windows 10, on which the programming and diagnosis can be done. Other than for diagnosis, the Windows-based operating system does not interfere with the run-time tasks of the robot. Cycle time of main tasks is given in Tab. 2. The main components presented in Fig. 3 are listed in Tab. 3.

The industrial controller features a bus protocol, called *POWERLINK* by *B&R Automation*, over which all run-time information is communicated. The green lines on the bottom represent the motor signals, such as the motor temperature sensor, motor brake control, and motor position and velocity signal.

Following the industrial bus from the real-time computer, a safety PLC is connected. This PLC is safety-certified, independent of all other hard- and software, and can shut down the main power supply in all cases. It explicitly handles hardware emergency stops and supervises the robot cell. It is the only component that cannot be reconfigured without advanced authorization.

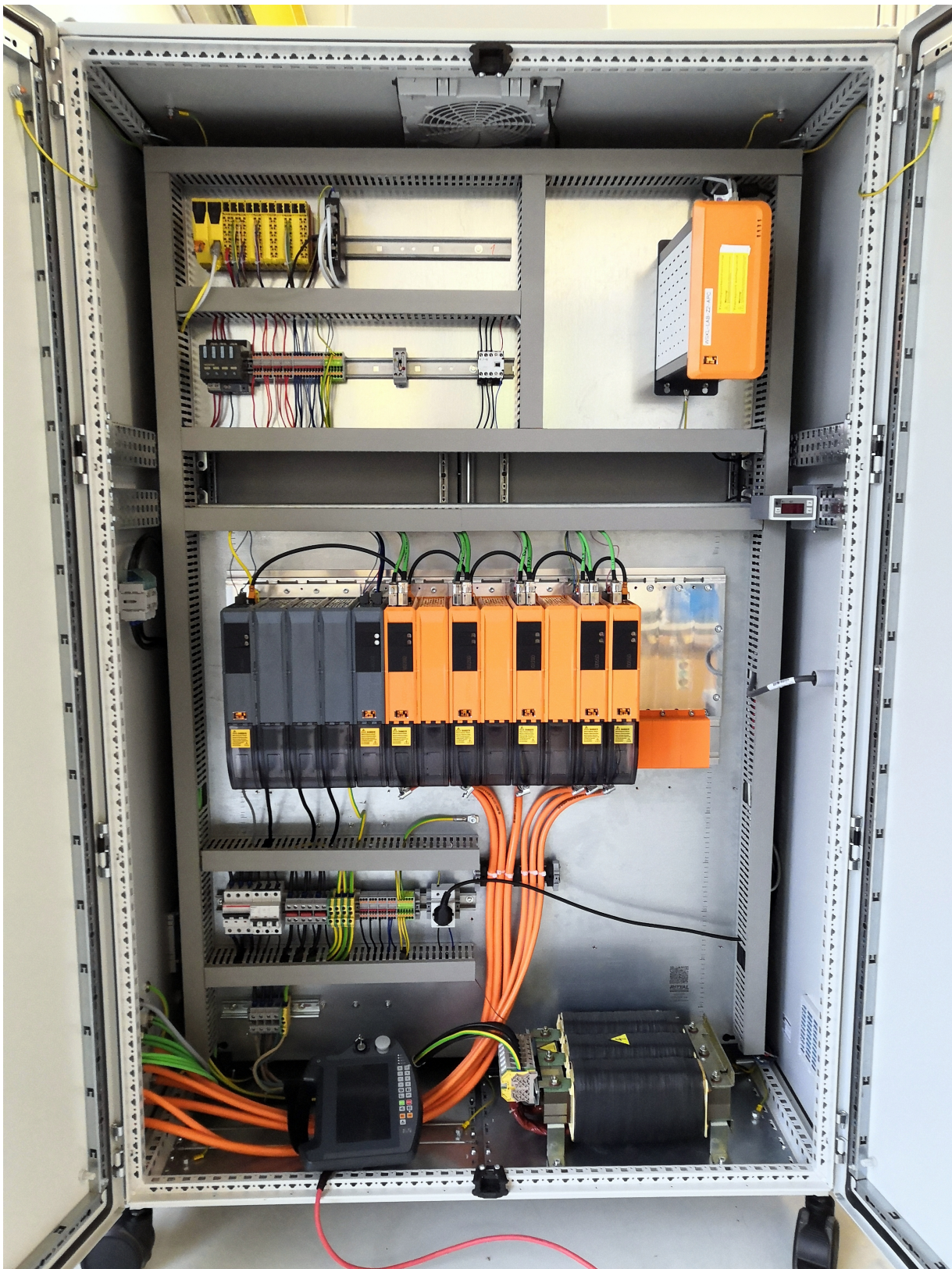


Figure 3: Picture of the Custom-made controller of the Industrial robot KUKA KR300 R2500 ultra SE manipulator at the Chair of Machine Tools and Control Systems, RPTU Kaiserslautern-Landau.

Table 2: (Deterministic) computation time of main software components [BR21b].

Task	Deterministic	Time
OPC-UA server	no	between 4 ms and 10 s, depending on task requirements and data volume. Bidirectional.
State machine	yes	4 ms
Diagnose system	yes	4 ms
Measurement handling	yes	4 ms, can be set to 0.8 ms
Position feedback control	yes	0.8 ms
Velocity feedback control	yes	0.2 ms
Motor current feedback control	yes	0.05 ms

The bus signal is sent from the safety PLC to the main power supply, which transforms the laboratory power supply corresponding to the motor inverter requirements. Parallel to the main power, the industrial bus signal is sent to all motor inverters, which locally compute the position, velocity, and current feedback control, record measurements, and read the secondary encoders. The code of the motor inverters, as well as the information which is sent over the bus system, can be adopted.

All motor inverters are designed for a double-axis. For joints 1, 2, and 3, only one high-power output is used. The other side of the double-axis is required for additional sensor integration and applied for software computation directly on the inverter. Joints 4, 5, and 6 operate on a double-axis, with Joint 4 and 5 on a single device. The other side of the double-axis device for joint 6 is unused.

Table 3: Main components of the robot control [BR21a; BR21b].

Component	Description
1x Real-Time Computer	Automation PC 910 series with Intel Core i7 6820EQ, QM170 chipset, 128 GB CFast memory, 8 GB DDR4 RAM, POWERLINK managing node, 2 MB SRAM battery buffered
1x Power Supply	ACOPOSmulti series with 3x 400 VAC input voltage, 750 V_{DC} link voltage, 60 kW continuous power
3x Inverter Joint 1, 2, 3	ACOPOSmulti3 series dual-axis module with 750 V_{DC} bus voltage, 55 A peak current, 22 A continuous current, 16 kW continuous power, 24 V_{DC} and 2.1 A for holding brake
2x Inverter Joint 4, 5, 6	ACOPOSmulti3 series dual-axis module with 750 V_{DC} bus voltage, 18.9 A peak current, 7.6 A continuous current, 5.5 kW continuous power, 24 V_{DC} and 1.1 A for holding brake

B.3 Sensor Specification

Table 4 shows the sensor types and corresponding resolutions for all joints. This work refers to the software resolution as the smallest incremental change that can be detected in the software. The real sensor uncertainty is greater: First of all, the sensor head, the measurement ring, and the resolver depend on manufacturing tolerances. Considering the manufacturer’s specifications, however, these uncertainties are not significant. Second, for both sensors, this work defines the resolution on the link side. This implies for the motor-side mounted resolvers, that the effective resolution depends on the gearbox factors. So this estimated effective resolution is corrupted by gearbox deformation and backlash, which sums up to an approximate error range of 0.01 deg to 0.05 deg (depending on the trajectory and payload, see [We22; WGR20]). All sensors are low pass filtered in the first order, with a cutoff frequency of 1000 Hz for both the secondary encoders and the motor resolves.

Table 4: Secondary encoders (SE) [AM21] and motor resolvers (RE) [SI06]. All resolutions are given for the link side. Software resolution is the smallest incremental change that can be detected. Effective resolution accounts for additional uncertainties, see main text.

Sensor	Type	Software Resolution deg	Effective Resolution deg
SE Joint 1	WMR-301-0507-01-S03 (Ring) WMR-301.12-0507-0.20-9-S01 (Head)	$4.33 \cdot 10^{-6}$	$5 \cdot 10^{-6}$
SE Joint 2	WMR-301-0413-01-S03 (Ring) WMR-301.12-0413-0.10-9-S01 (Head)	$5.32 \cdot 10^{-6}$	$6 \cdot 10^{-6}$
SE Joint 3	WMR-301-0339-01-S03 (Ring) WMR-301.12-0339-0.125-9-S01 (Head)	$6.48 \cdot 10^{-6}$	$7 \cdot 10^{-6}$
RE Joint 1	1FK7-101-5AY71-1SY3-Z	$85.5 \cdot 10^{-6}$	$5 \cdot 10^{-2}$
RE Joint 2	1FK7-103-5AY71-1SY3-Z	$82.2 \cdot 10^{-6}$	$5 \cdot 10^{-2}$
RE Joint 3	1FK7-103-5AY71-1SY3-Z	$87.1 \cdot 10^{-6}$	$5 \cdot 10^{-2}$
RE Joint 4	1FK7-063-5AF71-1SY3-Z	$99.4 \cdot 10^{-6}$	$5 \cdot 10^{-2}$
RE Joint 5	1FK7-063-5AF71-1SY3-Z	$91.7 \cdot 10^{-6}$	$5 \cdot 10^{-2}$
RE Joint 6	1FK7-063-5AF71-1SY3-Z	$240 \cdot 10^{-6}$	$5 \cdot 10^{-2}$

B.4 Physical Robot Model Parameters

Tab. 5 shows the parameters utilized of the KUKA KR300 R2500 ultra SE robot.

Table 5: Model parameters of KUKA Quantec KR300 Ultra SE robot.

description	symbol	joint 1	joint 2	joint 3	unit
asymmetrical friction	f_{asym}	54.02	-255.55	-200.53	Nm
viscous friction	f_v	2050.35	2306.7	2024.4	Nms/rad
Coulomb friction	f_c	285.36	10	29.86	Nm
degressive friction A	f_a	70.14	232.33	264.63	Nm
degressive friction B	f_b	530	28994	31572	s/rad
friction smoothness factor	s_F	500	300	100	s/rad
proportional speed gain	K_V	0.015	0.015	0.015	Nms/rad
proportional position gain	K_P	20	20	20	1/s
backlash angle	ϕ_B	0.15	0.15	0.15	10^{-3} rad
lost motion angle	ϕ_{LM}	0.15	0.15	0.15	10^{-3} rad
torsional rigidity stiffness	c_{TR}	8.4225	8.9381	5.5691	10^6 Nm/rad
stiffness smoothness factor	s_{E2}	0.02	0.015	0.015	1/Nm
gearbox ratio	u_G	256.86	267.43	252.33	—
motor inertia	J	0.0138	0.0177	0.0177	kgm ²
		joint 4	joint 5	joint 6	
asymmetrical friction	f_{asym}	-2.12	0.49	0.37	Nm
viscous friction	f_v	453.66	1000	458.8	Nms/rad
Coulomb friction	f_c	34.34	10	15.45	Nm
degressive friction A	f_a	16.05	36.73	60.68	Nm
degressive friction B	f_b	837.45	4855.06	2465.3	s/rad
friction smoothness factor	s_F	200	200	200	s/rad
proportional speed gain	K_V	0.015	0.015	0.015	Nms/rad
proportional position gain	K_P	20	20	20	1/s
backlash angle	ϕ_B	0.15	0.15	0.15	10^{-3} rad
lost motion angle	ϕ_{LM}	0.15	0.15	0.15	10^{-3} rad
torsional rigidity stiffness	c_{TR}	1.6845	1.6845	1.0726	10^6 Nm/rad
stiffness smoothness factor	s_{E2}	0.015	0.015	0.015	1/Nm
gearbox ratio	u_G	221.00	239.62	154.32	—
motor inertia	J	0.0150	0.0150	0.0150	kgm ²

C Additional Experiments Physical Model

This section presents additional experiments for the intermediate versions of the physical robot model. First, additional insight regarding flexible joints is given using simulations. Then, experiments on the real robot are presented. These two discussions are similar to previously published contribution [WGR20]. Using these simulations and real experiments, the final subsection motivates the additional advancements presented in chapter 4.

C.1 Simulation Results

In the following, the feed-forward and feedback algorithms are analyzed in simulation. The flatness-based feed-forward controller is compared with a nonlinear rigid model feed-forward controller. Perfect model knowledge, neglecting of sensor noise, and neglecting of any disturbances are assumed in order to focus on pure modeling differences. Unlike on the real robot, the feedback controller can be disabled if needed in simulation. The presented algorithm is compared with a nonlinear feed-forward control law without elastic joints, e.g., $\theta = Uq$. As a result, the model in (4.1b) and 4.1a reduces to the rigid joint model (2.1). Consequently, the model-based feed-forward control reduces to (2.2).

For compact notation, abbreviations of control structures are defined in the following.

- **Conventional Feedback Controller (C-FB)**. Feedback controller neglecting joint elasticity. Motor reference velocity is $\dot{\theta} = U\dot{q}$.
- **Model-Based Feedback Controller (MB-FB)**. Model-based feedback controller accounting for joint elasticity. Motor reference velocity is estimated with (4.39).
- **Rigid-Model Feed-Forward Controller (R-FF)**. Nonlinear rigid joint feed-forward control. See (2.2).
- **Flatness-Based Feed-Forward Controller (FB-FF)**. Nonlinear elastic joint feed-forward control. See (4.36).

Both feedback controllers, C-FB and MB-FB apply the structure using SE as explained in 4.3.1. A comparison without SE is not comprehensive. Furthermore, the feed-forward control, in both cases, applies the advanced nonlinear robot model - except that one is neglecting joint elasticity. Advanced nonlinear friction, HWC, and all other model parameters are identical. There is no comparison with the robot control toolbox supplied by the industrial partner *B&R Automation*. So the experiments focus on the effect of the elastic joint model and focus on the model-based feedback control.

For an unbiased comparison, the same nonlinear friction (2.3), Coriolis, centripetal, and gravity torque are applied as in the flatness-based controller. All model parameters are

identical, including the pose-dependent inertia matrix. Consequently, the only difference between the flatness-based and nonlinear rigid model feed-forward control law is neglecting the joint elasticity. Both feed-forward controllers are simulated independently on the same model, (4.1a) and (4.1b). For a compact notation, Flatness-Based Feed-Forward Controller (FB-FF) and Rigid-Model Feed-Forward Controller (R-FF) are defined and presented in (4.36) and (2.2) respectively.

Fig. 4 presents the simulation's reference trajectory, velocity, and acceleration. For a clear overview, the jerk and the jerk derivative are not illustrated, although both are continuously differentiable. Furthermore, for a comprehensive simulation, the acceleration profile is shown in Fig. 4 rather than a polynomial-based acceleration and deceleration phase.

Theoretically, since the simulation does not contain noise, model errors, or external disturbances, the flatness-based feed-forward controller should follow an arbitrary trajectory perfectly. As presented in Fig. 5, this can be achieved in simulation. Fig. 5 shows the angular error, e.g., $q_{\Delta,i} = q_{R,i} - q_i$, for the FB-FF and R-FF in simulation. For comparison, the range of the backlash angle ϕ_B is displayed as a gray area in Fig. 5. As expected, the R-FF is slightly worse than the FB-FF. Fig. 5 validates the FB-FF control, as it shows in the simulation that the inverse model almost perfectly matches the forward model. It also validates the assumptions in section 4.3.3, as the forward model does not neglect parameter changes during FB-FF computation time. However, a core insight of Fig. 5 is the performance of the R-FF controller on the elastic joint model.

1. Considering the dynamic trajectory, angular errors can be expected significantly greater than 0.06 deg. Fig. 5 is representative of many simulations confirming this result. If a model error is simulated, e.g., the feed-forward model parameters do not match the simulation model parameters, the angular error easily exceeds > 1 deg. In conclusion, the model accuracy matters regarding the model structure and parameter identification.
2. Note that the simulation presented in Fig. 5 does not apply any feedback control. If an additional feedback controller is applied, which does not account for elastic joints, the angular error increases by an order of magnitude. Therefore, it is very important to implement MB-FB instead of C-FB as explained in section 4.3.2.

