Deep Lagrangian Networks for Robot Control

Deep Lagrangian Networks für Roboterregelung
Bachelor-Thesis von Jasper Süß aus Langen
Oktober 2019

TECHNISCHE UNIVERSITÄT DARMSTADT

Deep Lagrangian Networks for Robot Control
Deep Lagrangian Networks für Roboterregelung

Vorgelegte Bachelor-Thesis von Jasper Süß aus Langen

1. Gutachten: Prof. Jan Peters
2. Gutachten: M.Sc. Michael Lutter

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Jasper Süß, die vorliegende Bachelor-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

In the submitted thesis the written copies and the electronic version are identical in content.

Datum / Date: ______________________ Unterschrift / Signature: ______________________

Abstract

A dynamics model is essential for the control of a robot. It can encapsulate information about different properties of the robotic system like masses, joint configuration, friction, the center of gravity and inertia parameters in a mathematical model. Online model learning is a viable alternative for robotic manipulators in comparison to analytical models and models learned offline, as online learning offers the advantage of being able to dynamically react to a changing model during execution. During an exploration phase, the model gets learned by an online algorithm using a constant stream of data generated by the robot performing a desired trajectory. In this phase, it is important that the robot is robustly controlled, while simultaneously generating rich data to learn on, ideally using a trajectory providing maximal excitation of the parameters for the dynamics model.
In this thesis, I will lay out the foundations of online model learning, develop a robust controller for the Barrett WAM 4DoF and will evaluate the performance of different trajectories that have been used for online model learning on the WAM 4DoF robot. Additionally, I will show that the analytical model of the WAM provided by Barrett is capturing the real dynamics only badly and that trajectories, which bear the excitation of dynamic parameters in mind, indeed perform superior to other, more trivial trajectories.

Zusammenfassung

Ein Dynamikmodell eines Roboters ist essentiell um diesen zu steuern. Es kann Informationen über den Roboter wie Masses, Konfiguration der Gelenke, Reibung, Schwerpunkte und Trägheitsparameter in einem mathematischen Modell vereinen. Das Online-Lernen von Modellen ist mittlerweile eine mögliche Alternative im Vergleich zu analytischen oder offline gelernten Modellen. Der Vorteil von online gelernten Modellen besteht darin, dass während der Ausführung dynamisch auf eine sich verändernde Umgebung, die damit das Dynamikmodell beeinflusst, reagiert werden kann. Während einer Explorationsphase wird das Modell von einem Online-Algorithmus gelernt, der dafür mit einem konstanten Strom von Daten versorgt wird, die von dem Roboter erzeugt werden, der eine gewünschte Trajektorie verfolgt. Während dieser Phase ist es wichtig, dass der Roboter robust geregelt wird und dass gleichzeitig die erzeugten Daten reich an Informationen sind, idealerweise indem eine Trajektorie verwendet wird, die die Parameter des Dynamikmodells maximal anregt.
In dieser Thesis werde ich die Grundlagen des Online-Lernens von Modellen darlegen, einen robusten Regler für den Barret WAM 4DoF entwickeln und werde bewerten, was für einen Einfluss verschiedene, für das Online-Lernen verwendete, Trajektorien auf die Leistung des Online-Lernens am WAM 4DoF Roboter haben. Außerdem werde ich zeigen, dass das analytische Modell des WAM das von Barrett zur Verfügung gestellt wird, die tatsächlichen Dynamiken nur unzulänglich darstellt und dass Trajektorien, die designt wurden, um die dynamischen Parameter anzuregen, tatsächlich besser abschneiden, als andere einfachere Trajektorien.

Acknowledgments

Contents

Figures and Tables

List of Figures

List of Tables

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
CTC	Computed Torque Control
DeLaN	Deep Lagrangian Networks
DoF	Degrees of Freedom
FFS	Finite Fourier Series
RBDM	Rigid Body Dynamics Model
SMC	Sliding Mode Control
URDF	Unified Robot Description Format
WAM	Whole Arm Manipulator

List of Symbols

Notation	Description
$\mathbf{q}$	vector of joint positions
$\dot{\mathbf{q}}$	vector of joint velocities
$\ddot{\mathbf{q}}$	vector of joint accelerations
$\mathbf{q}_\mathbf{d}$	vector of desired joint position
$\dot{\mathbf{q}}_\mathbf{d}$	vector of desired joint velocity
$\ddot{\mathbf{q}}_\mathbf{d}$	vector of desired joint acceleration
$\tilde{\mathbf{q}}$	vector of error between $\mathbf{q}$ and $\mathbf{q}_\mathbf{d}$
$\dot{\tilde{\mathbf{q}}}$	vector of error between $\dot{\mathbf{q}}$ and $\dot{\mathbf{q}}_\mathbf{d}$
$\ddot{\tilde{\mathbf{q}}}$	vector of error between $\ddot{\mathbf{q}}$ and $\ddot{\mathbf{q}}_\mathbf{d}$
$\ddot{\tilde{\mathbf{q}}}$	reference acceleration vector, i.e. $\ddot{\mathbf{q}}_\mathrm{d} - \lambda \dot{\tilde{\mathbf{q}}}$
$\dot{\tilde{\mathbf{q}}}$	reference velocity vector, i.e. $\dot{\mathbf{q}}_\mathrm{d} - \lambda \tilde{\mathbf{q}}$

$\boldsymbol{\tau}$	vector of joint torques
$\mathbf{H(q)}$	Inertia matrix
$\mathbf{c(q, \dot{q})}$	vector of coriolis and cetripetal forces
$\mathbf{g(q)}$	vector of gravitation
$\mathbf{K}$	Diagonal Matrix with control gains
$\mathbf{s}$	sliding

List of Operators

Notation	Description	Operator
V	Lyapunov function	$V(\bullet)$

1 Introduction

1.1 Motivation

Robotic manipulators have been developed for more than 60 years now, with the first automata already appearing several hundred years earlier. Those automata were either only plans, for example by Leonarda da Vinci in the 16th-century [3], appeared in fiction, as in form of the humanoid automata "Olimpia" in E.T.A. Hoffmanns novel "The Sandman" [4] or as a practical application, for example the first programmable loom in 1804. The development of transistors and integrated circuits after the second world war then resulted in rapid progress in the fields of computer science, mechanical engineering and in the intersection of both, robotics. In 1960 the company Unimation presented the first robot for industrial use [5], in 1974 KUKA presented the Famulus, the first 6-DoF industrial robot with electromechanically driven axes [6]. Since then robotic manipulators have established themselves as an essential workforce to automate industrial production, most prominently in the production of cars. The typical workflow of an industrial robotic manipulator is static. It has one preprogrammed task that it repeatedly executes using a set trajectory. Such behavior is sufficient for a static environment like it is found inside a factory. However, requirements for robots in other areas are more complex and challenging [7]

The use of robots expanded to many other areas, simplifying many tasks in our daily lives. Nowadays there exist vacuum cleaner and lawnmower robots [8], autonomous cars [9], delivery drones for packages [10] or robots that can assist humans, for example in elder care [11]. Their appearance can vary greatly: Walking [12], flying [10], driving [9] or tracked robots [13], humanoid [12] or nature-inspired [14] robots, extremely small [15] or big robots [16].

Previously it was often important to execute one specific task thousands of times. However, for other robotic systems it is also important that they are able to solve several different tasks only a few times, e.g. a robot working in close interaction with humans. For example, a robot assisting with elderly care needs to react to the actions of its ward accordingly in a dynamically changing environment.

Each of those different robots still shares one trait: Each of them can be described by a dynamical model, which encapsulates information about the properties of the robot, like masses or inertia of the matrix. The model of a robot can be used to calculate movements, e.g. how much force a motor needs to exert so that the robot follows a predefined trajectory to a desired position.

With the change in appearance and usage of robotic manipulators, other areas of research also evolved. Greater computational power and more available data to process also propelled the area of machine learning in the last decades. This lead to an interweaving of machine learning methods in the development of robots.

The task of finding a good model for a robot or any other dynamical system is known as model learning. Machine learning techniques can be used to help acquire such a model. Typically, data is fed into the machine learning algorithm which then generates a model fitting to the input data. This is known as offline learning since the model is generated without using the robot itself. However it is also possible to perform online learning. Here the robot performs a task while the model is learned, i.e. the robot creates a constant stream of data using a model, which gets fed to the model learning algorithm, which in turn feeds a constant stream of updates of the used model to the robot.

This technique has some advantages over classical offline learning, but also drawbacks. The most interesting aspect of online learning is the possibility for the robot to adapt to a dynamically changing environment, e.g. when it picks something up, where the mass is unknown.

However, the drawbacks include that at the start of learning it is necessary to generate some data which can be used for the first iteration of creating the model. Since no initial model is available, the robot needs an initial controller to generate the initial data. Once a model is learned it is still possible that the model deviates strongly from the actual model. The controller is then needed to compensate the torques generated by the model, should the robot deviate too much from its desired trajectory. Finally, it is also possible that other unforeseen circumstances happen during the model learning, which could lead to instability of the robot. Therefore the robot must be correctly controlled during the whole exploration phase of model learning. In this work, I am going to develop a robust controller for the cable-driven Barret WAM 4DoF. The controller will be able to guarantee the stability and safety of the robot during the whole exploration phase of model learning and demonstrate its usage on several excitation trajectories. As a model learning algorithm, I am going to use a system identification algorithm by Atkeson et al. [2], which finds the parameters for the dynamics model.

1.2 Structure of the Thesis

The following thesis is structured as follows: In chapter 2 I am going to introduce important concepts necessary to understand the importance of model learning and robust control. In chapter 3 I present the robust controller for the Barrett WAM 4DoF including a joint limit avoidance. In chapter 4 I show the experimental setup where I tested the controller on the Barrett WAM 4DoF using several excitation trajectories and compared the performance on an model learning algorithm. I also extensively test the joint limit avoidance and show the limits of the analytic model of the WAM. After that, I draw the conclusions and results of my work in chapter 5. In chapter 6 an outlook is presented, giving possible approaches for further research.

2 Foundations

2.1 Forward and Inverse Models

In a robotics system, the state of the system typically consists of the joint position vector $\mathbf{q}$ and its derivatives, velocity $\dot{\mathbf{q}}$ and acceleration $\ddot{\mathbf{q}}$. The action needed to change the state of the system is the torque vector $\boldsymbol{\tau}$ that is acting on the joints. A dynamics model can describe the relationship between the state of the system and a chosen action, either how a chosen action affects the next state or which action needs to be taken to reach a desired state. In order to predict the next state of the system, given the current state and an action, a forward mdoel is used. An inverse model is required to predict the action needed to reach a desired state from the current state. Thus, for the forward dynamics problem, given the current joint positions, velocities, and accelerations as well as the torque, the acceleration of the next time stamp needs to be computed. To solve this problem it is only required to calculate the acceleration, as velocity and position can be calculated based on the acceleration and the former state. For the inverse dynamics problem, the current state is also given, as well as the next desired state, both consisting of position, velocity, and acceleration. Here, it is necessary to compute the required torque to reach the desired state at the next time stamp. It is to note that there always exists a solution to the forward dynamics problem, but not necessarily for the inverse dynamics. Given an action and a state, there will always be a state that the action results in. However, there is not always an action to get from one state to another. Therefore the inverse dynamics are typically a harder problem. The forward and inverse dynamics problem can both be written as a function

$$f(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}) = \ddot{\mathbf{q}}, \qquad f^{-1}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \boldsymbol{\tau}.$$

Once the inverse dynamics problem is exactly solved and f^{-1} is known it is possible to let the robot track arbitrary trajectories, which are sequences of desired states in its workspace. It does this by simply feeding the known information about current position, velocity and acceleration and the next desired position, velocity and acceleration into f^{-1} and applying the resulting torques to the joints of the robot.

One common approach to model the dynamics of a robotic manipulator is the Rigid Body Dynamics Model (RBDM) [17], where the torques are dependent on a function consisting of the inertia matrix $\mathbf{H(q)}$, the coriolis and centripetal forces $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}}$ and the gravitational $\mathbf{g(q)}$ vector

$$f(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})^{-1} = \mathbf{H}(\mathbf{q}) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}.$$

The RBDM assumes rigid dynamics, i.e. that the used body cannot be deformed, therefore simplifying the model. This however, does not capture the reality of robotic manipulators, as every robot has some parts which can be deformed. Therefore the dynamics of a real system cannot be perfectly modeled by the RBDM. Often another term is added to model all the forces not covered by the RBDM, for example, friction, elasticities, couplings or sensor noise [18]

$$\mathbf{H}(\mathbf{q}) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) + \epsilon(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \boldsymbol{\tau}.$$

Another formulation of the RBDM is that of a linear model [19]

$$\Phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\Theta = \boldsymbol{\tau},$$

where Θ describes the vector of barycentric parameters for the robot and $\Phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is the identification matrix.
The Rigid Body Dynamics Model offers simplicity and efficiency [20], which is both an advantage and a disadvantage. Due to its simplicity, it makes it easier to find the few important parameters required to produce good results. However, this model will never be entirely sufficient for the real dynamics of a system. It does not exactly model friction, elasticity or sensor noise and can therefore not capture the real dynamics.

Another possibility of modeling is to use nonparametric models, where no static structure of the function is given. This allows for more flexible models, as there are no assumptions made of the exact structure of the model. However, it also reduces interpretability of the model, as the results cannot be mapped to certain parameters of the robot.

2.2 Model Learning and System Identification

A robot model is essential for the control of a robot. It can encapsulate information about different properties of the robotic system like masses, joint configuration, friction, the center of gravity and inertia parameters in a mathematical model [21], describing the kinematics or dynamics of the robot body. Those models can be analytically calculated by exactly measuring and weighing every single part of the robot, which usually requires complete decomposition of the robotic manipulator. This method requires time, effort and monetary resources and is not realistic for complex modern robots, e.g. humanoid robots [22]. Also, small details like the exact positioning of cables can never be exactly measured. Alternatively, CAD programs may calculate parameters for the dynamics model, which are, however, often not complete [23]. Additionally, all those analytical models are static and cannot react to external influences like an added mass at the end effector, a changed tool at the end effector, partly damaged parts of the robot, unknown nonlinearities like friction or a dynamically changing environment. Therefore, it is not favorable to use such a method unless the design of the robot is sufficiently simplistic and the work environment of the robot is exactly known and static.

Model learning [24] and system identification offer an alternative to the analytical model. They use statistical methods to find a model fitting to given data, which is strongly tied to the field of machine learning. Intuitively, one may think that using model learning may perform worse than analytical models because measuring a robot should result in a more exact model than finding some relation between data. However, most of the time model learning algorithms outperform the analytical models, often even the provided model of robot manufacturers [25]. Additionally, both analytical and learned models suffer from a model mismatch, which denotes the difference between an estimated model and the actual model. The real-world dynamics can never be exactly captured, simply because too many uncertainties exist. In our example this could be the temperature or the exact positioning of cables, which impacts inertia parameters of a cable very slightly. Such a model mismatch is however not too severe, since the sensors of the robot are always noisy, never resulting in exact measurement needed for exact control of the robotic manipulator.
Model learning has been successfully used for different applications in robotics, e.g. for learning a forward dynamics model [1], an inverse dynamics model [26] or learning an inverse kinematics model [27]. Both model learning and analytical models can be used to e.g. control a dynamic system like a robotic manipulator, but only model learning allows for estimating a model directly from the data collected on the real robot [24].
The authors of [28] state three different components for a model-based adaptive control system which can be transferred to the problem of online model learning: Modeling, exploration and control policy design. The first part describes the choice of an explicit model for a system and the algorithm that learns that model. In the context of online model learning, this could be for example using a nonparametric regression method like Local Gaussian Process Regression (LGP) [26] or use of the Rigid Body Dynamics Model (RBDM) (see 2.1) in Deep Lagrangian Networks (DeLaN) [1]. Exploration describes how the data used for learning is generated, which for robotic manipulators translates to the choice of the used trajectory during learning. The control policy part is about how the system is controlled during learning and how exactly the model is used for finding the next action. Most of the time the focus in research lies on the modeling part, however, the latter two are also important factors for successful online model learning. In [29] the three main components for model learning are identified as the used data, the set of possible models and the identification method, i.e. once again the data is an important factor for successful model learning.
It is important to note that the goal of model learning is to find a good model which can then be analyzed, used for other tasks or further used for the current task, e.g. tracking control. The goal is explicitly not to achieve perfect tracking performance during the exploration phase and not to learn a control policy that is able to achieve this tracking performance. This approach is related to the field of online adaptive control, where perturbations of the system are counteracted by a learned control policy to improve tracking performance.
It is therefore of utmost importance that the data generated online used for model learning supports the learning process as best as possible, i.e. the relation between torque and state should be as clear as possible for the learning algorithm and well-conditioned. The performance of the above-mentioned modeling part can be enhanced by a good selection of the trajectories used for the exploration to generate rich data, as well as a controller design that is robust while simultaneously having low gains and only small chattering. Trajectories that excite lots of model parameters, therefore, increasing the models performance are called "persistently exciting trajectories" in the system identification literature [30]. One important aspect of the controller design in this context is the problem of chattering, which is that the torque oscillates around a value, while the state changes only slightly, resulting in badly conditioned data. Both of these problems should be paid attention to when doing online model learning.

2.3 Online and Offline Learning

There are two different ways to learn a model, abstracting from the actually used learning algorithm. On one hand, the model can be learned offline, where learning data is fed to the algorithm and a model is learned, which, once the learning

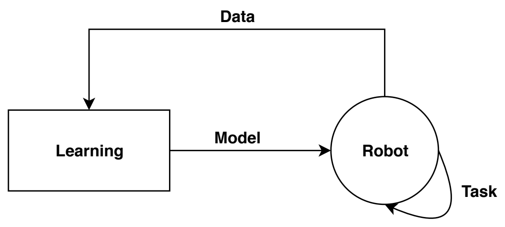


Figure 2.1.: Online Learning

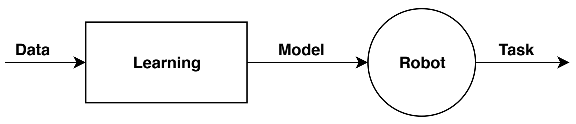


Figure 2.2.: Offline Learning

is finished, can be used for the robot to perform a task. On the other hand, the model can also be learned online, where the robot executes the desired task and while doing so collects data, which is used by the learning algorithm to constantly update the model. In turn, this model then gets used to control the robot and collect more data. The phase of online learning where the collected data is used to learn and update the model is called exploration. When learning offline, the model is learned only on the data initially supplied. Therefore the model performance may deteriorate strongly when the data of the actual task deviates from the initial data used to learn. Online learning has the big advantage that the data is generated while performing the execution of the task. Therefore the model can react to changing conditions impacting the model during execution, e.g. when the model of the robot changes because some mass is added to the end effector. The model can then be updated using the constant feedback stream of the acquired data.
In [31] the authors note that the three biggest problems for online model learning are the speed of the learning algorithm, the large amounts of data that need to be processed and that the algorithm needs to be able to continuously process a stream of data during online learning. A robotic manipulator is typically controlled with a frequency of about 100 - 1000Hz [32] and therefore needs that many predictions in one second, while having to learn and update the model in parallel. Online model learning has one further major drawback: The model cannot be evaluated before execution and therefore the model could differ strongly from the actual model during exploration, resulting in a large model mismatch. Thus, in the worst case, it is possible that the robot may self-destruct, for example, because a joint limit is violated or the computed torque results in a too strong force. It is essential to prevent this behavior and ensure that the robot is safely controlled during the online model learning.

2.4 Robust and Adaptive Control

Robust control explicitly deals with uncertainties in control systems as it assumes that there exists some bounded error between the real and a learned model. Robust control then ensures that as long as the error between the models stays bounded by some constant, the controlled system will be stable. As model mismatch will always exist, robust control offers guarantees that the model mismatch will not impact the stability of the system. The most simple type of robust control is to choose high-gain feedback laws, which will overcome model disturbances simply by the high gains. Other robust control methods include H_∞ control [33] or Sliding Mode Control (SMC) [34]. High gain feedback is undesirable for a model learning problem because it leads to chattering torques which results in a badly conditioned learning problem. Additionally, a perfect tracking of the desired trajectory is not favorable, as small perturbations from the trajectory allow for richer data to learn from. Sliding Mode Control allows for easy design of control laws with few parameters. Chattering depends on the choice of the gains and can be reduced with an activation function. As well, stability of the controller is easy to show.