Fig. 6 compares the feed-forward torque of R-FF and FB-FF. Due to the identical friction, inertia, gravity, Coriolis, and centripetal terms of FB-FF and R-FF, the computed motor torques in Fig. 6 are broadly similar. Differences only occur in the sections where the model traverses backlash or Coulomb friction.

3. Besides compensating backlash, see Fig. 5, these minor changes in motor torque have huge effects on the elastic torque of the model. As presented in Fig. 7, oscillations

induced by backlash within the joint are compensated. Note that a SE feedback controller is not employed in the simulations, which would be able to dampen these oscillations. However, it is beneficial if these oscillations are not induced in the first place by proper flatness-based control.

Nevertheless, some detrimental aspects during the analysis in simulation can be observed.

4. Modeling backlash as a flat function yields increased motor torque change rates. Generally, the less flat a system is, the more dynamic the input variable should be. On a real robot, a dynamic input is not desirable. As presented in Fig. 6, the torque change rates can be reduced to a reasonable level for the robot. However, the demanded torque change rate is the algorithm's bottleneck. This affects the change rate only, and the absolute limit of the motor torque was not problematic in any analysis.
5. The backlash is only one source of positional errors. Model errors, sensor noise, external disturbances, and conventional feedback controllers significantly impact positioning accuracy. Neither the nonlinear nor the flatness-based feed-forward torque can account for these effects. To achieve a high positioning accuracy, a model-based feedback controller with SE, is necessary.

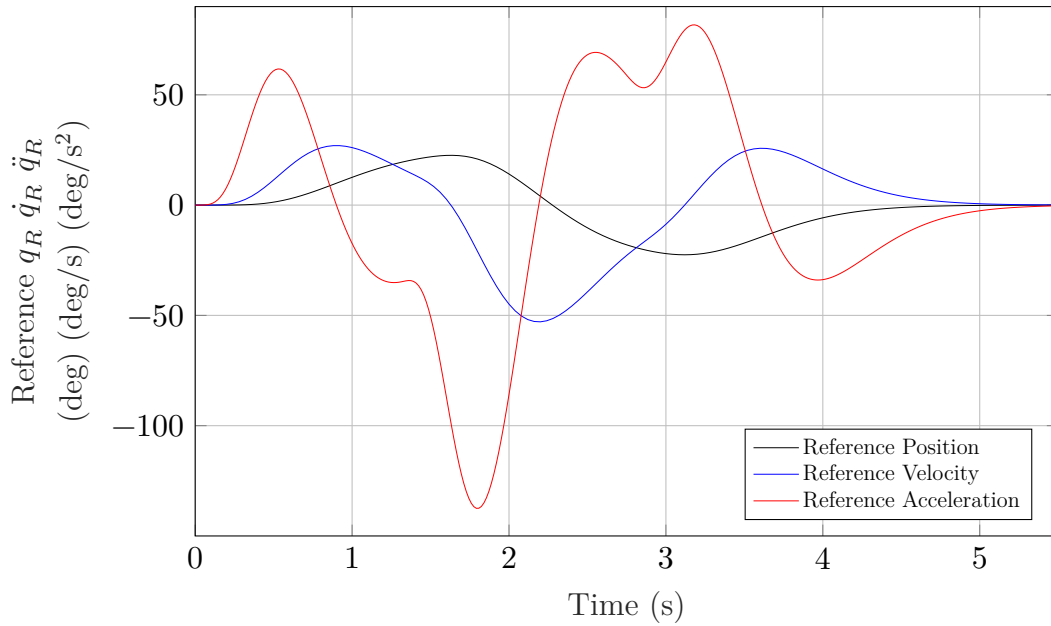


Figure 4: Reference trajectory with angular position, velocity, and acceleration used for simulation.

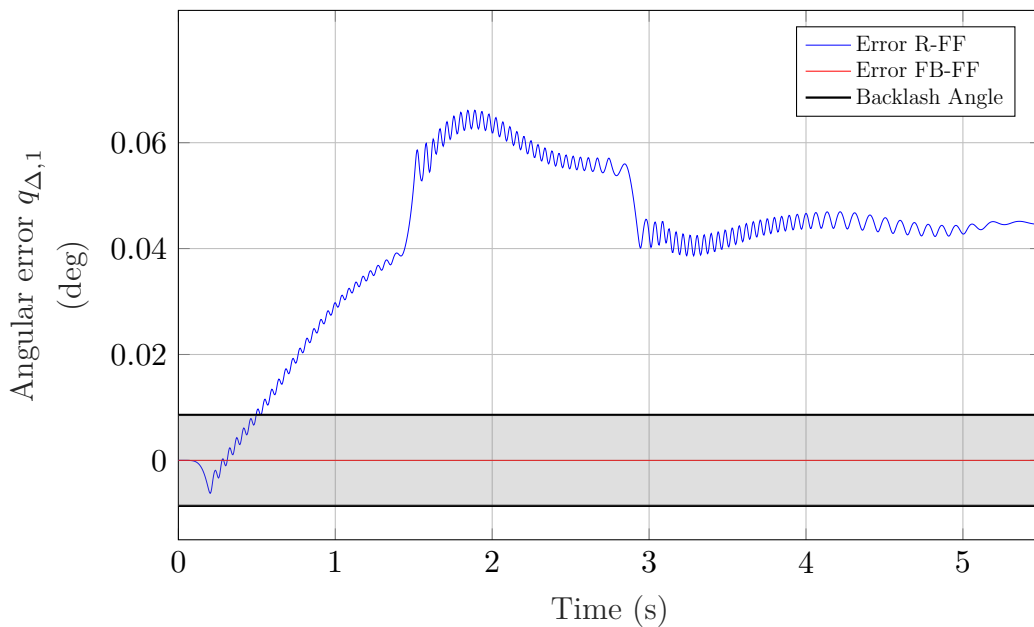


Figure 5: Angular error in simulation with perfect model knowledge, without noise, feedback controller, or any disturbances. The Gray area represents the backlash angle for comparison. FB-FF refers to the flatness-based controller and R-FF to the nonlinear rigid model controller. In the case of FB-FF, the forward and inverse models are identical.

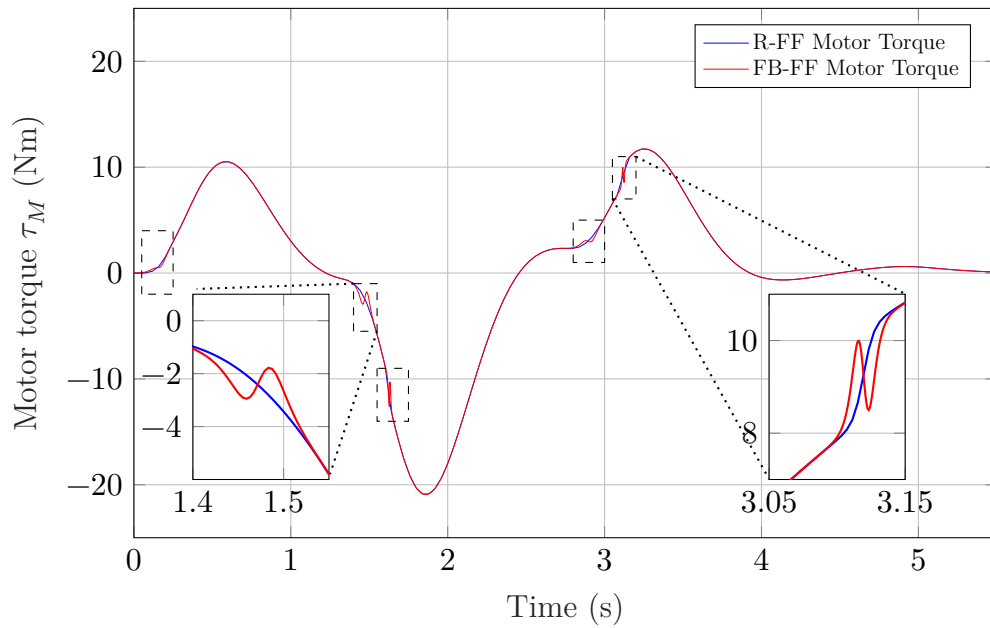


Figure 6: Feed-forward motor torque in simulation. FB-FF refers to the flatness-based controller and R-FF to the nonlinear rigid model controller.

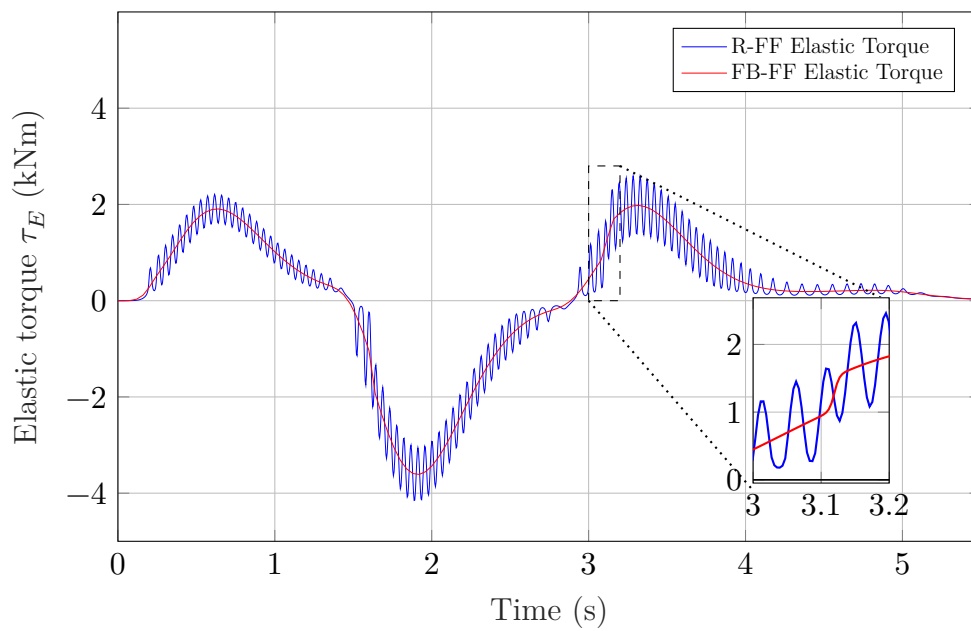


Figure 7: Elastic joint torques in simulation. FB-FF refers to the flatness-based controller and R-FF to the nonlinear rigid model controller.

C.2 Experimental Results

The presented feed-forward and feedback control laws are validated on an industrial robot, pictured in Fig. 6.1. The hardware and robot operating system is based on components from *KUKA* and industrial supplier *B&R Automation*. It is depicted in Fig. 3. The *B&R Automation* robot operating system allows implementation of the control algorithms as described in section 4.3.

A full robot movement in Cartesian space was chosen as an experimental scenario. The movement of joints 1, 2, and 3 can be measured using the same SE applied for position control. Technical details of the sensors are given in appendix B.3. All experiments are carried out on the cold robot, where both gearboxes and motors have approximately a temperature of 24.7°C, measured before and after the experiments. This leads to a significant effect of nonlinear friction in the base, shoulder, and elbow joints. The experiment has been carried out 10 times in a row, and the measurements are representative and reproducible. In order to account for a milling spindle in a robot machining application and to increase the dynamical loads on each joint, a payload of 150 kg on the robot's TCP is applied as presented in Fig. 6.1.

The trajectory is based on the Lemniscate of Geronon, which can be parametrized as

$$\begin{aligned} x(t) &= a \cos(\varphi(t)) \\ y(t) &= b \sin(\varphi(t)) \cos(\varphi(t)) \end{aligned} \quad (36)$$

with the horizontal and vertical length parameters a , b . The Cartesian angle $\varphi(t)$ performs a 7th-order continuously differentiable acceleration and deceleration trajectory. An additional joint space filter is applied after estimating the inverse kinematics. In order to explicitly test the effects of backlash, lost motion, and Coulomb friction, a Cartesian movement is chosen, with several changes in direction and parts with link velocities close to zero. The joint space reference trajectory is shown in Fig. 8. For a detailed discussion, the following argumentation utilizes the measured joint angles and measured motor torques instead of the estimated Cartesian coordinates with potential errors in the kinematic model.

The angular position improvements are shown in Fig. 9, which present the angular displacement $q_{\Delta,i} = q_{R,i} - q_i$ of joint 1, 2 and 3 for three cases. The same control parameters, filter constants, and dynamic model parameters are applied in all cases. A movement with a C-FB law is analyzed as a baseline. The subsequent measurement shows that a Model-Based Feed-Forward Controller (MB-FF) leads to an improvement for all joints. A further improvement can be achieved by applying the FB-FF and MB-FF. For joints 1, 2 and 3,

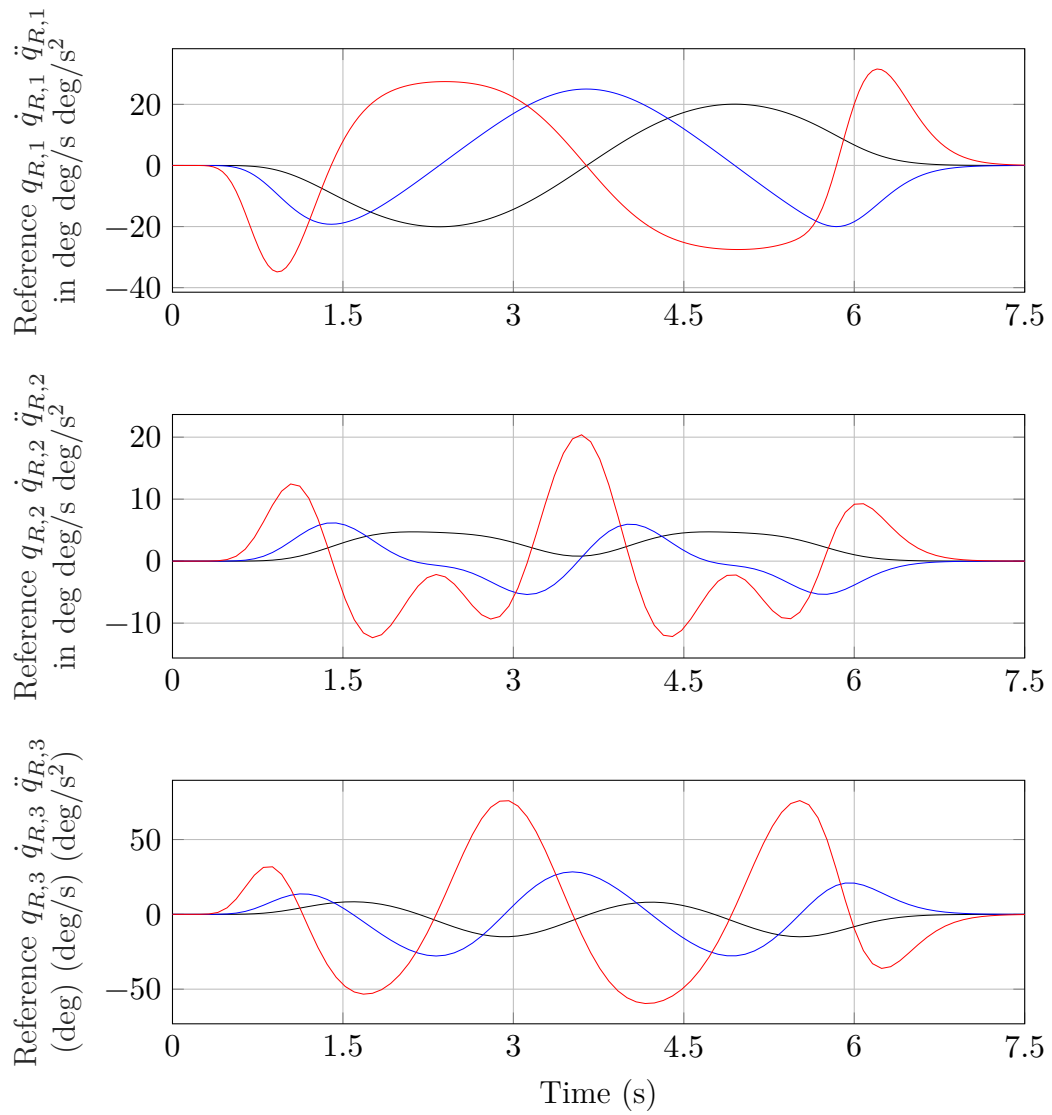


Figure 8: Joint space reference trajectories for each joint. Angular position in black, velocity in blue, and acceleration in red.

Table 6: Measured mean angular errors for all presented methods.

Method	Joint 1	Joint 2	Joint 3	Unit
C-FB	0.087	0.020	0.106	deg
MB-FB	0.081	0.019	0.101	deg
Improvement	-6.9	-5	-4.7	%
R-FF	0.052	0.018	0.034	deg
FB-FF	0.044	0.023	0.024	deg
Improvement	-15.4	27.8	-29.4	%

the maximum path angular error does not exceed ± 0.08 deg. Note that the angular resolution of the SE is only ± 0.017 deg. Therefore, the achieved performance of the feedback control law is based only on 5 measurement increments. All in all, the FB-FF and MB-FB improve the mean error of joint 1 by 49% and of joint 3 by 77% compared to C-FB. A detailed analysis of this improvement is presented in Tab. 6. The mean angular error is $\text{mean}(\text{abs}(q_{\Delta,i}))$ and the maximum angular error is $\text{max}(\text{abs}(q_{\Delta,1}))$ in Tab. 6.

As a comprehensive validation, a nonlinear R-FF with C-FB is compared to FB-FF with MB-FB on the same trajectory. Model-based feedback control is not applicable if the rigid joint model neglects elasticity. The experimental result is shown in Fig. 10. The flatness-based controller leads to better performance for joints 1 and 3 than the nonlinear rigid model feed-forward control law. The mean error for joint 1 is improved by 15% and for joint 3 by 29%. The detailed analysis is given in Tab. 6. Regarding joint 2, both controllers achieve a similar performance of less than 0.073 deg angular error.

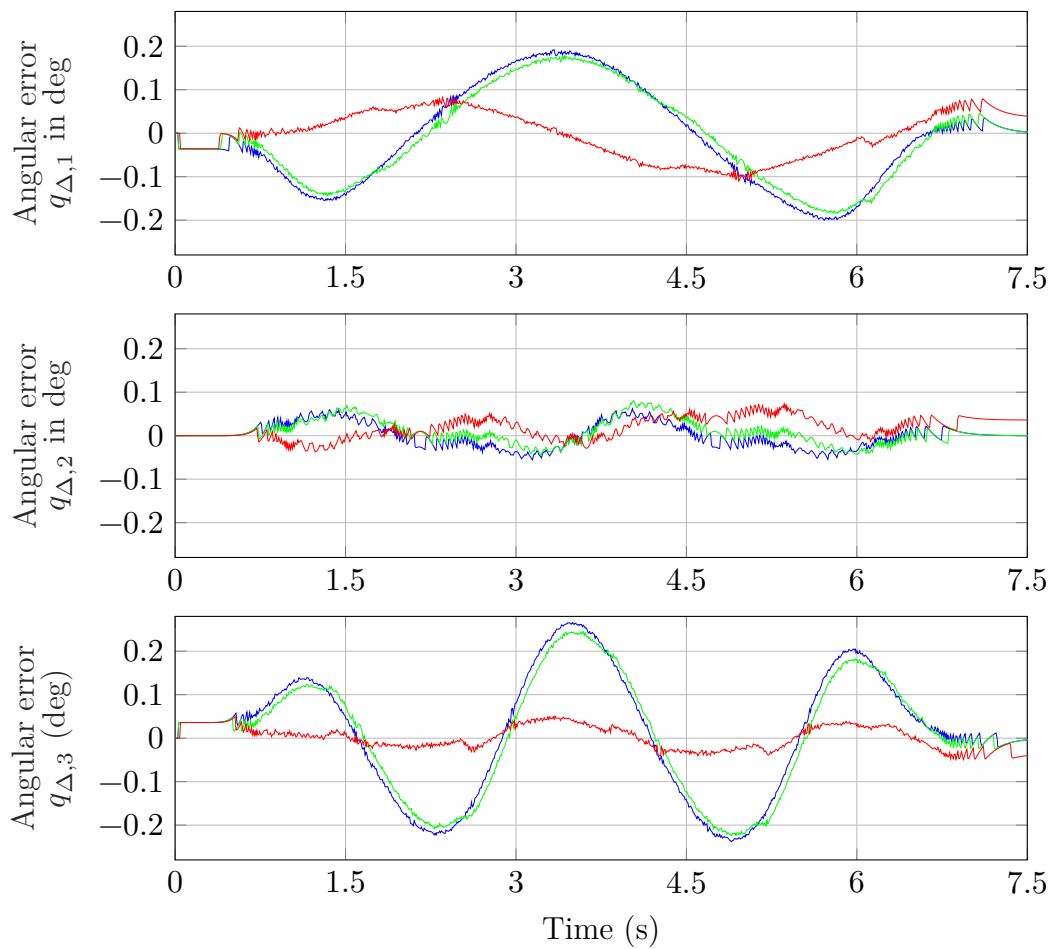


Figure 9: Measured angular error $q_{\Delta,i}$ for each joint in the second robot iteration. Error with C-FB in blue, MB-FB in green and FB-FF in red.

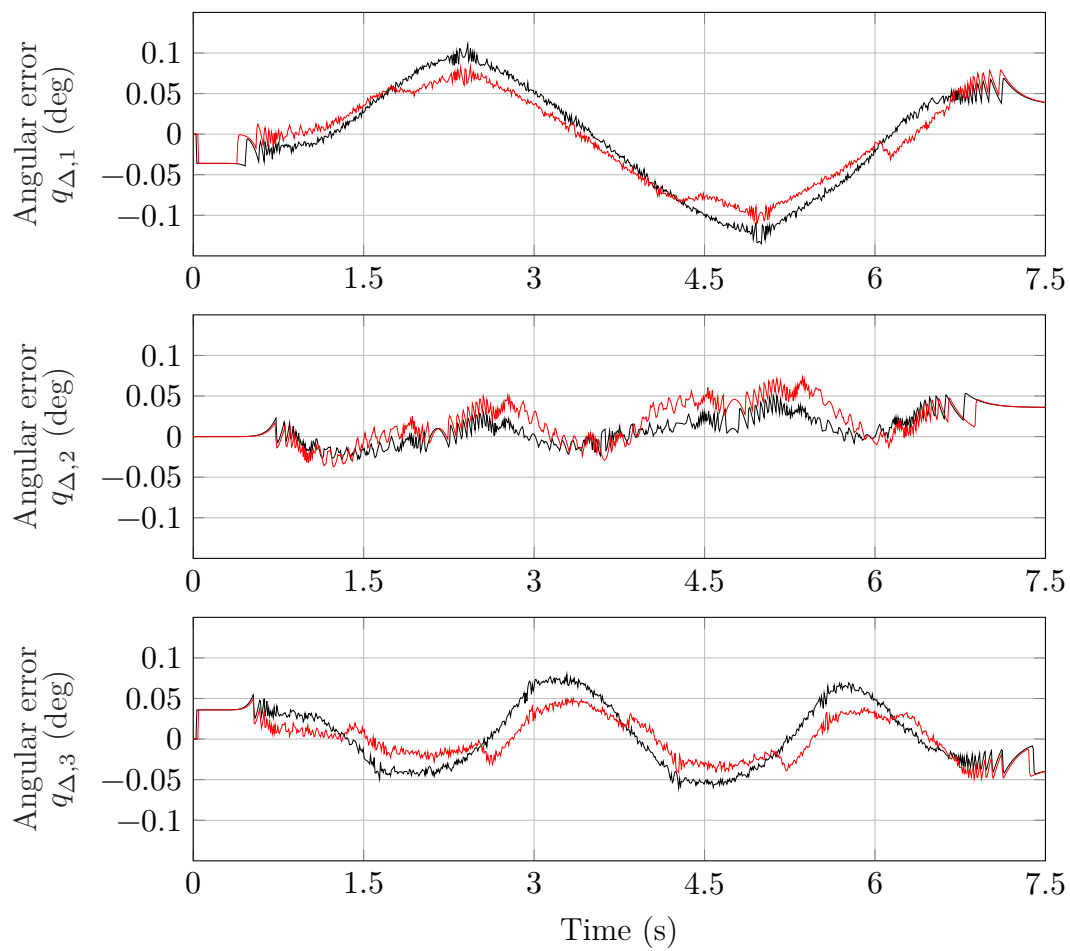


Figure 10: Measured angular error $q_{\Delta,i}$ for each joint in the second robot iteration. FB-FF in red and R-FF in black.

C.3 Improvements Based on Experimental Results

To evaluate the feed-forward controller, the most dynamic measurements of the robot, the torque measurements are investigated. Fig. 11 shows the measured motor torque for the proposed flatness-based controller, including a modified feedback controller for all major joints. For comparison, the pure feed-forward torque is added in the same figure. An ideal robot model applied in the robot controller would reduce the contribution of the feedback controller, making the predicted and measured torque identical. In this scenario, the feedback controller could focus solely on external disturbances (not model errors) and be tuned in a stiffer configuration.

This measurement in Fig. 11 demonstrates that the elastic joint model captures the major dynamics of joint 1. However, regarding joints 2 and 3, the measured motor torque significantly differs from the flatness-based calculation.

Note that in standstill phases, the measured motor torques at the beginning of the movement (0 s) and at its end (7.5 s) differ significantly. The measured joint angles, see Fig. 9, confirm that the end pose is identical to the start pose within the measurement resolution of ± 0.017 deg. Hence, gravitational and hydraulic spring loads on joint 2, which depend only on the pose, should be identical at the beginning and end. Inertia, Coriolis, centripetal, and friction forces are zero at a standstill. This leads to the conclusion that significant asymmetrical friction is present in joint 2 and 3.

For joint 2, the hydraulic counterbalance reduces the gravity torque. Unfortunately, it is not common in the robotic's community to model hydraulic loads. Therefore, the gravity torque and the HWC are neglected for joint 2 in the results of the second iteration. However, joint 2 in Fig. 11 demonstrates significant structural differences in the dynamic torque. Hence, a HWC model is included in the next robot model iteration.

The feed-forward torque error in joint 1 and 3 can be caused by

1. inertia, Coriolis, centripetal effects,
2. gravitational loads,
3. asymmetrical friction, Coulomb friction, degressive friction, or
4. backlash, lost motion, and linear joint elasticity.

Regarding the first group, the parameters are based on the CAD model. Therefore, the parameter uncertainty is low. Furthermore, errors in the inertia, Coriolis, and centripetal part of the model would lead to a different signature of torque errors, for example, modification of the error frequency or introducing additional changes in error direction, which cannot be observed in Fig. 11. Moreover, gravitational loads cannot affect joint 1 by definition. Regarding joint 3, the gravitational model parameters are based on the CAD

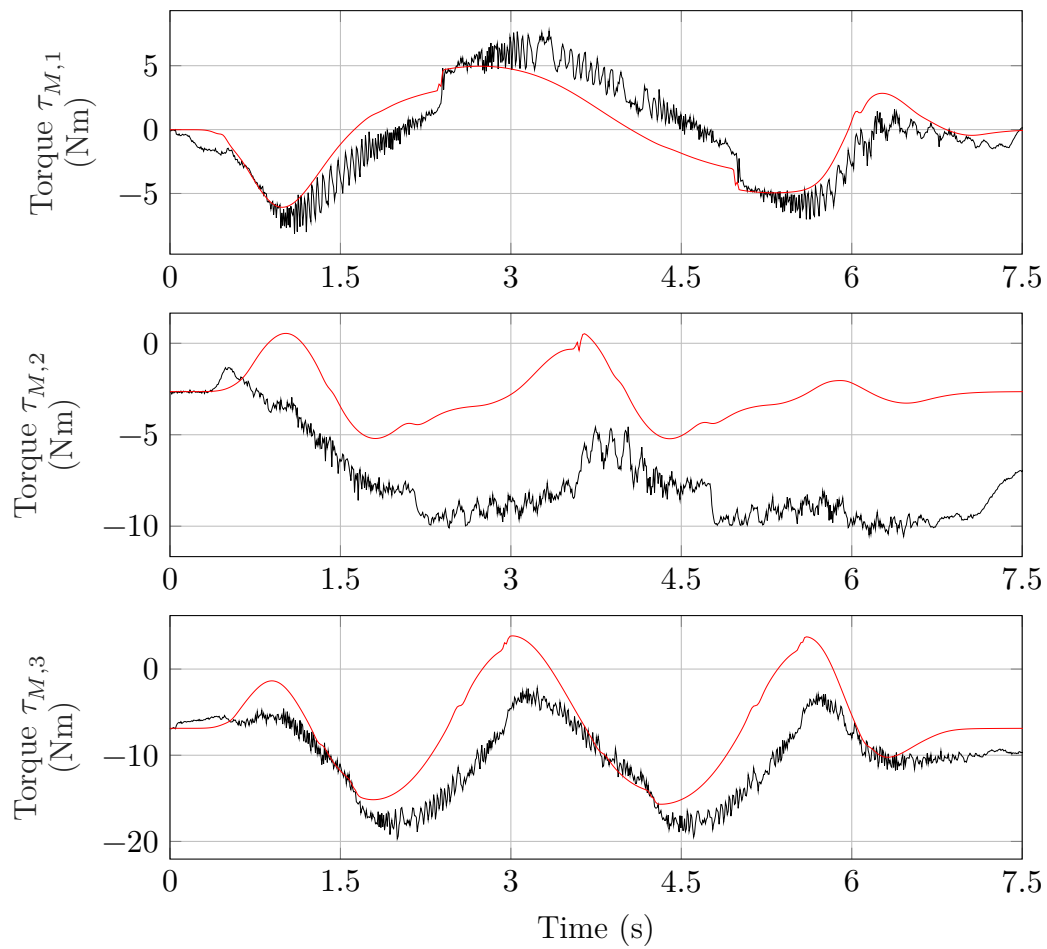


Figure 11: Measured motor torque $\tau_{M,i}$ of each joint in black. The stand-alone flatness-based feed-forward torque (FB-FF) executed is presented for comparison in red.

model, with only a minor uncertainty. The final group, backlash, lost motion, and linear elasticity, cannot cause the observed torque errors, as those effects occur only during backlash with a high amplitude. This leads to the conclusion that an advanced friction model is required to optimize the feed-forward controller.

D Additional Experiments Data-based Model

The following presents additional figures related to section 5.3.