Adaptive control on the other hand also deals with uncertainties. However, it does not assume anything about the bounds of these uncertainties. Instead, it changes the control law according to the change in the system. It is however important that the change of the control parameters happens faster than the change of the robot parameters, which sometimes may be hard to achieve. Therefore, robust control is a way to stabilize a system without changing the control term during execution, while adaptive control changes the control term during execution such that the controlled system stays stable. Therefore, the parameters for robust control are set, while they vary for adaptive control.
Robust control has the advantage of better dealing with unmodeled dynamics, e.g. elasticities, can better react to quick or immediate changes of perameters, e.g. when picking up a mass and can deal with other disturbances [34]. On the other hand, an advantage of adaptive control is that it does not need any prior information about the unknown parameters, i.e. it would not need some bounds as later introduced in 3.3.
Robust control is the better choice for online model learning as it is a predictable control law to use during the exploration, it is easier to implement and is more intuitively to understand.

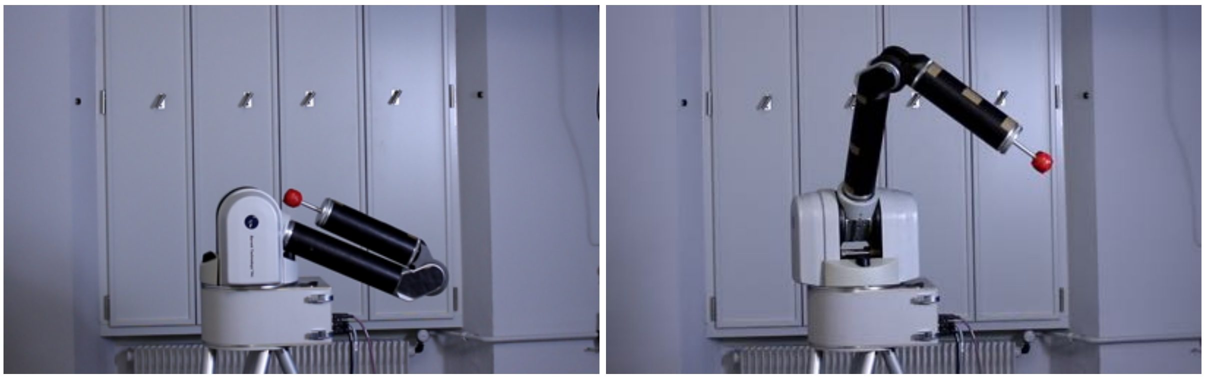


Figure 2.3.: Left: The physical Barrett WAM 4DoF robot in the lab of IAS in its resting position. **Right:** The WAM during execution of a trajectory. The base joint is rotated, the arm is tilted and rotated, the elbow joint is unfolded.

2.4.1 Control and its Impact on Learning

With very high control gains it is an easy task to achieve near-perfect position tracking. However, in the context of model learning, this is undesirable. While using high control gains ensures that almost no position error exists, it often also leads to bigger velocity and acceleration error as well as chattering torque, i.e. the torque oscillates between two values. This behavior is due to overshooting over the desired position and then applying the reverse torque to again overshoot. Such chattering results in data that is impracticable for learning, since a learning algorithm needs data that provides a relation between position, velocity, acceleration and torque. With chattering in the data, the problem becomes badly conditioned.
To summarize, it is important to choose the control law and the control gains so that the stability of the robot is ensured at all times while simultaneously creating data which is of sufficient quality to learn on.

2.5 Barrett WAM 4DoF

The Barrett Whole Arm Manipulator (WAM) 4Dof robot is a cable-driven, human-like robotic manipulator widely used in research, originally developed at MIT [35]. The WAM 4Dof mimics a human arm: The base joint as well as the second and third joint resemble the human shoulder, the final joint imitates the function of the elbow joint. The Barrett WAM is easily transformable into a 7DoF robot by replacing its lower arm module, also allowing it to mimic the function of the wrist. The WAM is a complex dynamic robot, mainly due to its direct cable drives, which induce a lot of nonlinearities, therefore making it hard to create a model analytically.
The robot has already been used for various online model learning experiments, for example in [26] or [1], where it has been demonstrated that the model of the WAM is learnable online.

2.6 Related Work

As addressed in 2.2 model learning consists of the model, the exploration and the control policy. In this thesis, I will focus on the latter two. In this section, I am going to present common approaches to the exploration and control policy during model learning.
During the exploration phase, it is necessary to stabilize the system using a controller. The choice of the exact control law is left up to the researcher. Most often some sort of computed torque control (CTC) with a PD controller is used, which is a special form of feedback linearization. This has been employed in model learning applications like [1], [26], [31], [36], [37], [38] or [39]. The disadvantage of computed torque control is the fact that control performance is bad if the model structure does not contain all of the existing dynamics, therefore assuming that the learned model is of sufficient quality [40]. Using robust control for designing a control law ensures good control performance, as long as the error on the learned model stays within predefined known bounds. It therefore guarantees stability during execution using the learned model, which is an advantage primarily for early stages of learning.
An alternative to CTC is to use a sliding mode controller (SMC), which has been used in addition to a neural network for control [41] and reinforcement learning [42]. Others [43], [44] have used a cartesian controller [45].

During the exploration phase, it is important that the used trajectory is meaningful for the robotic manipulator, i.e. all of the parameters that are relevant for the model should be employed for the execution of trajectory. Basically every

approach for exploration during online learning uses some sort of rhythmic movements, i.e. only cosine [1] or sine functions [46] or finite Fourier series (FFS) [22] in the joint space. Others have used trajectories in the task space, for example, a rhythmic movement in the form of an 8-figure in task space [47], hand-drawn characters in a virtual plane in the task space [26] or simply circles in different planes in the task space [43], [48], [49].
It is to note that only in [22] a rigorous explanation for the used trajectory and its parameters are found, while other papers do not explain their choice of trajectories.

The idea of inverse dynamics control is finding a nonlinear control law, canceling the nonlinear terms of the model, resulting in a linear system [50].
In the following, I am going to briefly present publications that I will use for the experiments in chapter 4.

2.6.1 Experimental Robot Identification using Optimised Periodic Trajectories

Swevers et al. [22] create excitation trajectories by transforming the RBDM into a set of linear equations using barycentric parameters Θ, which contain the robot parameters

$$\Phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\Theta = \tau.$$

$\Phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is a regressor matrix of dimensions $n \times r$, where n is the DoF and r is the number of parameters in Θ. During an excitation experiment, M samples can be collected to construct observation matrices of the regression matrix and the corresponding torque at each timestep

$$\mathbf{A} = \begin{pmatrix} \Phi(\mathbf{q}(t_1), \dot{\mathbf{q}}(t_1), \ddot{\mathbf{q}}(t_1)) \\ \Phi(\mathbf{q}(t_2), \dot{\mathbf{q}}(t_2), \ddot{\mathbf{q}}(t_2)) \\ \vdots \\ \Phi(\mathbf{q}(t_M), \dot{\mathbf{q}}(t_M), \ddot{\mathbf{q}}(t_M)) \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \tau(t_1)^T \\ \tau(t_2)^T \\ \vdots \\ \tau(t_M)^T \end{pmatrix}$$

i.e. $\mathbf{A}\Theta = b$.
They then use linear least squares to estimate a solution of Θ. To measure the sensitivity of this solution to disturbances the condition number of **A** can be used. The lower the condition number, the better is the resulting solution to the optimization problem and therefore enhances the model performance. Since A only depends on the regression matrix $\Phi(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$, which in turn only depends on the chosen trajectory which defines $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$, the choice of the trajectory directly influences the condition number of A. Therefore they choose excitation trajectories such that cond($\mathbf{A}$) gets minimized.
As trajectories, they suggest to use finite Fourier series

$$q_i(t) = q_0^{(i)} + \sum_{l=1}^{N} \frac{a_l^{(i)}}{\omega_f l} \sin(\omega_f l t) - \frac{b_l^{(i)}}{\omega_f l} \cos(\omega_f l t) \tag{2.1}$$

which are parametrized by $a_l^{(i)}$ and $b_l^{(i)}$ with $N = 5$. The frequency ω_f is the same for all joints, the velocities $\dot{q}_i$ and accelerations $\ddot{q}_i$ are the time derivatives of q_i. This results in a constrained optimization problem of the condition number of **A** over the parameters $a_l^{(i)}$ and $a_l^{(i)}$, with the constraints of the joint limits, maximum joint velocity, and acceleration. The constrained optimization then yields the trajectory (2.1) and the corresponding derivatives, which should provide few disturbances and therefore facilitate the model learning process.

Calafiore et al. [46] follow a similar approach and optimize the regression matrix. However, they only use harmonic sine equations as trajectories instead of finite Fourier series and have individual frequencies for the different sine terms.

2.6.2 Deep Lagrangian Networks: Using Physics as a Prior for Deep Learning

In this approach of a model learning algorithm, Lutter et al. [1] use a neural network to learn the RBDM using Lagrangian mechanics as a guide for the network. For the exploration, they use plain cosine trajectories with different frequencies for each joint during learning on the Barrett WAM 7DoF. The reasoning for using the cosine is to have zero velocity at the start of each trajectory. During the exploration phase, the robot is controlled by a low gain PD-Controller.

2.6.3 Computed Torque Control for Nonparametric Regression Models

In their work, Nguyen-Tuong et al. [36] compare the performance of Locally Weighted Projection Regression and Gaussian Process Regression for online model learning. They use feedforward nonlinear control and inverse dynamics control during the execution of trajectories consisting of two sinusoids, i.e.

$$q_i(t) = A_i \sin(2\pi f_1^i t) + A/3 \sin(2\pi f_2^i t),$$

where A_i denotes the amplitude and f_1^i and f_2^i the frequencies.