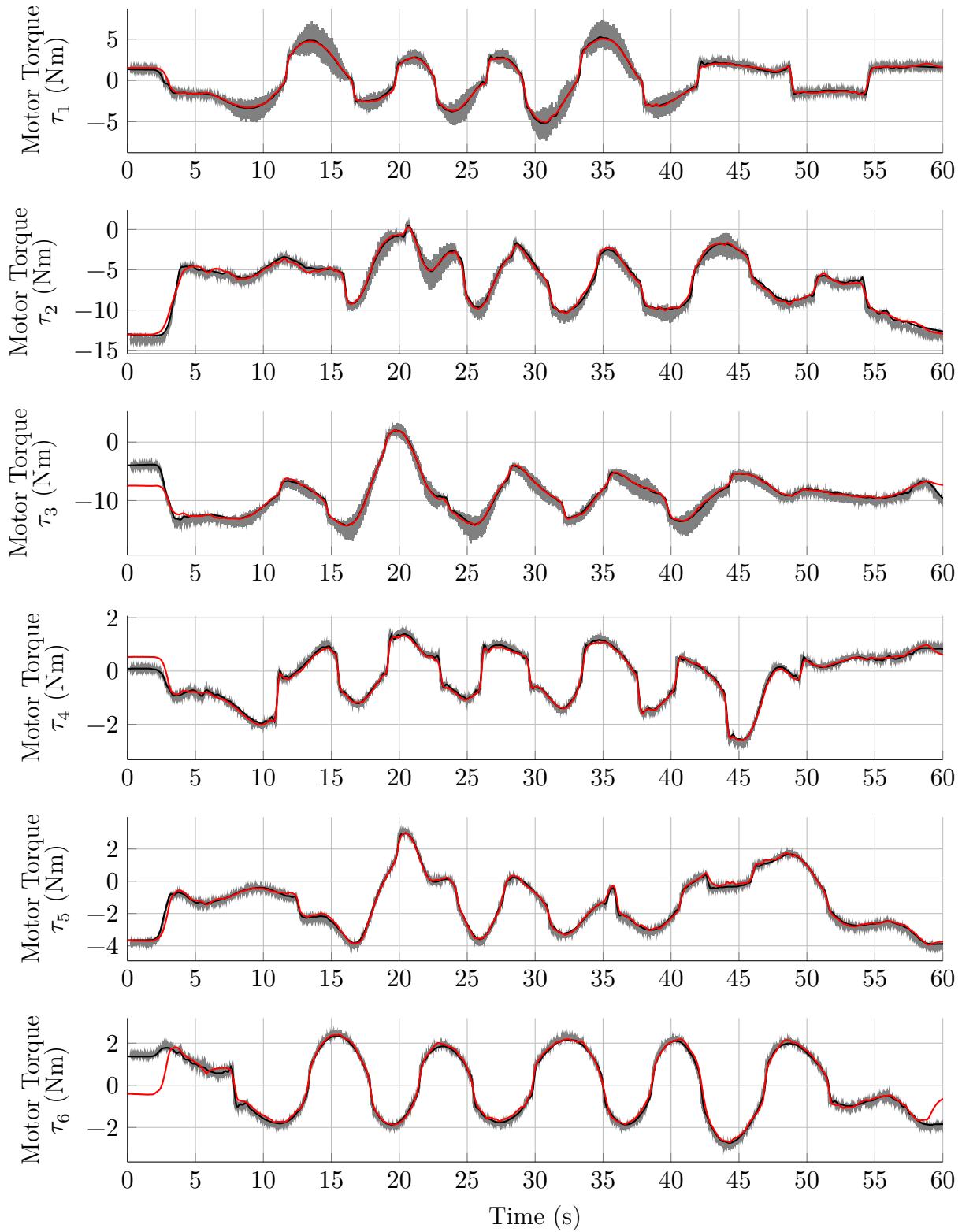


Figure 12: Performance of the discrete-time MLP model on the benchmark training data. Trained on a small dataset. Red: Model prediction. Gray: Motor torque measurements. Black: Filtered motor torque measurements.

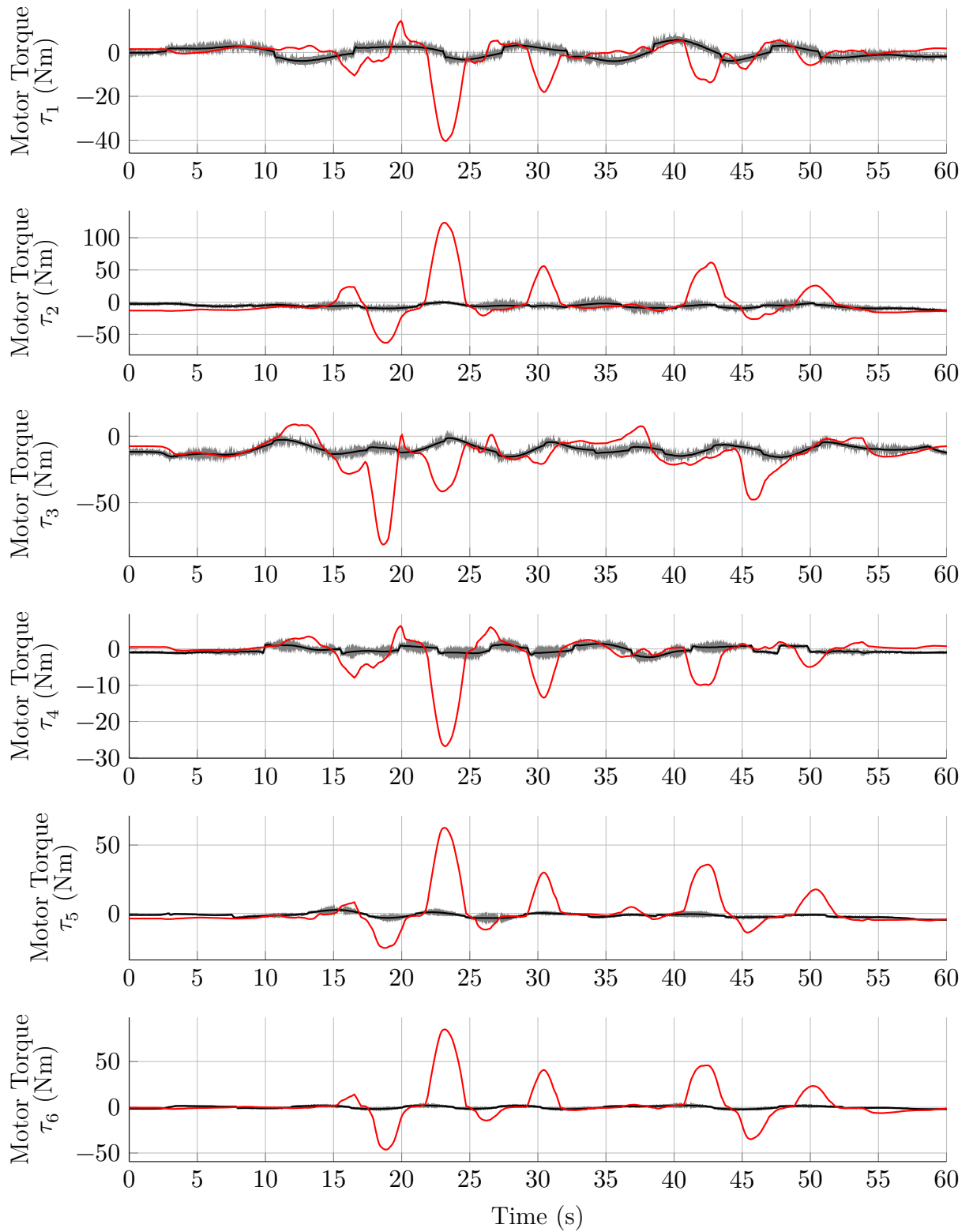


Figure 13: Performance of the discrete-time MLP model on the benchmark test data. Trained on a small dataset. Red: Model prediction. Gray: Motor torque measurements. Black: Filtered motor torque measurements.

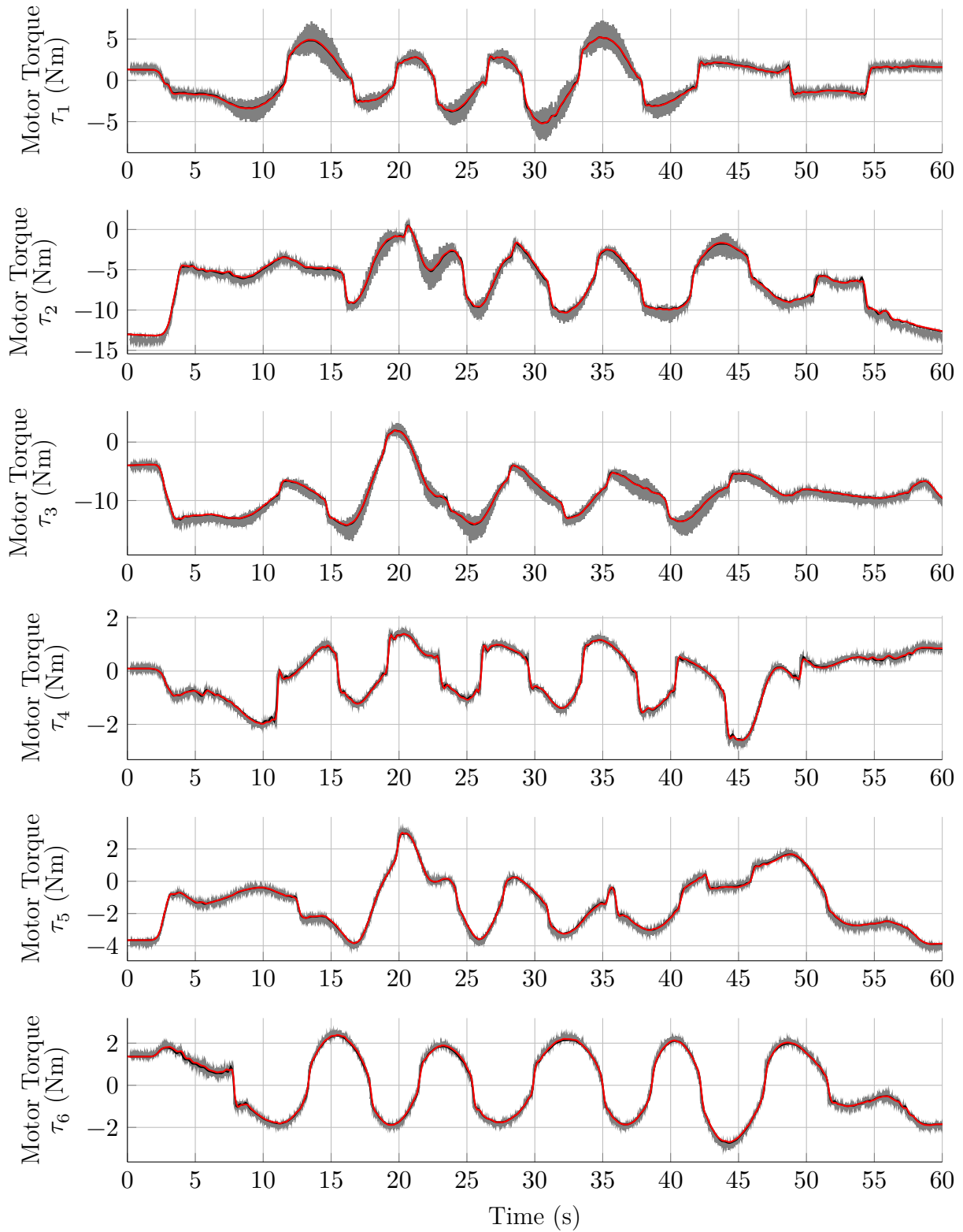


Figure 14: Performance of the continuous-time MLP model on the benchmark training data. Trained on a small dataset. Red: Model prediction. Gray: Motor torque measurements. Black: Filtered motor torque measurements.

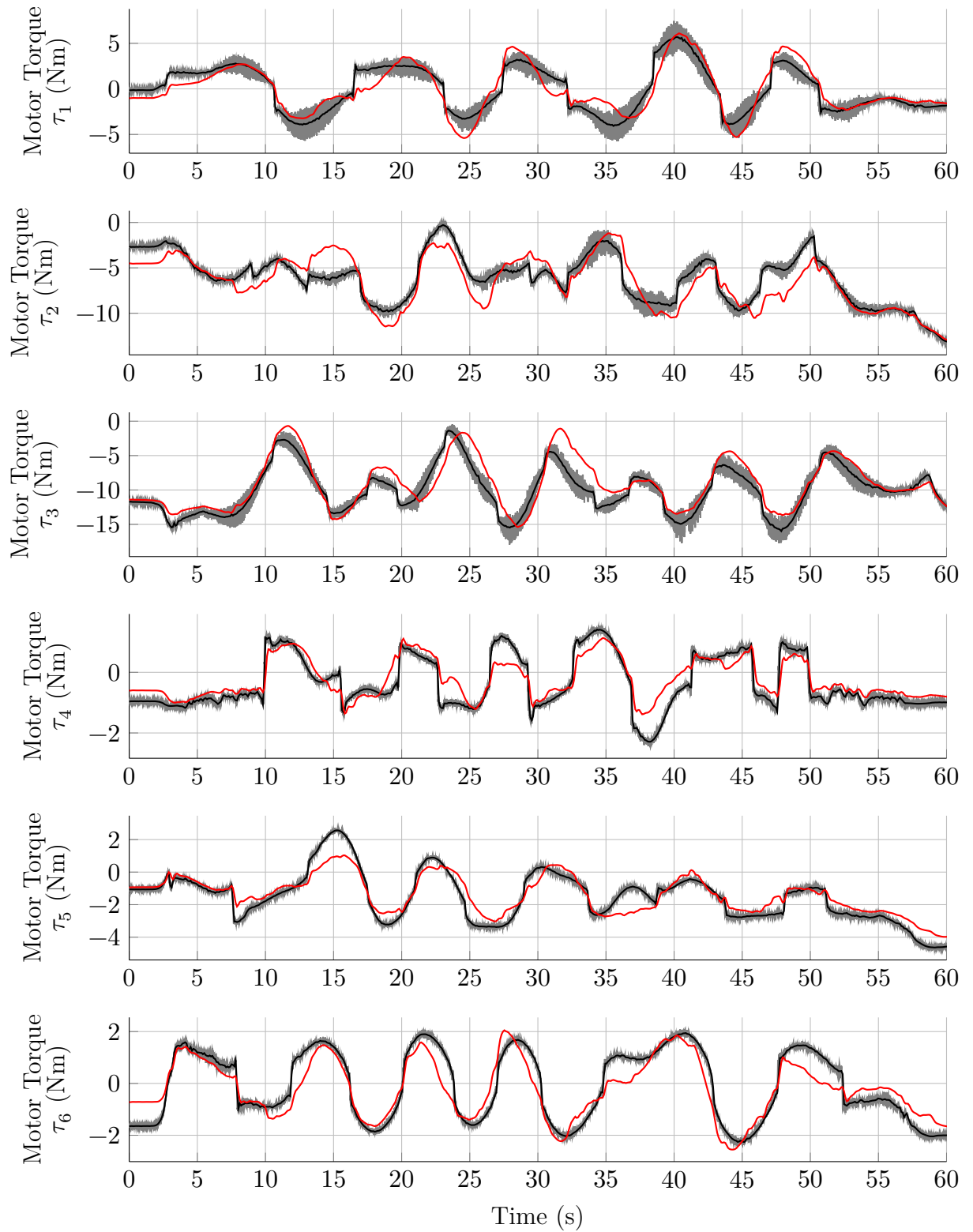


Figure 15: Performance of the continuous-time MLP model on the benchmark test data. Trained on a small dataset. Red: Model prediction. Gray: Motor torque measurements. Black: Filtered motor torque measurements.

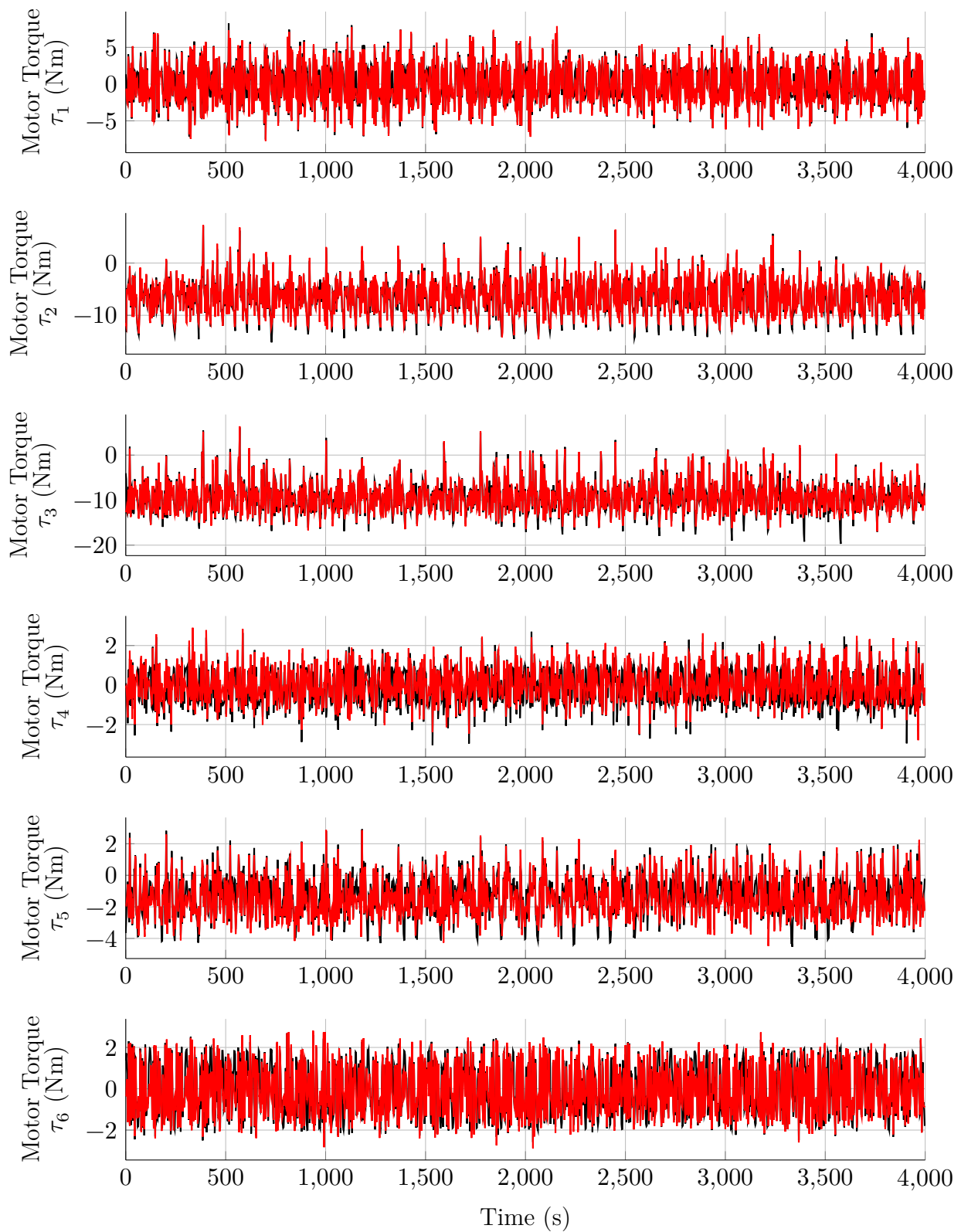


Figure 16: Performance of the discrete-time MLP model on the benchmark training data. Trained on many data points. Red: Model prediction. Gray: Motor torque measurements. Black: Filtered motor torque measurements.

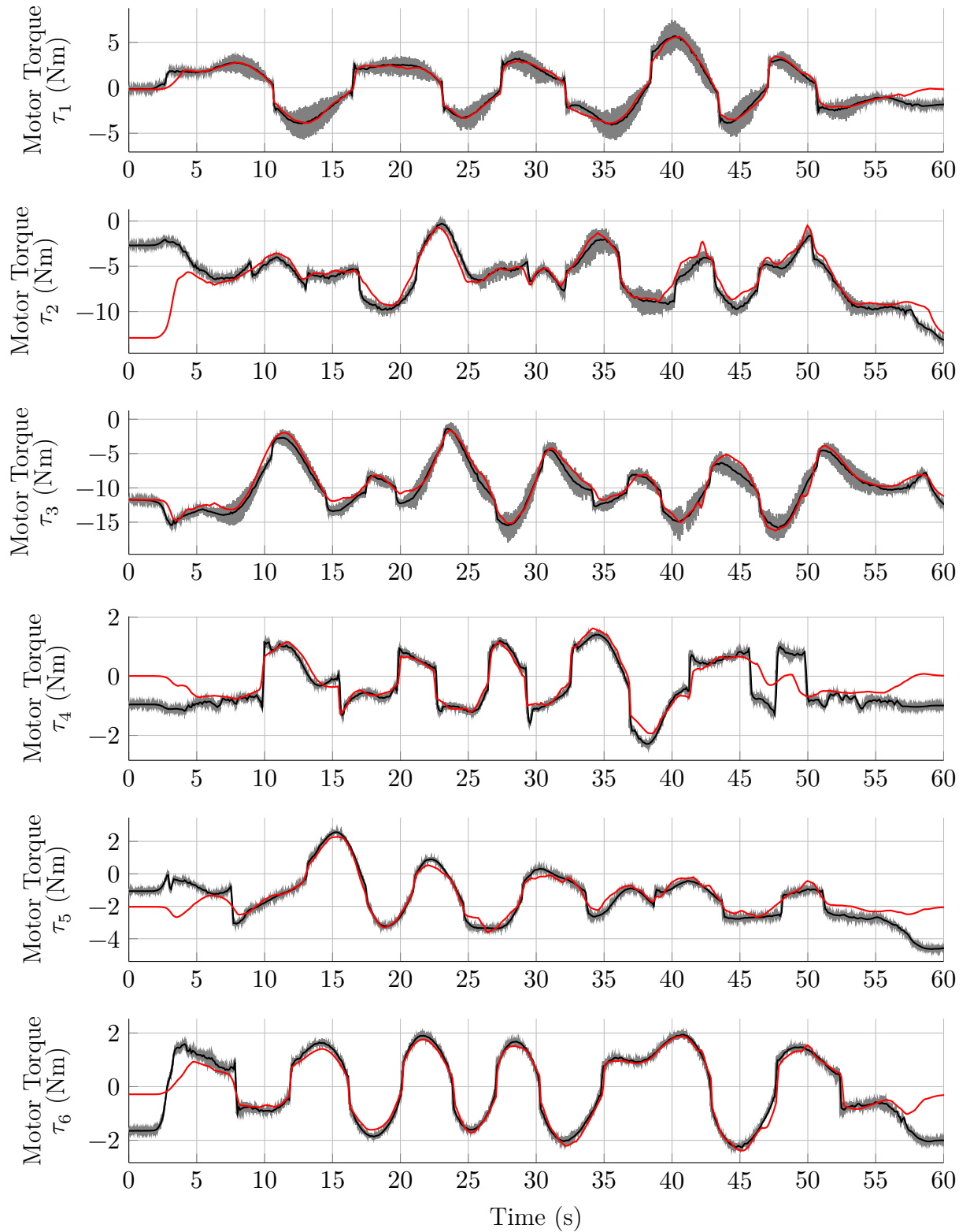


Figure 17: Performance of the discrete-time MLP model on the benchmark test data. Trained on many data points. Red: Model prediction. Gray: Motor torque measurements. Black: Filtered motor torque measurements.

E Source Code

E.1 Dynamic Model Update

Listing 1: Dynamic model update (Python).

```

1      """Update the robot model if there is a change in position
2      or velocity exceeds a threshold. Parameters must be persistent.
3      This can be achieved using global variables or class member
4      variables (recommended).
5      Norm() refers to the L2 norm definition.
6
7      q          current vector of joint angles
8      qd         current vector of joint velocities
9      first_eval  boolean indicator of a first function call
10     q_last     last vector of joint angles
11     qd_last    last vector of joint velocities
12     q_threshold threshold vector of joint angles
13     qd_threshold threshold vector of joint velocities
14     M         inertia matrix
15     C         coriolis and centripetal matrix
16     g         gravitational load vector
17     tau_hyd   hydraulic counter weight torque vector
18     """
19
20     def update_model(q: float_vector ,
21                    qd: float_vector ,
22                    first_eval: bool ,
23                    ) -> tuple[float_matrix ,
24                               float_matrix ,
25                               float_vector ,
26                               float_vector ]:
27
28         if first_eval \
29             or norm(q - q_last) > q_threshold \
30             or norm(qd - qd_last) > qd_threshold:
31
32             M, C, g, tau_hyd = model_core(q, qd)
33             q_last, qd_last = q, qd
34
35         return M, C, g, tau_hyd

```

E.2 Advanced Friction Flatness-based Feed-forward Control

Listing 2: Advanced friction flatness-based feed-forward control (Matlab).

```

1  function [tau_ff, qm_ref, qm_d_ref] = FeedForward(t, para)
2      %#codegen
3
4      % t          time (auto generated, not used)
5      % J          motor inertia
6      % Mii       link inertia
7      % c         torsional gearbox stiffness
8      % f_a       degressive friction parameter A
9      % f_b       degressive friction parameter B
10     % f_coul     Coulomb friction parameter
11     % f_vis     viscous friction parameter
12     % q0        joint angle
13     % q1        joint velocity
14     % q2        joint acceleration
15     % q3        joint jerk
16     % q4        joint jerk derivative
17     % qB        backlash angle
18     % sE        elastic joint smoothness
19     % sF        Coulomb friction smoothness
20     % tau_a     acceleration torque
21     % tau_c     coriolis and centripetal torque
22     % tau_g     gravitational and hydraulic torque
23     % u         transmission factor
24
25     % This function was generated by the
26     % Symbolic Math Toolbox version 8.5.
27     % 03-Aug-2022 08:55:31
28
29     J = para.J;
30     Mii = para.Mii;
31     c = para.c;
32     f_a = para.f_a;
33     f_b = para.f_b;
34     f_asym = para.f_asym;
35     f_coul = para.f_coul;
36     f_vis = para.f_vis;
37     q0 = para.q0;
38     q1 = para.q1;
39     q2 = para.q2;
40     q3 = para.q3;

```

```

41     q4 = para.q4;
42     qB = para.qB;
43     sE = para.sE;
44     sF = para.sF;
45     tau_a = para.tau_a;
46     tau_c = para.tau_c;
47     tau_g = para.tau_g;
48     u = para.u;
49
50     t2 = Mii.*q2;
51     t3 = Mii.*q3;
52     t4 = Mii.*q4;
53     t5 = f_b.*q1;
54     t6 = f_vis.*q1;
55     t7 = f_vis.*q2;
56     t8 = f_vis.*q3;
57     t9 = q1.*sF;
58     t10 = f_b.^2;
59     t11 = q2.^2;
60     t12 = sF.^2;
61     t15 = 1.0./c;
62     t13 = tanh(t5);
63     t14 = tanh(t9);
64     t16 = t13.^2;
65     t17 = t14.^2;
66     t18 = f_a.*t13;
67     t19 = f_coul.*t14;
68     t20 = t16-1.0;
69     t21 = t17-1.0;
70     t32 = f_asym+t2+t6+t18+t19+tau_a+tau_c+tau_g;
71     t22 = f_a.*f_b.*q2.*t20;
72     t23 = f_a.*f_b.*q3.*t20;
73     t24 = f_coul.*q2.*sF.*t21;
74     t25 = f_coul.*q3.*sF.*t21;
75     t30 = t10.*t11.*t18.*t20.*2.0;
76     t31 = t11.*t12.*t19.*t21.*2.0;
77     t33 = sE.*t32;
78     t26 = -t22;
79     t27 = -t23;
80     t28 = -t24;
81     t29 = -t25;
82     t34 = tanh(t33);
83     t35 = t34.^2;
84     t37 = t3+t7+t26+t28;

```

```

85     t38 = t4+t8+t27+t29+t30+t31;
86     t36 = t35 - 1.0;
87     tau_ff = t32./u+J.*u.*(q2+t15.*t38-q_back.*sE.*t36.* ...
88         t38+q_back.*sE.^2.*t34.*t36.*t37.^2.*2.0);
89     if nargout > 1
90         qm_ref = u.*(q0+q_back.*t34+t15.*t32);
91     end
92     if nargout > 2
93         qm_d_ref = u.*(q1+t15.*t37-q_back.*sE.*t36.*t37);
94     end
95     end

```

E.3 Feed-Forward Controller Code

The flatness-based control algorithm, obtained by *MATLAB Symbolic Toolbox* with additional manual modifications and transferred into C is presented. All exponential functions are limited to $\exp(\cdot) \leq 1e30$ to avoid Not-A-Number and infinity errors when using division. The code is extended for estimating all robot joints. In [WGR20] there is a minor formatting error, which is corrected here, it must be $t[13]$ and $t[14]$ instead of $t13$ and $t14$.

Listing 3: Flatness-based feed-forward controller(C++).

```

1     double [3][6] res = feedforward(
2         double [6] q0, double [6] q1,
3         double [6] q2, double [6] q3,
4         double [6] q4, double [6] qB,
5         double [6] J, double [6] M,
6         double [6] sE, double [6] sF,
7         double [6] f_v, double [6] f_c,
8         double [6] u, double [6] c,
9         double [6] tau_c, double [6] tau_g,
10        double [6] tau_a)
11    {
12
13        /* Description */
14        // Computes the flatness-based torque
15        // and motor reference variables for
16        // each axis of an industrial robot.
17
18        // The equations are defined in Section 4.3.3.
19
20        /* Input */
21        // Variables and parameters of

```

```

22 // the current state for all 6 axis.
23 // Type double.
24
25 /* Output */
26 // Result matrix. First column is
27 // the flatness-based torque in Nm,
28 // second column is the motor reference
29 // angle in rad, third column is
30 // the motor reference velocity in rad/s.
31 // Type double.
32
33 /* define variables */
34 // define axis index
35 int i_ax;
36 // define temporary variables
37 double t[36];
38 // limit exponential function
39 double exp_max = 1e30;
40 // define variables for each axis
41 double M_i, J_i, tau_c_i, tau_g_i, tau_a_i,
42         c_i, qB_i, sF_i, sE_i, f_v_i, f_c_i,
43         u_i, q0_i, q1_i, q2_i, q3_i, q4_i;
44
45 /* for each axis */
46 for (i_ax = 0; i_ax < 6; i_ax++)
47 {
48
49     /* get parameters for each axis */
50     // link inertia ,  $\text{kgm}^2$ 
51     M_i = M[i_ax];
52     // motor inertia ,  $\text{kgm}^2$ 
53     J_i = J[i_ax];
54     // backlash angle , rad
55     qB_i = qB[i_ax];
56     // coriolis and centripetal torque , Nm
57     tau_c_i = tau_c[i_ax];
58     // gravity torque , Nm
59     tau_g_i = tau_g[i_ax];
60     // acceleration torque , Nm
61     tau_a_i = tau_a[i_ax];
62     // transmission factor , without unit
63     u_i = u[i_ax];
64     // stiffness factor , Nm/rad
65     c_i = c[i_ax];

```



```

66      // friction smoothness factor , s/rad
67      sF_i = sF[i_ax];
68      // elastic torque smoothness factor ,
69      // rad/Nm
70      sE_i = sE[i_ax];
71      // viscoul friction coefficient , Nms/rad
72      f_v_i = f_v[i_ax];
73      // coulomb friction coefficient , Nm
74      f_c_i = f_c[i_ax];
75
76      /* get variables for each axis */
77      q0_i = q0[i_ax]; // link angle
78      q1_i = q1[i_ax]; // link velocity
79      q2_i = q2[i_ax]; // link acceleration
80      q3_i = q3[i_ax]; // link jerk
81      q4_i = q4[i_ax]; // link jerk derivative
82
83      /* compute feed-forward torque ,
84      motor angle and motor velocity */
85      t[0] = M_i*q2_i;
86      t[1] = M_i*q3_i;
87      t[2] = M_i*q4_i;
88      t[3] = f_v_i*q1_i;
89      t[4] = f_v_i*q2_i;
90      t[5] = f_v_i*q3_i;
91      t[6] = q1_i*sF_i;
92      t[7] = q2_i*q2_i;
93      t[8] = sE_i*sE_i;
94      t[9] = sF_i*sF_i;
95      t[10] = -f_c_i;
96      t[11] = 1.0/c_i;
97      t[12] = t[6]*2.0;
98      t[13] = -t[6];
99      t[14] = -t[12];
100     t[15] = fmin(exp(t[13]), exp_max);
101     t[16] = fmin(exp(t[14]), exp_max);
102     t[17] = t[15]+1.0;
103     t[18] = 1.0/t[17];
104     t[19] = t[18]*t[18];
105     t[20] = t[18]*t[18];
106     t[21] = f_c_i*t[18]*2.0;
107     t[22] = f_c_i*q2_i*sF_i
108             *t[15]*t[19]*2.0;
109     t[23] = f_c_i*q3_i*sF_i