2.6.4 Summary

Various robust control strategies have been proposed for online model learning. In section 3 I am going to develop a controller for the WAM 4DoF based on [34] and [51]. Next, I am experimentally validating the controller on the real Barrett WAM 4DoF and testing the model learning performance using a system identification algorithm [2] which uses trajectories proposed for model learning in different papers [36], [1], [47], [48] and [22].

3 Robust Controller

3.1 Introduction

A general nonlinear dynamic system is typically given as an n-th order problem, such that

$$x^{(n)} = f(x, \dot{x}, \cdots, x^{(n-1)}, t)$$

where x corresponds to the state of the system and x$^{(i)}$ denotes the i-th derivative of the state and f is the dynamics.
If the system is controlled a control input u and the control gain b is added

$$x^{(n)} = f(x, \dot{x}, \cdots, x^{(n-1)}, t) + b(x, \dot{x}, \cdots, x^{(n-1)}, t)u.$$

Typically the main interest in such a system is to show that the system is stable, i.e. after some time t$_f$, it stops at an equilibrium point so that x = x$_{equilibrium}$ and x$^{(i)}$ = 0 for i = 1,..., n. If f itself is not stable, the control gain b is chosen such that the system becomes stable and has the desired equilibrium point x$^d_{equilibrium}$. This can be extended such that the system does not just tend towards the desired equilibrium point, but instead follows a trajectory, i.e. a desired state x$_d$ varying over time. The control gain b again has to be chosen such that the system tracks the trajectory as close as possible, which means to reduce the error between the actual state x and the desired state x$_d$ and its derivatives.

3.2 Sliding Mode Control

To be able to control an n-th order problem more intuitiuvely, it can be advantageous to reduce its order. A first-order problem only consisting of x and $\dot{x}$ can be controlled by simply choosing the control law such that the sign of $\dot{x}$ is inverse to that of the error of x.
This leads to the idea of Sliding Mode Control (SMC). SMC reduces an n-th order nonlinear tracking problem into a first-order stabilization problem [34]. A sliding surface $s = 0$ is defined as

$$s = (\frac{d}{dt} + \lambda)^{n-1}\tilde{x}.$$

For a second-order problem this definition would result in s = $\dot{\tilde{x}} + \lambda\tilde{x}$, where $\tilde{x}$ is the position error, $\dot{\tilde{x}}$ is the velocity error and λ is a positive weighting coefficient. By application of a first-order lowpass filter (for a proof see [34]) it becomes evident that if $s = 0$ then $\dot{\tilde{x}} = 0$ and $\tilde{x} = 0$. Therefore, to track a trajectory consisting of a desired position and velocity, it is sufficient to control $s = 0$.

Now the goal is to find a control law, such that $s = 0$ after some time t_f. For that to happen it is necessary that s always tends towards 0, i.e. the sign of the derivative of s is always the opposite sign of s, which can be formalized as

$$\frac{1}{2}\frac{d}{dt}s^2 \leq 0. \tag{3.1}$$

This condition is known as "Sliding condition" [52].
A more strict alternative is to define

$$\frac{1}{2}\frac{d}{dt}s^2 \leq -\eta s$$

which implies that $\dot{s}$ is always smaller than -η, where η is a positive constant. With this condition, one can prove that s not only tends towards 0 at all times, but that it reaches s = 0 zero in finite time.
Theorem: Lyapunov Function:
A function $V(\mathbf{q})$ is a Lyapunov function if $V(\mathbf{q})$ is positive definite and its time derivative is continuous and negative semidefinite [34].

Theorem: Stability after Lyapunov:
A system is stable in the sense of Lyapunov if a Lyapunov function $V(\mathbf{q})$ exists.

3.3 Architecture

This sliding condition (3.1) only holds for one-dimensional problems. Since the Barret WAM consists of four degrees of freedom, the approach needs to be extended to more dimensions. The sliding surface $\mathbf{s} = 0$ is defined as

$$\mathbf{s} = \dot{\mathbf{q}} - \dot{\mathbf{q}}_\mathrm{d} + \Lambda(\mathbf{q} - \mathbf{q}_\mathrm{d})$$

where Λ is a symmetric positive definite weighting matrix [34].
The sliding condition can be constructed simultaneously as

$$\frac{1}{2}(\mathbf{s}^T\, \mathbf{H}(\mathbf{q})\, \mathbf{s}) \leq -\sum_{i=1}^{4} \eta_i \mid \mathbf{s}_i \mid .$$

Using the sliding condition a Lyapunov function can be constructed. If it is possible to show that its derivative is decreasing, the system is stable in the sense of Lyapunov.

3.3.1 Sliding condition proof:

To prove the stability of the controller, a Lyapunov function $V(\mathbf{q})$ gets constructed

$$V(\mathbf{q}) = \frac{1}{2}(\mathbf{s}^T\, \mathbf{H}(\mathbf{q})\, \mathbf{s}).$$

Since $\mathbf{H}(\mathbf{q})$ is positive definite, $V(\mathbf{q})$ will always be positive. To prove the stability it needs to be shown that the derivative of V is always less than or equal to zero, i.e. the value of $V(\mathbf{q})$ monotonously decreases.

$$\begin{aligned}
\dot{V}(\mathbf{q}) &= \mathbf{s}^T\, \mathbf{H}(\mathbf{q})\, \dot{\mathbf{s}} + \frac{1}{2}\, \mathbf{s}^T\, \dot{\mathbf{H}}(\mathbf{q})\, \mathbf{s} \\
&= \mathbf{s}^T\, \mathbf{H}(\mathbf{q})(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}_\mathrm{d} + \Lambda(\dot{\mathbf{q}} - \dot{\mathbf{q}}_\mathrm{d}) + \frac{1}{2}\, \mathbf{s}^T\, \dot{\mathbf{H}}(\mathbf{q})\, \mathbf{s} \\
&= \mathbf{s}^T\, \mathbf{H}(\mathbf{q})(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}_\mathrm{ref})) + \frac{1}{2}\, \mathbf{s}^T\, \mathbf{H}(\mathbf{q})\, \mathbf{s} \\
&= \mathbf{s}^T(\tau - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}} - \mathbf{g}(\mathbf{q}) - \mathbf{H}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref}) + \frac{1}{2}\, \mathbf{s}^T\, \dot{\mathbf{H}}(\mathbf{q})\, \mathbf{s} \\
&= \mathbf{s}^T(\tau - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})(\mathbf{s} + \dot{\mathbf{q}}_\mathrm{ref}) - \mathbf{g}(\mathbf{q}) - \mathbf{H}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref}) + \frac{1}{2}\, \mathbf{s}^T\, \dot{\mathbf{H}}(\mathbf{q})\, \mathbf{s} \\
&= \mathbf{s}^T(\tau - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}_\mathrm{ref} - \mathbf{g}(\mathbf{q}) - \mathbf{H}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref}) - \mathbf{s}^T\, \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\, \mathbf{s} + \frac{1}{2}\, \mathbf{s}^T\, \dot{\mathbf{H}}(\mathbf{q})\, \mathbf{s} \\
&= \mathbf{s}^T(\tau - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}_\mathrm{ref} - \mathbf{g}(\mathbf{q}) - \mathbf{H}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref}) + \mathbf{s}^T\, \underbrace{(\dot{\mathbf{H}}(\mathbf{q}) - 2\, \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}))}_{=0}\, \mathbf{s} \\
&= \mathbf{s}^T(\tau - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}_\mathrm{ref} - \mathbf{g}(\mathbf{q}) - \mathbf{H}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref})
\end{aligned}$$

If the control term is chosen as $\tau = \hat{\mathbf{H}}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref} + \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}_\mathrm{ref} + \hat{\mathbf{g}}(\mathbf{q})$, where $\hat{\mathbf{H}}(\mathbf{q}), \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})$ and $\hat{\mathbf{g}}(\mathbf{q})$ are the estimated inertia matrix, the matrix of coriolis and centripetal forces and the vector of gravitational forces, then

$$\begin{aligned}
\dot{V}(\mathbf{q}) &= \mathbf{s}^T(\hat{\mathbf{H}}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref} + \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}_\mathrm{ref} + \hat{\mathbf{g}}(\mathbf{q}) - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}_\mathrm{ref} - \mathbf{g}(\mathbf{q}) - \mathbf{H}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref}) \\
&= \mathbf{s}^T(\hat{\mathbf{H}}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref} - \mathbf{H}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref} + \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}_\mathrm{ref} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}_\mathrm{ref} + \hat{\mathbf{g}}(\mathbf{q}) - \mathbf{g}(\mathbf{q})) \\
&= \mathbf{s}^T(\tilde{\mathbf{H}}(\mathbf{q})\, \ddot{\mathbf{q}}_\mathrm{ref} + \tilde{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}}_\mathrm{ref} + \tilde{\mathbf{g}}(\mathbf{q})).
\end{aligned}$$

With that result it becomes apparent that if the there was no model mismatch, i.e. $\tilde{\mathbf{H}}(\mathbf{q}), \tilde{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})$ and $\tilde{\mathbf{g}}(\mathbf{q})$, which denote the error between the estimated and the actual inertia matrix, the matrix of coriolis and centripetal forces and the vector of gravitational forces would all be zero, then $\dot{V}(\mathbf{q})$ would always be zero. If the desired state and the actual state were equal at the start, the sliding surface could never be left by the robot. However, since a non-existent model mismatch cannot be assumed and the sensors are noisy, it has to be ensured that those errors are compensated. This can be done by adding the term $-\mathbf{K}\,\mathrm{sgn}(\mathbf{s})$ to the control term, where $\mathbf{K}$ is a diagonal matrix of control gains and sgn() is the signum function.

$$
\begin{aligned}
\dot{V}(\mathbf{q}) = \mathbf{s}^T(\tilde{\mathbf{H}}(\mathbf{q})(q)\,\ddot{\mathbf{q}}_{\mathrm{ref}} + \tilde{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}_{\mathrm{ref}} + \tilde{\mathbf{g}}(\mathbf{q}) - \mathbf{K}\,\mathrm{sgn}(\mathbf{s})) &\leq 0 \\
\mathbf{s}^T(\tilde{\mathbf{H}}(\mathbf{q})\,\ddot{\mathbf{q}}_{\mathrm{ref}} + \tilde{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}_{\mathrm{ref}} + \tilde{\mathbf{g}}(\mathbf{q})) &\leq \mathbf{s}^T\,\mathbf{K}\,\mathrm{sgn}(\mathbf{s}) \\
\mathbf{s}^T(\tilde{\mathbf{H}}(\mathbf{q})\,\ddot{\mathbf{q}}_{\mathrm{ref}} + \tilde{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}_{\mathrm{ref}} + \tilde{\mathbf{g}}(\mathbf{q})) &\leq \sum_{i=1}^{4} \mathbf{K}_{ii} \mid \mathbf{s}_i \mid \\
\mid \tilde{\mathbf{H}}(\mathbf{q})\,\ddot{\mathbf{q}}_{\mathrm{ref}} + \tilde{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}_{\mathrm{ref}} + \tilde{\mathbf{g}}(\mathbf{q}) \mid_i &\leq \mathbf{K}_{ii}
\end{aligned}
\tag{3.2}
$$

Therefore the gains $\mathbf{K}$ have to be chosen such that the above inequality is always true. Since the exact model is not known the modeling error also needs to be estimated. The gain matrix $\mathbf{K}$ is chosen as $\mathbf{K} = \mathrm{diag}(15, 40, 8, 15)$, which values are chosen with respect to the approximate torques needed to control the WAM 4DoF only using the term $-\mathbf{K}\,\mathrm{sgn}(\mathbf{s})$.