```

```

110         *t [15]*t [19]*2.0;
111     t [24] = f_c_i*t [7]*t [9]
112         *t [15]*t [19]*2.0;
113     t [25] = f_c_i*t [7]*t [9]
114         *t [16]*t [20]*4.0;
115     t [26] = t [0]+t [3]+t [10]+t [21]
116         +tau_c_i+tau_g_i+tau_a_i;
117     t [27] = -t [24];
118     t [28] = sE_i*t [26];
119     t [29] = t [1]+t [4]+t [22];
120     t [30] = -t [28];
121     t [31] = t [29]*t [29];
122     t [32] = t [2]+t [5]+t [23]+t [25]
123         +t [27];
124     t [33] = fmin (exp (t [30]), exp_max);
125     t [34] = t [33]+1.0;
126     t [35] = 1.0/(t [34]*t [34]);
127
128     /* feed-forward torque in Nm */
129     res [0][i_ax] = t [26]/u_i+J_i*u_i*
130         (q2_i+t [11]*t [32]
131         +qB_i*sE_i*t [33]
132         *t [35]*t [32]*2.0
133         -qB_i*t [8]*t [31]
134         *t [33]*t [35]*2.0
135         +qB_i*t [8]*t [31]
136         *1.0/(t [34]*t [34]
137         *t [34])*fmin (exp (t [28]
138         *-2.0), exp_max)*4.0);
139
140     /* motor reference angle in rad */
141     res [1][i_ax] = u_i*(q0_i-qB_i
142         +t [11]*t [26]
143         +(qB_i*2.0)/t [34]);
144
145     /* motor reference velocity in rad/s */
146     res [2][i_ax] = u_i*(q1_i+t [11]
147         *t [29]+qB_i*sE_i
148         *t [29]*t [33]
149         *t [35]*2.0);
150 }
151 }

```

E.4 Weak Stability Barrier

Listing 4: Weak Stability Barrier (Python).

```

1  def stability_barrier(self) -> torch.Tensor:
2      """Compute the weak stability barrier.
3
4      Returns:
5          torch.Tensor: Scalar barrier value for loss function.
6      """
7
8      # get linear weight matrix of neural network
9      A = self.net_dx_linear[0].weight
10
11     # get number of states
12     n_states = A.shape[0]
13
14     # create empty list for all coefficients
15     must_be_positive = []
16
17     # define constants for barrier function
18     stability_offset = torch.tensor(1.0e-2)
19     stability_scaling = torch.tensor(1.0e+6)
20     stability_barrier = torch.tensor(1.0e+12)
21
22     # estimate Sylvester criterion for all submatrices
23     for idx in numpy.arange(n_states):
24
25         # get submatrix
26         submatrix = A[0:idx + 1, 0:idx + 1]
27
28         # estimate determinant of submatrix
29         det = torch.det(submatrix)
30
31         # add coefficient to list
32         must_be_positive.append((-1)**(idx + 1) * det)
33
34     # get minimum value of list ,
35     # multiply with -1 for loss functions greater than zero
36     must_be_negative = -1 * min(must_be_positive)
37
38     # if bound is exceeded, return large value
39     if must_be_negative >= 0.0:
40         return stability_barrier

```

```

41
42     # if close to bound, linear increase penalty value
43     if must_be_negative >= -1 * stability_offset:
44         return stability_scaling \
45             * torch.relu(must_be_negative + stability_offset)
46
47     # if constraint is satisfied, return zero
48     return tensor(0.0)

```

E.5 Differential Algebraic Equation Barrier

Listing 5: Differential Algebraic Equation Barrier (Python).

```

1  def dae_barrier(self, input: torch.Tensor) -> torch.Tensor:
2      """Compute a differential-algebraic-equation barrier.
3
4      Args:
5          input (torch.Tensor): Network input data.
6          Contains state, augmented state,
7          and exogenous information.
8          3 Dimensions, defined as
9          [n_batches
10         x (n_states + n_aug + n_exo_input)
11         x n_time_steps]
12
13     Returns:
14         torch.Tensor: Scalar barrier
15         value for loss function.
16     """
17
18     # get number of time steps
19     nt = input.shape[-1]
20
21     # initialize mean squared error
22     mse = tensor(0.0)
23
24     # constant
25     scaling_dae = torch.tensor(1e-3)
26
27     # for all time steps t
28     for t in numpy.arange(nt):
29
30         # compute net_dae for each time step

```

```
31         out = self.net_dae(input[:, :, t])
32
33         # compute MSE loss for each time step, sum up
34         mse += nn.functional.mse_loss(
35             out, 0 * out, reduction='mean')
36
37     return scaling_dae * mse / nt
```


Bibliography

Articles and Monographs

- [21] ISO 8373 : 2021 Robotics, tech. rep., ISO, 2021.
- [Ah10] Ahmed, N. K.; Atiya, A. F.; Gayar, N. E.; El-Shishiny, H.: An Empirical Comparison of Machine Learning Models for Time Series Forecasting. *Econometric Reviews* 29/5-6, pp. 594–621, 2010, ISSN: 0747-4938.
- [AM21] AMO Automatisierung Messtechnik Optik, St. Peter am Hart, Austria: Modulare Winkelmessgeräte nach dem induktiven AMOSIN® – Messprinzip, <https://www.amo-gmbh.com/>, 2021.
- [Am93] Amari, S.-i.: Backpropagation and stochastic gradient descent method. *Neurocomputing* 5/4-5, pp. 185–196, 1993.
- [An19] Andersson, J. A. E.; Gillis, J.; Horn, G.; Rawlings, J. B.; Diehl, M.: CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation* 11/1, pp. 1–36, 2019.
- [An23] Ansari, A. F.; Heng, A.; Lim, A.; Soh, H.: Neural Continuous-Discrete State Space Models for Irregularly-Sampled Time Series./, 2023, arXiv: 2301.11308 [cs.LG].
- [AOH07] Albu-Schäffer, A.; Ott, C.; Hirzinger, G.: A Unified Passivity-based Control Framework for Position, Torque and Impedance Control of Flexible Joint Robots. *The International Journal of Robotics Research* 26/1, pp. 23–39, 2007, ISSN: 0278-3649.
- [BCS18] Birpoutsoukis, G.; Csurscia, P. Z.; Schoukens, J.: Efficient multidimensional regularization for Volterra series estimation. *Mechanical Systems and Signal Processing* 104/, pp. 896–914, 2018, ISSN: 08883270.
- [BDS21] Beise, H.-P.; Da Cruz, S. D.; Schröder, U.: On decision regions of narrow deep neural networks. *Neural Networks* 140/, pp. 121–129, 2021.
- [BIS06] Bona, B.; Indri, M.; Smaldone, N.: Rapid Prototyping of a Model-Based Control With Friction Compensation for a Direct-Drive Robot. *IEEE/ASME Transactions on Mechatronics* 11/5, pp. 576–584, 2006, ISSN: 1083-4435.

- [BKH16] Ba, J. L.; Kiros, J. R.; Hinton, G. E.: Layer Normalization, 2016, URL: <http://arxiv.org/pdf/1607.06450v1>.
- [Bo12] Bottou, L.: Stochastic gradient descent tricks. In: Neural networks: Tricks of the trade. Springer, pp. 421–436, 2012.
- [BP02] Barabanov, N. E.; Prokhorov, D. V.: Stability analysis of discrete-time recurrent neural networks. *IEEE transactions on neural networks* 13/2, pp. 292–303, 2002, ISSN: 1045-9227.
- [Br16] Brüning, J.; Denkena, B.; Dittrich, M.; Park, H.-S.: Simulation Based Planning of Machining Processes with Industrial Robots. *Procedia Manufacturing* 6/December, pp. 17–24, 2016, ISSN: 23519789.
- [Br18] Brunete, A.; Gambao, E.; Koskinen, J.; Heikkilä, T.; Kaldestad, K. B.; Tyapin, I.; Hovland, G.; Surdilovic, D.; Hernando, M.; Bottero, A.; Anton, S.: Hard material small-batch industrial machining robot. *Robotics and Computer-Integrated Manufacturing* 54/April 2018, pp. 185–199, 2018, ISSN: 07365845.
- [BR21a] BR Automation, Bad Homburg, Germany: ACOPOSmulti user’s manual, <https://www.br-automation.com/en/products/motion-control/acoposmulti/>, 2021.
- [BR21b] BR Automation, Bad Homburg, Germany: APC910 user’s manual, <https://www.br-automation.com/en/products/industrial-pcs/automation-pc-910/>, 2021.
- [BST22] Beintema, G. I.; Schoukens, M.; Tóth, R.: Deep subspace encoders for continuous-time state-space identification, 2022, arXiv: 2204.09405v1 [cs.LG].
- [BST23] Beintema, G. I.; Schoukens, M.; Tóth, R.: Continuous-time identification of dynamic state-space models by deep subspace encoding, 2023, arXiv: 2204.09405v2 [cs.LG].
- [BTC18] Bahloul, A.; Tliba, S.; Chitour, Y.: Dynamic parameters identification of an industrial robot with and without payload. *IFAC-PapersOnLine* 51/15, pp. 443–448, 2018.
- [BTS21a] Beintema, G.; Tóth, R.; Schoukens, M.: Nonlinear state-space identification using deep encoder networks./, pp. 241–250, 2021.
- [BTS21b] Beintema, G. I.; Tóth, R.; Schoukens, M.: Nonlinear State-space Model Identification from Video Data using Deep Encoders, 2021, URL: <http://arxiv.org/pdf/2012.07721v3>.

- [Bu16] Butcher, J. C.: Numerical methods for ordinary differential equations. Wiley, Chichester, UK, 2016, ISBN: 9781119121534.
- [BWS15] Brahma, P. P.; Wu, D.; She, Y.: Why deep learning works: A manifold disentanglement perspective. *IEEE transactions on neural networks and learning systems* 27/10, pp. 1997–2008, 2015.
- [CBG90] Chen, S.; Billings, S. A.; Grant, P. M.: Nonlinear system identification using neural networks. *International Journal of Control* 51/6, pp. 1191–1214, 1990, ISSN: 0020-7179.
- [Ch14] Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling./, 2014, arXiv: 1412.3555 [cs.NE].
- [CH17] Cordes, M.; Hintze, W.: Offline simulation of path deviation due to joint compliance and hysteresis for robot machining. *The International Journal of Advanced Manufacturing Technology* 90/1-4, pp. 1075–1083, Apr. 2017, ISSN: 0268-3768, URL: <http://link.springer.com/10.1007/s00170-016-9461-z>.
- [Ch18] Chen, R. T. Q.; Rubanova, Y.; Bettencourt, J.; Duvenaud, D.: Neural Ordinary Differential Equations. *Advances in neural information processing systems*/31, 2018, URL: <http://arxiv.org/pdf/1806.07366v5>.
- [Co14] Correia, A.; Matias, J.; Mestre, P.; Serodio, C.: Comparison of Some Penalty and Barrier Techniques in Direct Search Methods. In: 2014 14th International Conference on Computational Science and Its Applications. IEEE, pp. 191–195, 2014, ISBN: 978-1-4799-4264-0.
- [Co17] Corke, P.: Robotics, vision and control: fundamental algorithms in MATLAB. Springer, 2017, ISBN: ISBN 978-3-319-54413-7.
- [Co84] Cooper, G. J.: A Generalization of Algebraic Stability for Runge-Kutta Methods. *IMA Journal of Numerical Analysis*/, 1984.
- [DB16] De Luca, A.; Book, W. J.: Robots with Flexible Elements. In: Springer Handbook of Robotics. Springer International Publishing, Cham, pp. 243–282, 2016.
- [DDT19] Dupont, E.; Doucet, A.; Teh, Y. W.: Augmented Neural ODEs. *Advances in neural information processing systems*/32, 2019, URL: <http://arxiv.org/pdf/1904.01681v3>.
- [De10] Deflorian, M.: On Runge-Kutta neural networks: Training in series-parallel and parallel configuration. 49th IEEE Conference on Decision and Control/, 2010.

- [De11a] Deflorian, M.: Runge-Kutta Neural Networks for identification of dynamic models at the testbed. In: Proceedings of the 6th Conference Design of Experiments (DoE) in Engine Development. 2011.
- [De11b] Deflorian, Michael: Versuchsplanung und Methoden zur Identifikation zeitkontinuierlicher Zustandsraummodelle am Beispiel des Verbrennungsmotors, Ph.D. Thesis, München: Technische Universität München, 2011.
- [De11c] Devlieg, R.: High-Accuracy Robotic Drilling/Milling of 737 Inboard Flaps. SAE International Journal of Aerospace 4/2, pp. 1373–1379, 2011, ISSN: 19463855.
- [De20] Demeester, T.: System identification with time-aware neural sequence models. In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. 04, pp. 3757–3764, 2020.
- [Di18] Ding, L.; Li, X.; Li, Q.; Chao, Y.: Nonlinear Friction and Dynamical Identification for a Robot Manipulator with Improved Cuckoo Search Algorithm. Journal of Robotics 2018/, pp. 1–10, 2018, ISSN: 1687-9600.
- [DK11] Deflorian, M.; Klöpfer, F.: Schnelles Training neuronaler Netze zur Identifikation nichtlinearer dynamischer Systeme. at - Automatisierungstechnik 59/, pp. 75–83, 2011.
- [DKR10] Deflorian, M.; Klöpfer, F.; Rückert, J.: Online Dynamic Black Box Modelling and Adaptive Experiment Design in Combustion Engine Calibration. IFAC Proceedings Volumes 43/7, pp. 703–708, 2010, ISSN: 14746670.
- [DL98] De Luca, A.; Lucibello, P.: A general algorithm for dynamic feedback linearization of robots with elastic joints. Proceedings - IEEE International Conference on Robotics and Automation 1/May, pp. 504–510, 1998, ISSN: 10504729.
- [DR14] Deflorian, M.; Rungger, M.: Generalization of an input-to-state stability preserving Runge–Kutta method for nonlinear control systems. Journal of Computational and Applied Mathematics 255/, pp. 346–352, 2014, ISSN: 03770427.
- [EK00] Efe, M. O.; Kaynak, O.: A comparative study of soft-computing methodologies in identification of robotic manipulators. Robotics and Autonomous Systems/30: 221–230, 2000.
- [EK99] Efe, M. O.; Kaynak, O.: A comparative study of neural network structures in identification of nonlinear systems. Mechatronics/9(3):287–300, 1999.
- [FN93] Funahashi, K.-i.; Nakamura, Y.: Approximation of dynamical systems by continuous time recurrent neural networks. Neural Networks 6/6, pp. 801–806, 1993, ISSN: 08936080.

- [FP21] Forgione, M.; Piga, D.: Continuous-time system identification with neural networks: Model structures and fitting criteria. *European Journal of Control* 59/, pp. 69–81, 2021, ISSN: 09473580.
- [Fr14] Freising, M.; Kothe, S.; Rott, M.; Susemihl, H.; Hintze, W.: Increasing Accuracy of Industrial Robots in Machining of Carbon Fiber Reinforced Plastics. In: *Lecture Notes in Production Engineering*. September, pp. 115–121, 2014, ISBN: 978-3-319-01964-2.
- [Fr17] Frommknecht, A.; Kuehnle, J.; Effenberger, I.; Pidan, S.: Multi-sensor measurement system for robotic drilling. *Robotics and Computer-Integrated Manufacturing* 47/January, pp. 4–10, 2017, ISSN: 07365845.
- [GB10] Glorot, X.; Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics (AISTATS)/*, 2010.
- [GDH01] Grotjahn, M.; Daemi, M.; Heimann, B.: Friction and rigid body identification of robot dynamics. *International journal of solids and structures* 38/10-13, pp. 1889–1902, 2001.
- [Gi17] Giorgio, G.: Various Proofs of the Sylvester Criterion for Quadratic Forms. *Journal of Mathematics Research* 9/6, p. 55, 2017, ISSN: 1916-9795.
- [GJV13] Gautier, M.; Janot, A.; Vandanjon, P. O.: A New Closed-Loop Output Error Method for Parameter Identification of Robot Dynamics. *IEEE Transactions on Control Systems Technology* 21/2, pp. 428–444, Mar. 2013, ISSN: 1063-6536.
- [GK92] Gautier, M.; Khalil, W.: Exciting Trajectories for Robot Inertial Parameters Identification. *IFAC Proceedings Volumes* 25/15, pp. 585–590, 1992, ISSN: 14746670.
- [Ha18b] Hasani, R. M.; Lechner, M.; Amini, A.; Rus, D.; Grosu, R.: Liquid Time-constant Recurrent Neural Networks as Universal Approximators, 2018, URL: <http://arxiv.org/pdf/1811.00321v1>.
- [Ha19] Hanin, B.: Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics* 7/10, p. 992, 2019.
- [Ha20] Hasani, R.: Interpretable Recurrent Neural Networks in Continuous-time Control Environments, Ph.D. Thesis, TU Wien, 2020.
- [Ha21] Hasani, R.; Lechner, M.; Amini, A.; Rus, D.; Grosu, R.: Liquid time-constant networks. 35/9, pp. 7657–7666, 2021.

- [Ha22b] Hasani, R.; Lechner, M.; Amini, A.; Liebenwein, L.; Ray, A.; Tschalkowski, M.; Teschl, G.; Rus, D.: Closed-form continuous-time neural networks. *Nature Machine Intelligence*/, pp. 1–12, 2022.
- [Ha90] Haykin, S.: *Neural Networks - A Comprehensive Foundation*. Prentice hall, 1990.
- [He16] He, K.; Zhang, X.; Ren, S.; Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Pp. 770–778, 2016.
- [HEV18] Huynh, H. N.; Edouard Riviere-Lorphevre; Verlinden, O.: Multibody modelling of a flexible 6-axis robot dedicated to robotic machining. *The 5 th Joint International Conference on Multibody System Dynamics/July*, pp. 1–18, 2018.
- [HGG15] Hamon, P.; Gautier, M.; Garrec, P.: New dry friction model with load- and velocity-dependence and dynamic identification of multi-DOF robots. *IEEE International Conference on Robotics and Automation (ICRA)/*, pp. 1077–1084, 2015.
- [HJ12] Horn, R. A.; Johnson, C. R.: *Matrix analysis*. Cambridge University Press, Cambridge and New York, 2012, ISBN: 9780521839402.
- [Ho91] Hornik, K.: Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4/2, pp. 251–257, 1991, ISSN: 08936080.
- [HR18] Haber, E.; Ruthotto, L.: Stable architectures for deep neural networks. *Inverse Problems* 34/1, p. 014004, 2018, ISSN: 0266-5611.
- [HS97] Hochreiter, S.; Schmidhuber, J.: Long Short-Term Memory. *Neural Computation*/9, pp. 1735–1780, 1997.
- [Hu20] Huynh, H. N.; Assadi, H.; Rivière-Lorphèvre, E.; Verlinden, O.; Ahmadi, K.: Modelling the dynamics of industrial robots for milling operations. *Robotics and Computer-Integrated Manufacturing* 61/March 2019, p. 101852, 2020, ISSN: 07365845.
- [HW02] Hu, S.; Wang, J.: Global stability of a class of continuous-time recurrent neural networks. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 49/9, pp. 1334–1347, 2002, ISSN: 10577122.
- [IF22] IFR: World Robotics, tech. rep., 2022.
- [ISA15] Iglesias, I.; Sebastián, M. A.; Ares, J. E.: Overview of the State of Robotic Machining: Current Situation and Future Potential. *Procedia Engineering* 132/, pp. 911–917, 2015, ISSN: 18777058.

- [JGB19] Janot, A.; Gautier, M.; Brunot, M.: Data Set and Reference Models of EMPS. 2019 Workshop on Nonlinear System Identification Benchmarks, Eindhoven, The Netherlands/, 2019.
- [Ji18] Jin, L.; Li, S.; Yu, J.; He, J.: Robot manipulator control using neural networks: A survey. *Neurocomputing* 285/, pp. 23–34, 2018, ISSN: 0925-2312, URL: <https://www.sciencedirect.com/science/article/pii/S0925231218300158>.
- [JL92] Johnson, C. T.; Lorenz, R. D.: Experimental identification of friction and its compensation in precise, position controlled mechanisms. *IEEE Transactions on Industry Applications* 28/6, pp. 1392–1398, 1992, ISSN: 0093-9994.
- [JW01] Jiang, Z.-P.; Wang, Y.: Input-to-state stability for discrete-time nonlinear systems. *Automatica*/37, pp. 857–869, 2001, ISSN: 00051098.
- [JW21] Janot, A.; Wensing, P. M.: Sequential semidefinite optimization for physically and statistically consistent robot identification. *Control Engineering Practice* 107/, p. 104699, 2021.
- [KB14] Kingma, D. P.; Ba, J.: ADAM: A Method for Stochastic Optimization, 2014, URL: <http://arxiv.org/pdf/1412.6980v9>.
- [KB20] Karagoz, R.; Batselier, K.: Nonlinear system identification with regularized Tensor Network B-splines. *Automatica* 122/, p. 109300, 2020, ISSN: 00051098.
- [KC19] Kim, J.; Croft, E. A.: Full-state tracking control for flexible joint robots with singular perturbation techniques. *IEEE Transactions on Control Systems Technology* 27/1, pp. 63–73, 2019, ISSN: 1558-0865.
- [KF18] Khodabandehlou, H.; Fadali, M. S.: Nonlinear System Identification using Neural Networks and Trajectory-Based Optimization./, 2018, arXiv: 1804.10346 [eess.SP].
- [KG13] Kammerer, N.; Garrec, P.: Dry friction modeling in dynamic identification for robot manipulators: Theory and experiments. In: 2013 IEEE International Conference on Mechatronics (ICM). IEEE, pp. 422–429, 2013.
- [KG97] Kircanski, N. M.; Goldenberg, A. A.: Experimental study of nonlinear stiffness, hysteresis, and friction effects in robot joints with harmonic drives and torque sensors. *International Journal of Robotics Research* 16/2, pp. 214–239, 1997, ISSN: 02783649.
- [KGL07] Khalil, W.; Gautier, M.; Lemoine, P.: Identification of the payload inertial parameters of industrial manipulators. In: Proceedings 2007 IEEE International Conference on Robotics and Automation. IEEE, pp. 4943–4948, 2007.
- [Kh02] Khalil, H. K.: *Nonlinear Systems*. Prentice hall Englewood Cliffs, 2002.

- [KH91] Krogh, A.; Hertz, J.: A simple weight decay can improve generalization. *Advances in neural information processing systems* 4/, 1991.
- [Ki22] Kidger, P.: On Neural Differential Equations, 2022, URL: <http://arxiv.org/pdf/2202.02435v1>.
- [Kl14] Klimchik, A.; Bondarenko, D.; Pashkevich, A.; Briot, S.; Furet, B.: Compliance Error Compensation in Robotic-Based Milling. In: *Lecture Notes in Electrical Engineering*. Vol. 283, October, pp. 197–216, 2014, ISBN: 978-3-319-03499-7.
- [KPM07] Kermani, M. R.; Patel, R. V.; Moallem, M.: Friction Identification and Compensation in Robotic Manipulators. *IEEE Transactions on Instrumentation and Measurement* 56/6, pp. 2346–2353, 2007, ISSN: 0018-9456.
- [KSH17] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: Imagenet classification with deep convolutional neural networks. *Communications of the ACM* 60/6, pp. 84–90, 2017.
- [Ku08] Kurze, M.: Modellbasierte Regelung von Robotern mit elastischen Gelenken ohne abtriebseitige Sensorik, Dissertation, Technische Universität München, 2008.
- [KU21] KUKA Aktiengesellschaft, Augsburg, Germany: KR QUANTEC KR300 R2500 ultra SE Operating Manual, <https://www.kuka.com/en-de>, 2021.
- [Lä09] Längkvist, M.: Online Identification of Friction Coefficients in an Industrial Robot, Examensarbete, Linköpings universitet, 2009.
- [LC13] Ljung, L.; Chen, T.: What can regularization offer for estimation of dynamical systems? *IFAC International Workshop on Adaptation and Learning in Control and Signal Processing*/, 2013.
- [Le17] Lee, J. H.: Relaxing coherence for modern learning applications, PhD thesis, Georgia Institute of Technology, 2017.
- [LH20] Lechner, M.; Hasani, R.: Learning Long-Term Dependencies in Irregularly-Sampled Time Series, June 2020, arXiv: 2006.04418v4.
- [Lo99] Lohmiller, W.: Contraction Analysis of Nonlinear Systems, Ph.D. Thesis, Massachusetts Institute of Technology, 1999.
- [LS17] Liang, S.; Srikant, R.: Why Deep Neural Networks for Function Approximation?/, 2017, arXiv: 1610.04161 [cs.LG].
- [LS97] Lohmiller, W.; Slotine, J.-J. E.: Applications of Contraction Analysis. *Conference on Decision and Control* 36/, 1997.

- [LZD17] Lin, Y.; Zhao, H.; Ding, H.: Posture optimization methodology of 6R industrial robots for machining using performance evaluation indexes. *Robotics and Computer-Integrated Manufacturing* 48/April 2016, pp. 59–72, 2017, ISSN: 07365845.
- [Ma12] Malzahn, J.: *Robotics Toolbox*, RST, Technische Universität Dortmund, 2012.
- [Ma15] Martin Abadi; Ashish Agarwal; Paul Barham; Eugene Brevdo; Zhifeng Chen; Craig Citro; Greg S. Corrado; Andy Davis; Jeffrey Dean; Matthieu Devin; Sanjay Ghemawat; Ian Goodfellow; Andrew Harp; Geoffrey Irving; Michael Isard; Yangqing Jia; Rafal Jozefowicz; Lukasz Kaiser; Manjunath Kudlur; Josh Levenberg; Dandelion Mané; Rajat Monga; Sherry Moore; Derek Murray; Chris Olah; Mike Schuster; Jonathon Shlens; Benoit Steiner; Ilya Sutskever; Kunal Talwar; Paul Tucker; Vincent Vanhoucke; Vijay Vasudevan; Fernanda Viégas; Oriol Vinyals; Pete Warden; Martin Wattenberg; Martin Wicke; Yuan Yu; Xiaoqiang Zheng: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015, URL: <https://www.tensorflow.org/>.
- [Ma19] MathWorks, I.: *Symbolic Math Toolbox*, 2019, URL: <https://www.mathworks.com/help/symbolic/>.
- [MA21] MATLAB: *MATLAB (R2021a)*, Natick, Massachusetts, 2021.
- [MAS03] Meireles, M. R.; Almeida, P. E.; Simões, M. G.: A comprehensive review for industrial applicability of artificial neural networks. *IEEE transactions on industrial electronics* 50/3, pp. 585–601, 2003.
- [Me20] Mesmer, P.; Neubauer, M.; Lechler, A.; Verl, A.: Drive-Based Vibration Damping Control for Robot Machining. *IEEE Robotics and Automation Letters* 5/2, pp. 564–571, 2020, ISSN: 23773766.
- [MFP20] Mavkov, B.; Forgiione, M.; Piga, D.: Integrated Neural Networks for Nonlinear Continuous-Time System Identification. *IEEE Control Systems Letters*, p. 1, 2020.
- [MH08] Moberg, S.; Hanssen, S.: On Feedback Linearization for Robust Tracking Control of Flexible Joint Robots. 41/2, pp. 12218–12223, 2008, ISSN: 14746670.
- [Ne01] Nelles, O.: *Nonlinear system identification*. Springer Verlag, Heidelberg, Germany, 2001.
- [NH10] Nair, V.; Hinton, G. E.: Rectified linear units improve restricted boltzmann machines./, pp. 807–814, 2010.

- [NP90] Narendra, K.; Parthasarathy, K.: Identification and Control of Dynamical Systems Using Neural Networks. *IEEE transactions on neural networks* 1/1, pp. 4–27, 1990, ISSN: 1045-9227.
- [NSP08] Nguyen-Tuong, D.; Seeger, M.; Peters, J.: Computed torque control with non-parametric regression models. In: 2008 American Control Conference. IEEE, pp. 212–217, 2008.
- [Og10] Ogasawara, E.; Martinez, L.; de Oliveira, D.; Zimbrão, G.; Pappa, G.; Matoso, M.: Adaptive Normalization: A Novel Data Normalization Approach for Non-Stationary Time Series. *IJCNN/*, 2010.
- [Og16] Ogunmolu, O.; Gu, X.; Jiang, S.; Gans, N.: Nonlinear Systems Identification Using Deep Dynamic Neural Networks./, 2016, arXiv: 1610.01439 [cs.NE].
- [Ol12] Olabi, A.; Damak, M.; Bearee, R.; Gibaru, O.; Leleu, S.: Improving the accuracy of industrial robots by offline compensation of joints errors. 2012 IEEE International Conference on Industrial Technology, ICIT 2012, Proceedings/, pp. 492–497, 2012.
- [OR20] Onken, D.; Ruthotto, L.: Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and Continuous Normalizing Flows, 2020, arXiv: 2005.13420v2.
- [Pa19] Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in neural information processing systems*/32, pp. 8024–8035, 2019, URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [PKC11] Pashkevich, A.; Klimchik, A.; Chablat, D.: Enhanced stiffness modeling of manipulators with passive joints. *Mechanism and Machine Theory* 46/5, pp. 662–679, 2011, ISSN: 0094114X.
- [Pr98] Prechelt, L.: Early Stopping - But When? In (Orr, G. B.; Müller, K.-R., eds.): *Neural Networks: Tricks of the Trade*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 55–69, 1998, ISBN: 978-3-540-49430-0, URL: https://doi.org/10.1007/3-540-49430-8_3.
- [RCD19] Rubanova, Y.; Chen, R. T.; Duvenaud, D. K.: Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems* 32/, 2019.

- [Re17] Relan, R.; Tiels, K.; Marconato, A.; Schoukens, J.: An Unstructured Flexible Nonlinear Model for the Cascaded Water-tanks Benchmark. *IFAC-PapersOnLine* 50/1, pp. 452–457, 2017, ISSN: 2405-8963.
- [RH20] Ruthotto, L.; Haber, E.: Deep Neural Networks Motivated by Partial Differential Equations. *Journal of Mathematical Imaging and Vision* 62/3, pp. 352–364, 2020, ISSN: 0924-9907.
- [RHB09] Ruderman, M.; Hoffmann, F.; Bertram, T.: Modeling and Identification of Elastic Robot Joints With Hysteresis and Backlash. *IEEE Transactions on Industrial Electronics* 56/10, pp. 3840–3847, 2009, ISSN: 0278-0046.
- [Sc14] Schneider, U.; Momeni-K, M.; Ansaloni, M.; Verl, A.: Stiffness modeling of industrial robots for deformation compensation in machining. *IEEE International Conference on Intelligent Robots and Systems/Iros*, pp. 4464–4469, 2014, ISSN: 21530866.
- [SC15] Sergey Ioffe; Christian Szegedy: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *International conference on machine learning/*, 2015.
- [Sc16a] Schneider, U.; Drust, M.; Ansaloni, M.; Lehmann, C.; Pellicciari, M.; Leali, F.; Gunnink, J. W.; Verl, A.: Improving robotic machining accuracy through experimental error investigation and modular compensation. *The International Journal of Advanced Manufacturing Technology* 85/, pp. 3–15, 2016.
- [Sc16b] Schoukens, M.; Mattsson, P.; Wigren, T.; Noel, J. P.: Cascaded tanks benchmark combining soft and hard nonlinearities. *Workshop on Nonlinear System Identification Benchmarks, Brussels, Belgium/*, 2016.
- [Sc22] Schüssler, M.: Machine learning with nonlinear state space models. *at-Automatisierungstechnik* 70/11, pp. 1027–1028, 2022.
- [Sh20] Sherstinsky, A.: Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. *Physica D: Nonlinear Phenomena* 404/8, p. 132306, 2020, ISSN: 0167-2789, arXiv: 1808.03314v9.
- [SI06] SIEMENS AG, München, Germany: *Projektierungshandbuch Ausgabe 12/2006, Synchronmotoren 1FK7, SINAMICS S120*, <https://support.industry.siemens.com/cs/document/28683106/sinamics-s120-synchronmotoren-1fk7-?dti=0&lc=de-WW>, 2006.
- [Sj95] Sjöberg, J.; Zhang, Q.; Ljung, L.; Benveniste, A.; Deylon, B.; Glorenec, P.-Y.; Hjalmarsson, H.; Juditsky, A.: *Nonlinear Black-Box Modeling in System Identification: a Unified Overview./*, 1995.