3.3.2 Activation Function against chattering

When using the sign function another problem occurs. Since the torque is not calculated continuously in real-time but at a static frequency, the system is time-discrete. When the system is reasonably close to the sliding surface, then due to Equation 3.2 s tends towards 0. But since the next time step happens after a set time, s will not equal 0 perfectly at that next time step but instead will have changed its sign now being reasonably close to $s = 0$ from the opposite side. This problem is known as chattering, where the sign of s changes every time step, which also results in chattering of the torque.
This problem can be tackled in three different ways: First, the frequency of the controller can be increased so that one time step becomes smaller and s does not overshoot as much. Second, the value of the gain matrix $\mathbf{K}$ could be lowered to decrease the resulting force and therefore decrease the amount of overshooting. Third, instead of using the sign function an activation function can be used to smooth the chattering, which indirectly lowers the value of $\mathbf{K}$ but only in close proximity to the sliding surface.
The first two options are not feasible because the frequency has to be chosen such that a model learning algorithm can be used in parallel while online learning, which does not allow for a higher frequency. The gains k are chosen such that they fulfill the sliding condition and so that they result in strong enough torques to control the robot without any other forces applied.
Only the third option can be implemented to reduce chattering. Using a linear activation that clips the values in [-1, 1], the control law becomes

$$\tau = \hat{\mathbf{H}}(\mathbf{q})\,\ddot{\mathbf{q}}_{\mathrm{ref}} + \hat{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}_{\mathrm{ref}} + \hat{\mathbf{g}}(\mathbf{q}) - \mathbf{K}\,\mathrm{lin}_{[-1,1]}(\mathbf{s})$$

which is short for

$$\tau_i = \begin{cases} (\hat{\mathbf{H}}(\mathbf{q})\,\ddot{\mathbf{q}}_{\mathrm{ref}} + \hat{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}_{\mathrm{ref}} + \hat{\mathbf{g}}(\mathbf{q}))_i - k_{ii} & s \leq -1 \\ (\hat{\mathbf{H}}(\mathbf{q})\,\ddot{\mathbf{q}}_{\mathrm{ref}} + \hat{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}_{\mathrm{ref}} + \hat{\mathbf{g}}(\mathbf{q}))_i - k_{ii}s_i & -1 \leq s \leq 1 \\ (\hat{\mathbf{H}}(\mathbf{q})\,\ddot{\mathbf{q}}_{\mathrm{ref}} + \hat{\mathbf{C}}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}_{\mathrm{ref}} + \hat{\mathbf{g}}(\mathbf{q}))_i + k_{ii} & 1 \leq s \end{cases}$$

for i = 1,2,3,4.
Now the gains scale with the distance to the sliding surface, therefore reducing the chattering in close proximity to the sliding surface, since the full gains do not get activated at all times.

3.4 Joint Limit Avoidance

The most dangerous thing for the robot to happen during model learning would be that the joint limits are violated, i.e. one joint position equals a joint limit while still moving in direction of that joint limit. This results in a collision, depending on the velocity of the joint and could lead from deformations in the link and joint to a collapse of the robot. Therefore additional safety measures have to be implemented in the controller to ensure avoidance and a minimum distance to the joint limits.
This avoidance is implemented using an artificial spring-damper system that is only activated after some threshold $\mathbf{q}_t$

close to a joint limit is crossed.
Should a joint cross this threshold, torque calculated by

$$\tau^i_{avoidance} = -\mathrm{k}^{\mathrm{i}}_{\mathrm{p}}(q_i - q^i_t) - \mathrm{k}^{\mathrm{i}}_{\mathrm{d}}\,\dot{q}_i \tag{3.3}$$

is added to the other torque of the controller. $\tau^i_{avoidance}$ is the joint-wise torque calculated only if the threshold q^i_t for a joint is crossed. The spring coefficient $\mathrm{k}^{\mathrm{i}}_{\mathrm{p}}$ and the damping coefficient $\mathrm{k}^{\mathrm{i}}_{\mathrm{d}}$ are simply joint-wise proportional and derivative gains. This results in a simple PD-Controller in the critical area near the joint limits.

The joint limits of the WAM 4Dof are

Joint	1	2	3	4
Minimum position	-2.6	-2.0	-2.8	-0.9
Maximal position	2.6	2.0	2.8	3.2

The area equal to 5% of the range of each joint before each joint limit is chosen as the critical area where the joint limit avoidance is applied, which results in the following thresholds q_t

Joint	1	2	3	4
Lower thresholds	-2.34	-1.8	-2.52	-0.69
Upper thresholds	2.34	1.8	2.52	3.0

The gain vectors $\mathbf{k}_p$ and $\mathbf{k}_d$ are chosen as

Joint	1	2	3	4
k_p	400	600	200	200
k_d	3.5	7.5	2.5	1.25

Each used trajectory for model learning also avoids the critical area, therefore ensuring that the critical areas are only entered due to modeling errors, but not intentional.

4 Evaluation

4.1 Joint Limit Avoidance Experiment

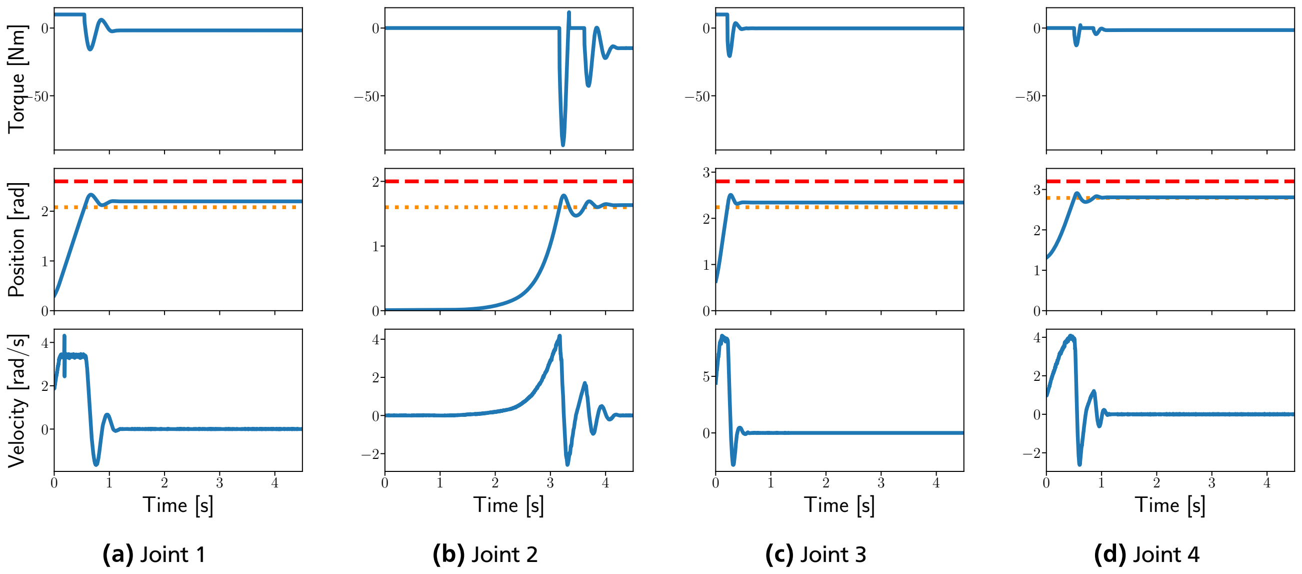


Figure 4.1.: The four different responses to the joint limit avoidance for each joint in applied torque, position and velocity is plotted in blue, the dotted orange line is the threshold after which the joint limit avoidance is active, the red dashed line is the joint limit which cannot be crossed. During execution only the shown joint is loose while all others have been stabilized.

To evaluate the performance of the implemented joint limit avoidance in the controller I am going to simulate a 'failed' model, which either does not output any torque at all or pushes with some torque in the direction of a joint limit. First, each joint but one is stabilized in a position while the one joint is loose and free to move due to an external influence like gravity. Due to the geometric construction of the robot and its installment in the lab the first and the third joint can not be influenced by gravity towards their joint limits. Therefore a torque of 10Nm is applied to each when they are loose to force them towards their joint limit. Each joint starts in the center of its range, i.e. $q_0 = [0, 0, 0, 1.15]$. The threshold after which the joint limit avoidance is activated was chosen to be 10% of the total range of each joint. The response of the joint limit avoidance is depicted in B.3. It can be seen that the trajectories are under critically damped. If an exact model would be known, e.g. by model learning, the gain parameters could be tuned such that the response would be critically damped. Also, a stationary point is adopted once the response has worn off due to the damping term which reduces the energy in the system.

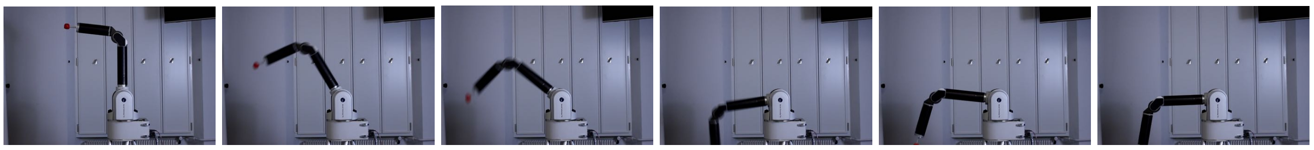


Figure 4.2.: The joint limit avoidance experiment for the second joint. The behavior of the robot is depicted from left to right. The first picture shows the initial position of the robot, in the next two pictures the robots falls due to gravity. The fourth image shows the maximum overshoot over the virtual joint limit threshold. The arm is then sprung up, until it ends in the resting position of the last picture.

Table 4.1.: The different frequencies used to create the four different cosine classes, the amplitude A and the starting position q_0 in Lutter et al.

	Joint			
cosine class	1	2	3	4
0	0.7	5	3	2
1	0.7	3	5	2
2	1.1	2	7	5
3	1.1	5	7	3

	Joint			
	1	2	3	4
A	0.866	0.66	0.933	0.683
q_0	0	0	0	1.15

4.2 Excitation Trajectories Experiment

Depending on the task the robotic manipulator performs different desired trajectories during the online learning phase. For this exploration, it is unfavorable to directly perform the task as the model learnings performance may be deteriorated based on the chosen trajectory. Instead, it is desirable to lay a baseline for the model by performing some set trajectories first, which are known to produce a stable model. Those trajectories should be designed in a way such that they support the model learning as best as possible.
In this section, I am going to evaluate the model learning performance of different trajectories proposed in literature for online model learning. The four different trajectories that I am going to use are: Cosine curves in the joint space as used in [1], optimized FFS in the joint space as proposed in [22] and a circle [44] and an 8-figure [47], each in task space. Each of those trajectories has been used for online model learning in experiments. The reasoning for using the optimized FFS trajectories is that the resulting parameters should be insensitive to perturbations of the trajectory. The cosine curves are used because of their inherent property of having a zero velocity at the start of the trajectory. For the trajectories in the task space, no motivation or explanation is given in the corresponding publications.

4.2.1 Trajectories

In the following I am going to shortly present the four different trajectories used for the later experiments. All of them have been used for a model learning application in literature.

Cosine Curves

The trajectories used are simple joint wise cosine curves following the formula

$$q_i(t) = q0_i + A_i \cos(2\pi 0.05 c_i^{(m)} t + \psi \sin(0.5\pi t))$$

where q_i denotes the joint position of the i-th joint, $q0_i$ the initial starting position of the i-th joint and A_i the amplitude, which is one-sixth of the range of each joint, resulting in using one-third of the total range of each joint. $c_i^{(m)}$ is the value for the i-th joint of the m-th of four so-called 'cosine classes', which allow for four different trajectories in total, and ψ is a sine factor. The used values for the trajectories are given in 4.1. The trajectories in joint as well as in task space can be seen in 4.3. Of the four available trajectories, I chose to use the first, i.e. cos_0, which showcased the best model learning performance of the four. Results of the four trajectories are shown in B.2. In the following, I will refer to this trajectory as cosine curve/trajectory.

Optimized Finite Fourier Series

The used trajectories for the optimized FFS have been calculated by myself using the methodology proposed in [22]. For that I used the linear model of the WAM and FFS of length 3. I optimized the condition number of the regression matrix using the python library scipy with the optimization method "trust-constr" which optimized over the parameters $a_i^{(l)}$, $b_i^{(l)}$ and $q0_i$ with $i = 0, \cdots, 3$ denoting the joint number and $l = 0, \cdots, 4$ denoting the Fourier coefficient number, resulting in the aforementioned equation

$$q_i(t) = q0_i + \sum_{l=1}^{N} \frac{a_i^l}{\omega_f l} \sin(\omega_f l t) - \frac{b_i^l}{\omega_f l} \cos(\omega_f l t).$$

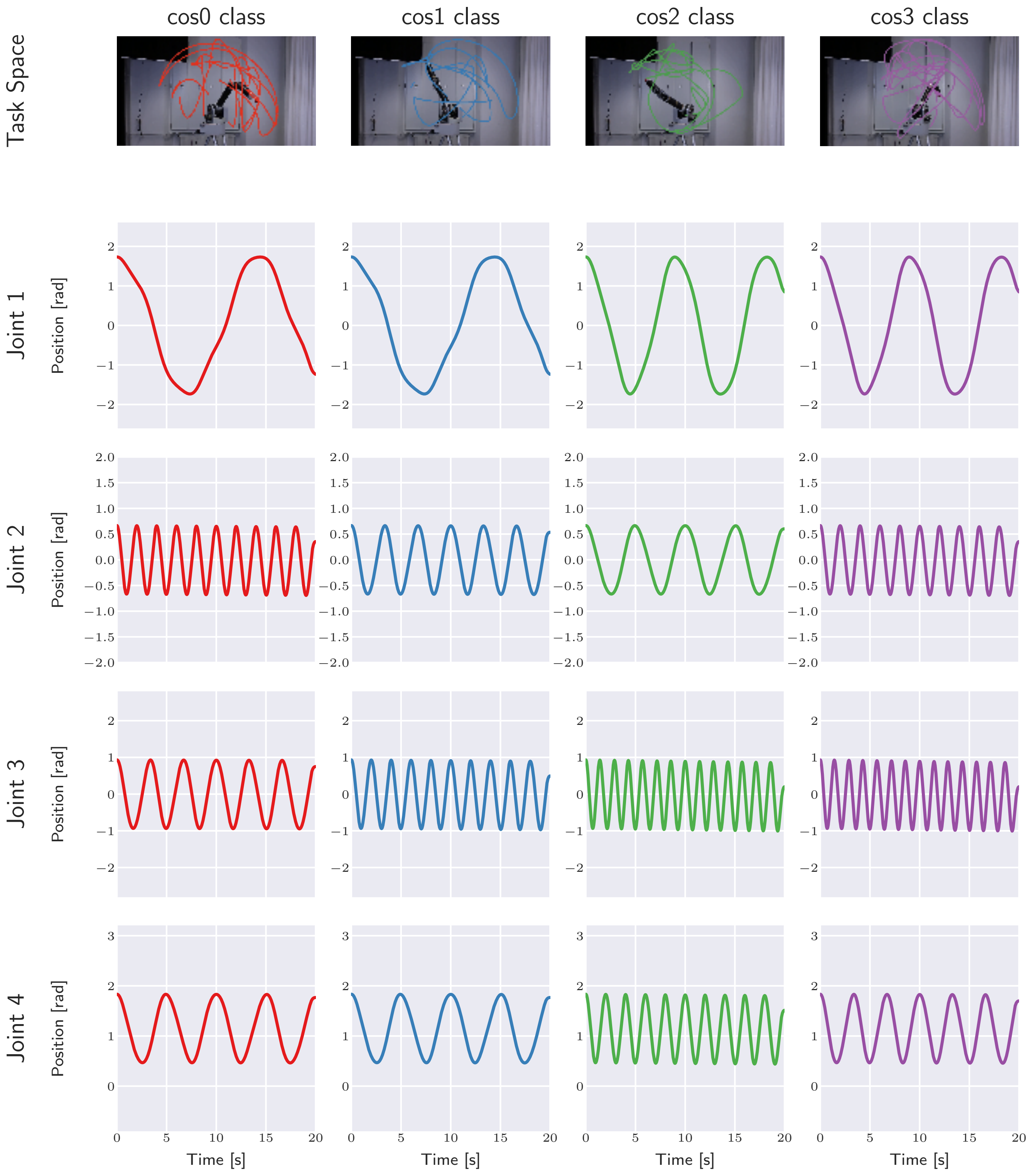


Figure 4.3.: The four different classes for trajectories used in Lutter et al. [1] depicted in the joint space as well as in the task space when executed on the physical WAM. The different frequencies of the joints can be seen that allow for strong coupling effects between the joints and also the used amplitude for the joints, which is the same for the last three joints but different for the first joint. In the task space it can be seen that the trajectory uses a wide area of the workspace without repeating a movement in the task space due to the different frequencies.

Table 4.2.: The parameters used to create the FFS of Swevers.

	Joint			
	1	2	3	4
$a^{(1)}$	0.163	-0.087	0.597	-0.244
$a^{(2)}$	0.608	0.120	-0.257	-0.583
$a^{(3)}$	-0.004	-0.330	0.299	0.014
$b^{(1)}$	-0.510	-0.112	-0.371	-0.215
$b^{(2)}$	0.006	-0.247	-0.531	-0.415
$b^{(3)}$	-0.193	0.181	0.241	-0.036
q_0	-0.362	-0.196	-0.663	0.319

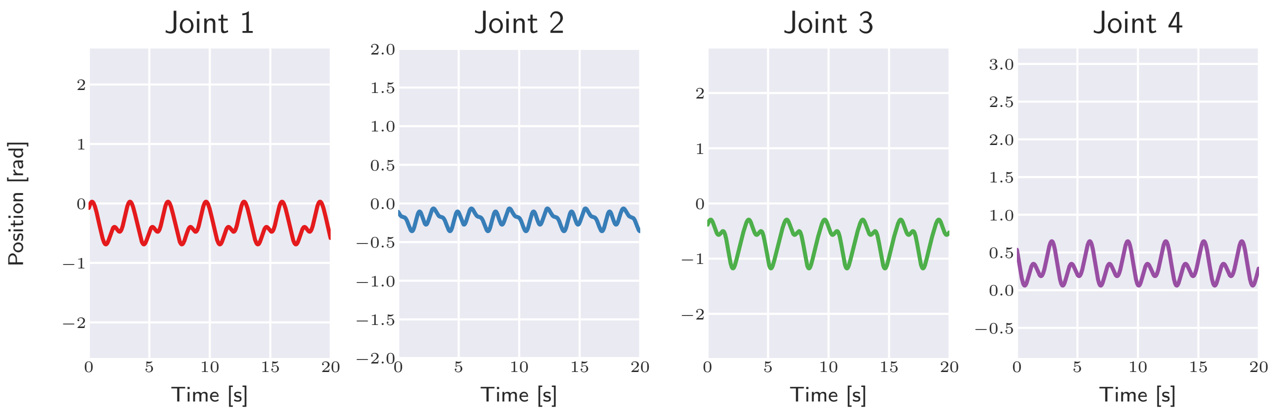


Figure 4.4.: The resulting swevers trajectory given as the joint position over time for each joint in its corresponding work space after optimization with parameters $N = 3$ and $\omega = 2$, which result in a period length of πs.