- [SL19] Schoukens, J.; Ljung, L.: Nonlinear System Identification: A User-Oriented Roadmap. *IEEE Control Systems Magazine*/39, pp. 28–99, 2019, URL: <http://arxiv.org/pdf/1902.00683v1>.
- [Sm20] Smyl, S.: A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting* 36/1, pp. 75–85, 2020, ISSN: 01692070.
- [So08] Sontag, E. D.: Input to State Stability: Basic Concepts and Results. *Nonlinear and optimal control theory: lectures given at the CIME/1932:163–220*, 2008.
- [Sp87] Spong, M. W.: Modeling and control of elastic joint robots. *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME* 109/4, pp. 310–319, 1987, ISSN: 15289028.
- [SP99] Sanchez, E. N.; Perez, J. P.: Input-to-state stability (ISS) analysis for dynamic neural networks. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 46/11, pp. 1395–1398, 1999, ISSN: 1057-7122.
- [Sr14] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15/1, pp. 1929–1958, 2014.
- [SS17] Svensson, A.; Schön, T. B.: A flexible state–space model for learning nonlinear dynamical systems. *Automatica* 80/, pp. 189–199, 2017, ISSN: 00051098.
- [SS97] Sola, J.; Sevilla, J.: Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science* 44/3, pp. 1464–1468, 1997, ISSN: 0018-9499.
- [ST17] Schoukens, M.; Tiels, K.: Identification of block-oriented nonlinear systems starting from linear approximations: A survey. *Automatica* 85/, pp. 272–292, 2017.
- [ST20] Schoukens, M.; Tóth, R.: On the Initialization of Nonlinear LFR Model Identification with the Best Linear Approximation, 21st IFAC World Congress, 2020, URL: <https://www.sciencedirect.com/science/article/pii/S2405896320304006>.
- [ST98] Scarselli, F.; Tsoi, A. C.: Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks* 11/1, pp. 15–37, 1998.
- [SVS07] Swevers, J.; Verdonck, W.; de Schutter, J.: Dynamic Model Identification for Industrial Robots. *IEEE Control Systems* 27/5, pp. 58–71, 2007, ISSN: 1066-033X.

- [TB97] Tsoi, A. C.; Back, A. D.: Discrete time recurrent neural network architectures: A unifying review. *Neurocomputing* 15/3-4, pp. 183–223, 1997.
- [TPA16] Tran, T. L.; Pham, A. D.; Ahn, H.-J.: Lost motion analysis of one stage cycloid reducer considering tolerances. *International Journal of Precision Engineering and Manufacturing* 17/8, pp. 1009–1016, Aug. 2016, ISSN: 2234-7593.
- [TUB20] Tika, A.; Ulmen, J.; Bajcinca, N.: Dynamic Parameter Estimation Utilizing Optimized Trajectories. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 7300–7307, 2020.
- [Uç19] Uçak, K.: A Runge–Kutta neural network-based control method for nonlinear MIMO systems. *Soft Computing* 23/17, pp. 7769–7803, 2019, ISSN: 1432-7643.
- [Uç20] Uçak, K.: A Novel Model Predictive Runge–Kutta Neural Network Controller for Nonlinear MIMO Systems. *Neural Processing Letters* 51/2, pp. 1789–1833, 2020, ISSN: 1370-4621.
- [VD94] Van Overschee, P.; De Moor, B.: N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems. *Automatica* 30/1, Special issue on statistical signal processing and control, pp. 75–93, 1994, ISSN: 0005-1098, URL: <https://www.sciencedirect.com/science/article/pii/0005109894902305>.
- [Ve15] Vemula, B. R.: Evaluation of robot structures: For applications that require high performance, safety and low energy consumption, PhD thesis, Mälardalen University, 2015.
- [VKL17] Vieler, H.; Karim, A.; Lechler, A.: Drive based damping for robots with secondary encoders. *Robotics and Computer-Integrated Manufacturing* 47/February, pp. 117–122, 2017, ISSN: 07365845.
- [We90] Werbos, P. J.: Backpropagation Through Time: What It Does and How to Do It. *Proceedings of the IEEE* 78/10, pp. 1550–1560, 1990.
- [WKK18] Wu, K.; Krewet, C.; Kuhlenkötter, B.: Dynamic performance of industrial robot in corner path with CNC controller. *Robotics and Computer-Integrated Manufacturing* 54/, pp. 156–161, Dec. 2018, ISSN: 07365845, URL: <https://linkinghub.elsevier.com/retrieve/pii/S0736584516303775>.
- [WKS18] Wensing, P. M.; Kim, S.; Slotine, J.-J. E.: Linear Matrix Inequalities for Physically Consistent Inertial Parameter Identification: A Statistical Perspective on the Mass Distribution. *IEEE Robotics and Automation Letters* 3/1, pp. 60–67, 2018.

- [WL92] Wang, W. S.; Liu, C. H.: Controller Design and Implementation for Industrial Robots with Flexible Joints. *IEEE Transactions on Industrial Electronics* 39/5, pp. 379–391, 1992, ISSN: 15579948.
- [WL98] Wang, Y.-J.; Lin, C.-T.: Runge-Kutta Neural Network For Identification Of Dynamical Systems In High Accuracy. *IEEE transactions on neural networks*/9(2):294–307, 1998, ISSN: 1045-9227.
- [WWY10] Wu, J.; Wang, J.; You, Z.: An overview of dynamic parameter identification of robots. *Robotics and Computer-Integrated Manufacturing* 26/5, pp. 414–419, 2010, ISSN: 07365845.
- [WX06] Wang, L.; Xu, Z.: Sufficient and necessary conditions for global exponential stability of discrete-time recurrent neural networks. *Circuits and Systems I: Regular Papers, IEEE Transactions on* 53/6, pp. 1373–1380, 2006.
- [WZ89] Williams, R. J.; Zipser, D.: A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*/1(2):270–280, 1989.
- [WZF09] Wang, J.; Zhang, H.; Fuhlbrigge, T.: Improving machining accuracy with robot deformation compensation. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*/, pp. 3826–3831, 2009.
- [XJ02] Xiong, Q.; Jutan, A.: Grey-box modelling and control of chemical processes. *Chemical Engineering Science*/, 2002.
- [Ya16] Yang, T.; Yan, S.; Ma, W.; Han, Z.: Joint dynamic analysis of space manipulator with planetary gear train transmission. *Robotica* 34/5, pp. 1042–1058, May 2016, ISSN: 0263-5747.
- [YP18] Yin, X.; Pan, L.: Enhancing trajectory tracking accuracy for industrial robot with robust adaptive control. *Robotics and Computer-Integrated Manufacturing* 51/, pp. 97–102, June 2018, ISSN: 07365845.
- [Yu04] Yu, W.: Nonlinear system identification using discrete-time recurrent neural networks with stable learning algorithms. *Information sciences* 158/, pp. 131–147, 2004.
- [Yu18] Yuan, L.; Pan, Z.; Ding, D.; Sun, S.; Li, W.: A Review on Chatter in Robotic Machining Process Regarding Both Regenerative and Mode Coupling Mechanism. *IEEE/ASME Transactions on Mechatronics* 23/5, pp. 2240–2251, 2018, ISSN: 10834435.
- [YYH15] Yang, T.; Yan, S.; Han, Z.: Nonlinear model of space manipulator joint considering time-variant stiffness and backlash. *Journal of Sound and Vibration* 341/, pp. 246–259, 2015, ISSN: 10958568.

- [Zh20] Zhang, B.; Wu, J.; Wang, L.; Yu, Z.: Accurate dynamic modeling and control parameters design of an industrial hybrid spray-painting robot. *Robotics and Computer-Integrated Manufacturing* 63/, p. 101923, June 2020, ISSN: 07365845, URL: <https://linkinghub.elsevier.com/retrieve/pii/S0736584519300043>.

Own Publications

- [WDR21] Weigand, J.; Deflorian, M.; Ruskowski, M.: Input-to-state stability for system identification with continuous-time Runge–Kutta neural networks. *International Journal of Control*/, pp. 1–17, 2021, ISSN: 0020-7179.
- [We16] Weigand, J.: Integration of Physically-Motivated Dynamical Models into Iterative Learning Controllers: Bachelorthesis, RWTH Aachen University, Institute for Control, supervised by Sebastian Stemmler, 2016.
- [We17] Weigand, J.: Systemidentification of a CNC-Milling Center Using Artificial Neural Networks: Masterthesis, RWTH Aachen University, Institute for Control, supervised by Sebastian Stemmler, 2017.
- [We21] Weigand, J.; Raible, J.; Zantopp, N.; Demir, O.; Trachte, A.; Wagner, A.; Ruskowski, M.: Hybrid Data-Driven Modelling for Inverse Control of Hydraulic Excavators. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 2127–2134, 2021, ISBN: 978-1-6654-1714-3.
- [We22] Weigand, J.; Götz, J.; Ulmen, J.; Ruskowski, M.: Dataset and Baseline for an Industrial Robot Identification Benchmark. *Workshop on Nonlinear System Identification Benchmarks*/, 2022.
- [We23] Weigand, J.; Beintema, G.; Ulmen, J.; Görges, D.; Tóth, R.; Schoukens, M.; Ruskowski, M.: State Derivative Normalization for Continuous-Time Neural Networks. In: Publication unter review. 2023.
- [WGR20] Weigand, J.; Gafur, N.; Ruskowski, M.: Flatness Based Control of an Industrial Robot Joint Using Secondary Encoders. *Robotics and Computer-Integrated Manufacturing*. 68/, p. 102039, 2020, ISSN: 07365845.
- [WVR20a] Weigand, J.; Volkmann, M.; Ruskowski, M.: Neural Adaptive Control of a Robot Joint Using Secondary Encoders. In (Berns, K.; Görges, D., eds.): *Advances in Service and Industrial Robotics*. Vol. 980, *Advances in Intelligent Systems and Computing*, Springer International Publishing, Cham, pp. 153–161, 2020, ISBN: 978-3-030-19647-9.
- [WVR20b] Weigand, J.; Volkmann, M.; Ruskowski, M.: Roboter in der autonomen Produktion der Zukunft. *Sammelband Smart Factory*/, 2020.

Supervised Student Thesis

- [As21] Asao, T.: Development of a Mobile Robot with Robot Operating System (ROS), Studienarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2021.
- [As22] Asao, T.: Development and Implementation of an Exploration Algorithm for a Mobile Robot, Masterthesis, On-Going, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, Leonhard Kuntz, 2022.
- [Be18] Belz, S.: Nonlinear Modeling and Flatness-based Control of an Robot Joint, Studienarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2018.
- [Be19] Belz, S.: Development of Compliant Robot External Structures for the Absorption of Collision Energy During Human-Robot-Collaboration, Masterthesis, Chair of Machine Tools and Control Systems, TU Kaiserslautern, carried out externally at KUKA AG, supervised by Jonas Weigand, Markus Wünsch, 2019.
- [Bl21] Blumhofer, B.: Development of a Visual Inertial Simultaneous Localization and Mapping based Navigation for a Mobile Robot, Projektarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2021.
- [Bl22] Blumhofer, B.: Diagnosis of Rare Fault Cases for a Mobile Robot, Masterthesis, On-Going, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, Leonhard Kuntz, 2022.
- [Di19] Diederichs, B.: Design of an Adaptive Cascade Control System for a KUKA Industrial Robot, Studienarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2019.
- [Ga20] Gassen, E.: Configuration and Operation of a UR3 Robot with Robot Operating System ROS, Projektarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2020.
- [Ge18] Gertje, S.: Nonlinear Optimization Algorithms for a Model Predictive Control of a KUKA Industrial Robot, Masterthesis, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2018.
- [Gö19] Götz, J.: Conception and Development of a Control Application with User Interface for an Industrial Robot, Projektarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2019.

- [Gö20] Götz, J.: Realization of a Robot Control with Modular Hardware and Self-Developed Software for an Industrial Robot Mechanics, Masterthesis, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2020.
- [Ha18a] Harttig, S.: Modeling Dynamics of the KUKA Quantec Industrial Robot for Use in the Milling Process, Masterthesis, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2018.
- [Ha22a] Hagen, N.: Development of a Robot Controller with OPCUA Data Transfer, Masterthesis, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2022.
- [He19] Hensel Leon; Wendling, F.: Influence of Dead Time on the Control Quality of an Industrial Robot Drive, Studienprojekt, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2019.
- [Kl18] Kleber, T.: Human-Robot-Collaboration: Safety and Challenges for Machining Applications, Studienarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, Indujan Sivanesarajah, 2018.
- [Ku20] Kuntz, L.: Swinging-Up of an Inverted Pendulum using Reinforcement Learning, Bachelorthesis, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2020.
- [La19a] Lafuente Larrañaga, J. S.: Design and Development of a Digitally Controlled Magnetic Levitation System, Masterthesis, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, Indujan Sivanesarajah, 2019.
- [La19b] Lamoth, S.: Realization of an Industrial Drive Train on an Inverted Pendulum, Projektarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2019.
- [Le19] Leffler, C.: Control for Active Vibration Damping for Milling Machines, Projektarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, Indujan Sivanesarajah, 2019.
- [Lu19] Lump, J.: Miniaturization of a Grinding Arm for Machining Internal Gears, Bachelorthesis, Chair of Machine Tools and Control Systems, TU Kaiserslautern, carried out externally at KLINGELNBERG AG, supervised by Jonas Weigand, Alois Mundt, Leif Heckes, 2019.

- [Mi19] Mirgel, J.: Conception and Realization of an Inverted Pendulum, Bachelorthesis, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2019.
- [Mi21] Mirgel, J.: Development of a Test System for the Determination of Technological Limits of Tactile Robot-based Assembly, Masterthesis, Chair of Machine Tools and Control Systems, TU Kaiserslautern, carried out externally at AUDI AG, supervised by Jonas Weigand, Thomas Schraml, Wilhelm Hacker, 2021.
- [Ra19] Raible, J.: Development of Stability Criteria for Neural Adaptive Controllers, Studienarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2019.
- [Ra20] Raible, J.: Development of Data-based Control for Assistance and Automation Functions for an Hydraulic Excavator, Diplomarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, carried out externally at Robert Bosch GmbH, supervised by Jonas Weigand, Ozan Demir, 2020.
- [St20] Steinmetz, F.: Spatio Temporal Object Detection in Videos, Projektarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2020.
- [Wa19] Wang, X.: Parameter Identification of a Single Joint of a KUKA Industrial Robot, Projektarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2019.
- [Za18] Zantopp, N.: Using Runge-Kutta Neural Networks for Systemidentification of a KUKA Industrial Robot, Studienarbeit, Chair of Machine Tools and Control Systems, TU Kaiserslautern, supervised by Jonas Weigand, 2018.

Curriculum Vitae

Personal Information

Name: Jonas Benjamin Weigand
Nationality: German

Education

11.2017 - 10.2023 Doctoral student, Chair of Machine Tools and Control Systems, RPTU Kaiserslautern-Landau
09.2011 - 10.2017 Mechanical Engineering Studies, RWTH Aachen University
07.2008 - 10.2008 Glenfield College, New Zealand, School Exchange
08.2002 - 03.2011 Leibniz Gymnasium, Neustadt a. d. Weinstraße
Advanced Level Subjects: Physics, Math and English

Work Experience

since 09.2022 Senior Data Scientist, Siemens AG, Karlsruhe
01.2021 - 08.2022 Researcher, German Research Center for Artificial Intelligence (DFKI), Kaiserslautern
11.2017 - 08.2022 Researcher, Chair of Machine Tools and Control Systems, RPTU Kaiserslautern-Landau
03.2016 - 03.2017 Student assistant at the Institute of Automatic Control (IRT), RWTH Aachen
03.2015 - 10.2015 ZF Friedrichshafen, internship
02.2014 - 06.2014 Teaching assistant at German University of Technology, Muscat, Oman
10.2013 - 03.2014 Student assistant at the Institute of General Mechanics, RWTH Aachen
05.2011 - 07.2011 RONAL, Landau, internship