Velocity and acceleration are computed by taking the time derivative of the above equation

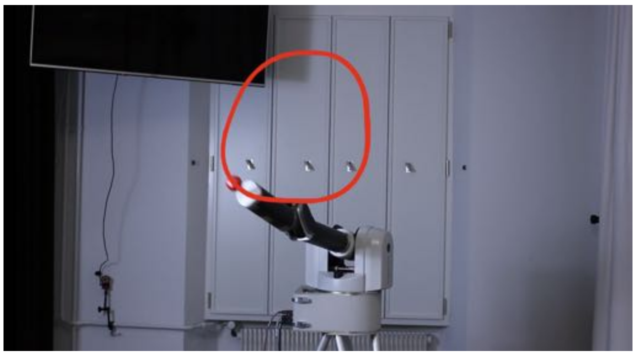
$$\dot{q}_i(t) = \sum_{l=1}^{N} a_i^l \cos(\omega_f l t) + b_i^l \sin(\omega_f l t)$$

$$\ddot{q}_i(t) = \sum_{l=1}^{N} -a_i^l \omega_f l \sin(\omega_f l t) + b_i^l \omega_f l \cos(\omega_f l t).$$

After optimization, the condition number of the matrix was 5.154e16, which is several magnitudes higher than the number of 3 reported in the original paper. This is probably because of the linear model of the WAM has more DoF than the robot used in the paper and the dynamic parameters of the WAM have not been summarized to be independent. The resulting coefficients $a_i^{(l)}$, $b_i^{(l)}$ and $q0_i$ are shown in 4.2, the resulting trajectory in the joint space in 4.4. An additional trajectory created for a FFS with length 1 can be found in the appendix. In the following I will refer to this trajectory as the swevers trajectory.

Task Space Trajectories

The trajectory of the 8-figure and the circle in task space can be seen in 4.5. They are performed in the Y-Z-plane of the WAM. Both trajectories are performed with a frequency of $1/\pi$, i.e. one circle/8-figure is completed every π seconds. In the following, I will refer to those trajectories as circle/8-figure respectively.


(a) circle

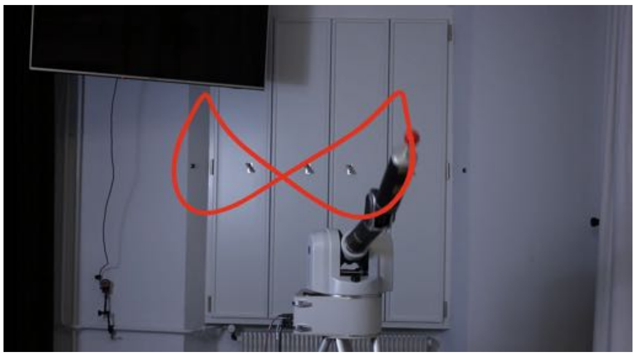
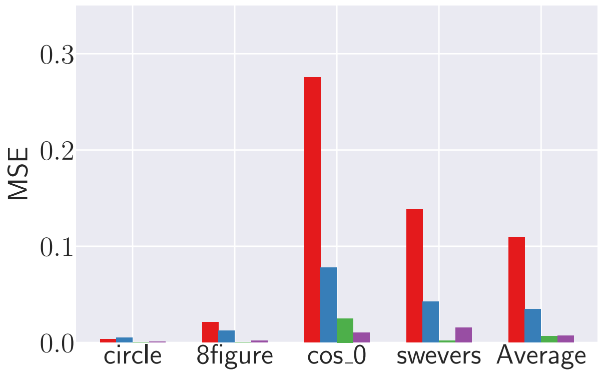

(b) 8-figure

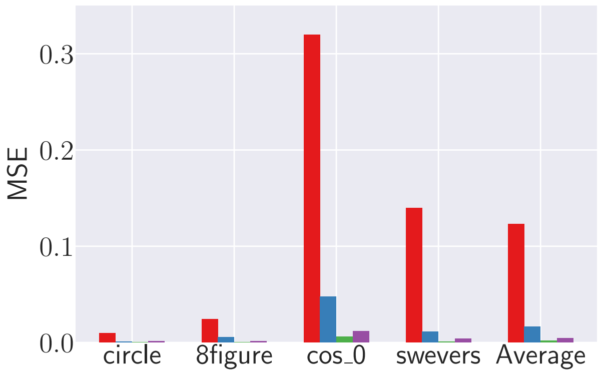
Figure 4.5.: Tracking of the task space trajectories performed on the physical WAM 4DoF using an analytical model.

4.2.2 Experiment Setup Simulation

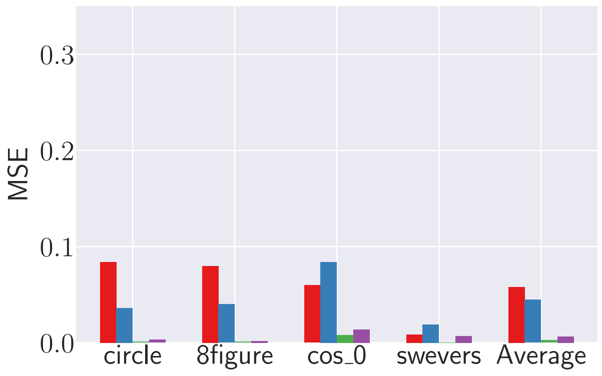
First, one of the four trajectories is executed on the simulated Barrett WAM 4DoF using the SL simulation environment. The generated data from the simulation, consisting of torque, position, velocity, and acceleration for each joint, which has been sampled at a rate of 200Hz is then used to learn a dynamics model of the robot using a system identification algorithm by Atkeson [2]. The dynamic model then consists of a total of 60 parameters, the mass of the segment, the mass times the position of the center of mass in each direction and the six inertias of the segment for each of the six segments.
The learned model is then evaluated on a total of four different test trajectories, which are the same trajectories presented in 4.2.1. During that evaluation, FF-control is applied with a PD-feedback controller. Each test trajectory is performed for 20 seconds, in which the mean squared error (MSE) is tracked. In 4.6 and 4.7 the results are shown.



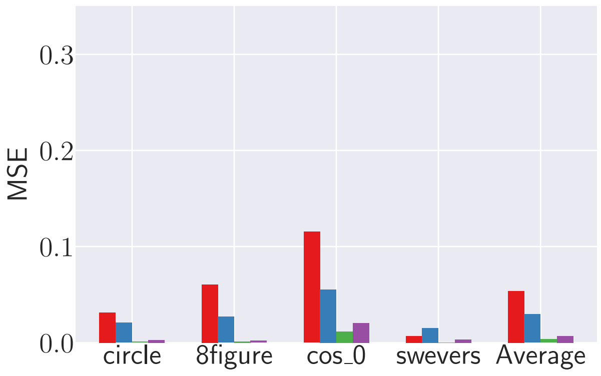
(a) circle



(b) 8-figure

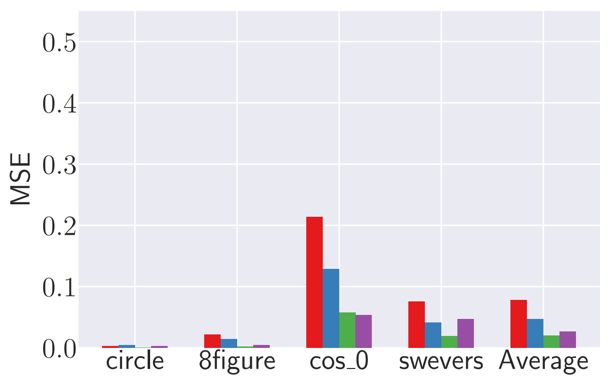


(c) cosine

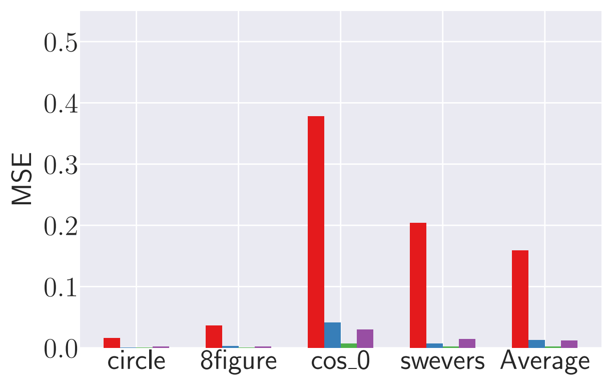


(d) swevers

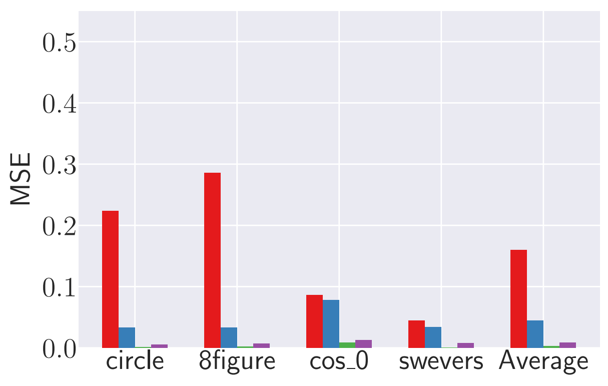
Figure 4.6.: The results of the model learning using system identification [2] in simulation, after performing the trajectory under each diagram. The mean squared error (MSE) is shown for each joint and each test trajectory as well as the average MSE for each joint over all test trajectories.



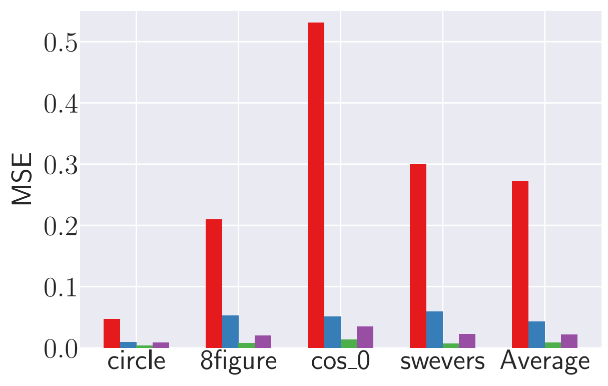
(a) circle



(b) 8-figure



(c) cosine



(d) swevers

Figure 4.7.: The results of the model learning using system identification [2] in simulation, after performing a trajectory slightly altered in comparision to 4.6. The trajectories used for evaluation are the same as in 4.6. (a): The circle trajectory is only performed with half the frequency. (b): The 8-figure is only performed with half the frequency. (c): The cosine trajectory is performed with only half the amplitude. (d): The swevers trajectory is only optimized over a FFS with parameter $N = 1$.

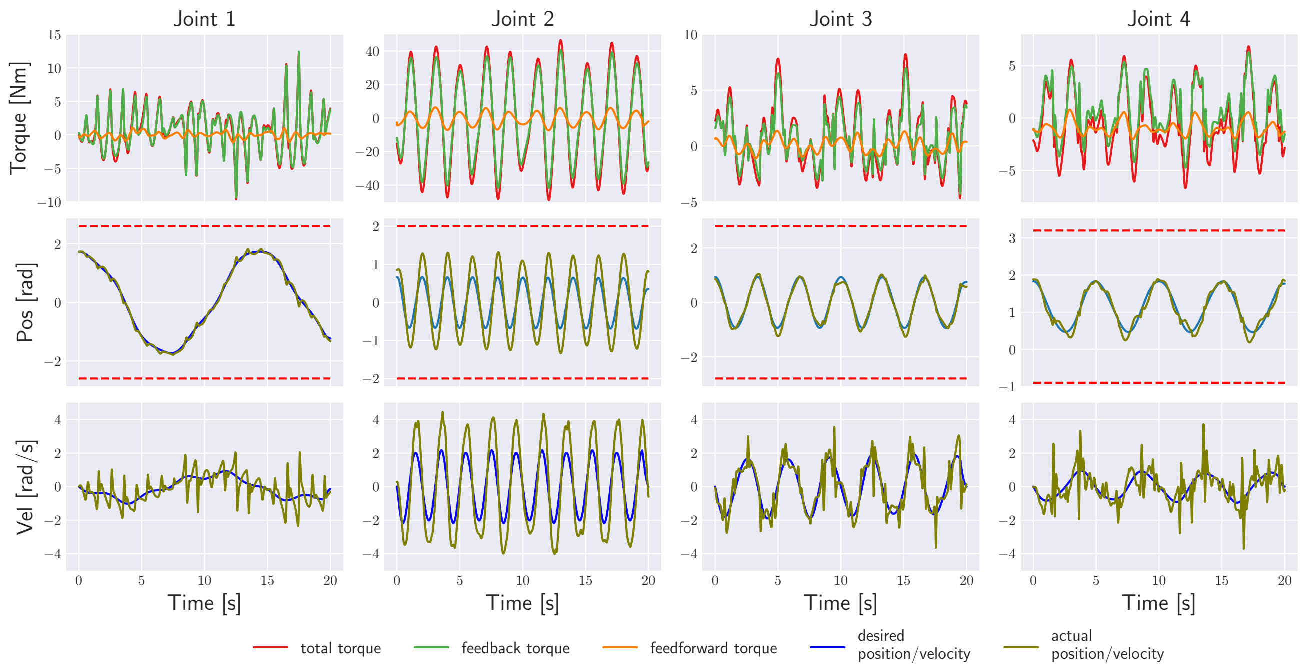


Figure 4.8.: The results of the cosine trajectory performed in simulation. On the top, the applied torque is plotted, divided in the feedforward and the feedback part. Below, the desired position and velocity as well as the actual position and velocity are plotted.

Finally, to highlight the importance of model learning, I am going to demonstrate how the analytical model of the WAM provided by the manufacturer Barrett performs. For that I will perform a trajectory on the robot, which is controlled using feedforward control, i.e. a feedforward term, which utilizes the inverse dynamics of the analytical model in form of the recursive Newton Euler algorithm [53] combined with a feedback term, which is a PD controller. This setup is performed in simulation as well as on the real robot.
The supplied model consists of the mass, the center of gravity and the six moments of inertia for each link of the robot, as well as the kinematic structure of the joints and links. During the work with the model I discovered that the provided model by Barrett is inconsistent in at least one parameter: In the description of the model the sign of one moment of inertia differs from the provided URDF file.

In 4.9 the results of a trajectory is depicted where the analytical model was used in the aforementioned setup on the physical WAM, in 4.8 the results in simulation are plotted. The used trajectory here is the same cosine curve used in section 4.2, the results are similar when using other trajectories.
First, it is to note that the actual torques required on the real robot are lower than that of the simulation, even though the simulation does not takes friction into account. Also, the position and velocity error on the physical WAM are smaller than in simulation. Apart from that, it can be seen that the feedforward part of the control term only plays a slight role in the applied torque, while the feedback term dominates the total torque, which leads to a still reasonable tracking of the trajectory. However, due to this the total torque is higher than it needs to be, resulting in higher tracking error, higher power consumption and higher wear of the robot. Looking at the second joint, the most dominant part of the robot motion and the smoothest torque curve, it is visible that the overall direction of the feedforward control is right. However, the feedforward torque would need to be stronger to apply the desired movement. Similar observations can be made for the other joints, although not as clearly visible. While the used algorithm assumes rigid body dynamics and no friction, the results still seem to be too far off the real dynamics of the robot. The WAM is advertised as nearly frictionless, therefore friction forces should not be responsible for the reality gap of the model.
When repeating the trajectory and experimentally scaling up the feedforward part the model performs better. The required torques are smaller, the feedforward part is bigger and the feedback term needed to compensate errors is smalle. Additionally, the position and velocity errors are smaller. Scaling the feedforward part has similar effects as scaling the moments of inertia for each link. Those additional results are shown in **??**.

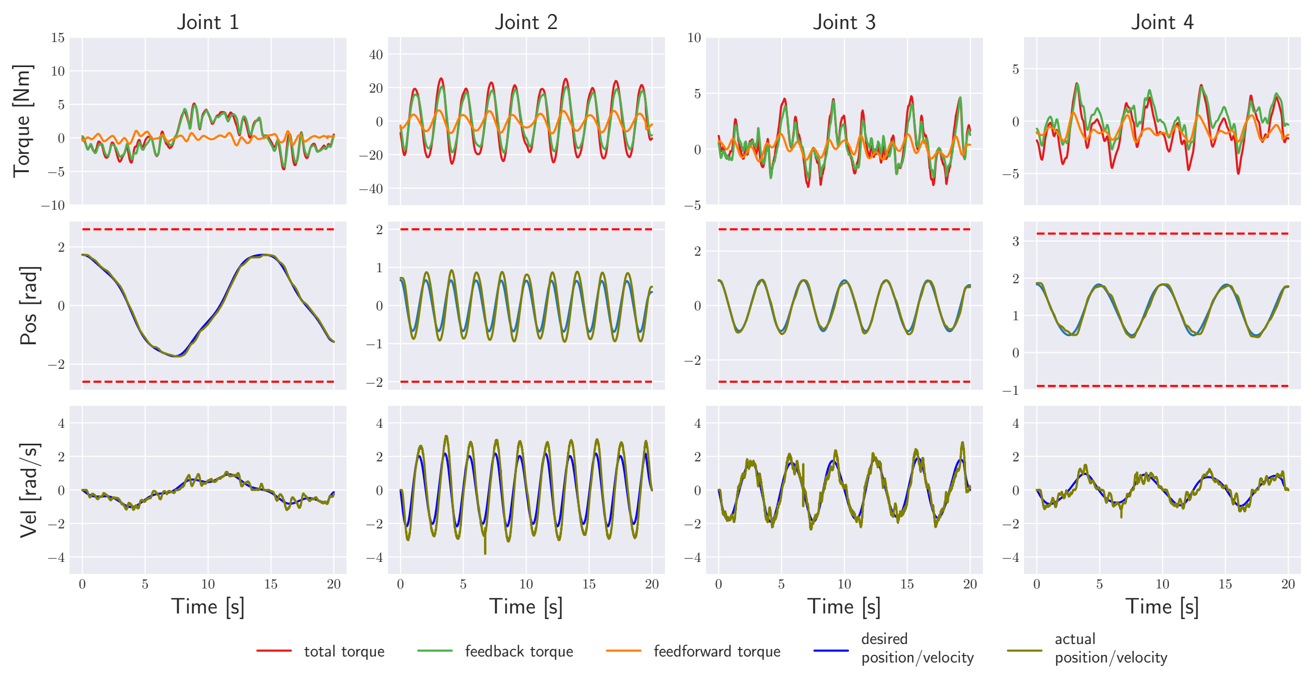


Figure 4.9.: The results of the cosine trajectory performed on the physical WAM. On the top, the applied torque is plotted, divided in the feedforward and the feedback part. Below, the desired position and velocity as well as the actual position and velocity are plotted.

While the method used to compute the inverse dynamics does not reflect the whole reality, those results show that, while the parameters might be physically plausible, they don't results in a model that is usable for the inverse dynamics. The masses might be close to the real values, however both the center of gravity and the moments of inertia are complex or impossible to calculate exactly using analytical methods. Therefore and because of the results of **??**, it is to assume that the given parameters might not be exact, especially the moments of inertia.
Thus, this example shows, why (online) model learning can be necessary to get more exact dynamic parameters of a robotic manipulator.

5 Discussion

5.1 Excitation Trajectories

The results in 4.6 indicate that the choice of the trajectory used for learning does indeed has an influence on the performance of the model learning. While the algorithm is able to learn a model with every trajectory, the tracking performance on some are better than on others.
First, it is interesting to note, that the tracking error on the different trajectories varies by quite a margin. The larger the frequency of the movement on the test trajectory the bigger the error, which is only natural because the faster movement leads to less precise tracking. The circle trajectory was performed with a frequency of $1/\pi$, i.e. a period length of πs. The optimized swevers trajectory also had a frequency of $1/\pi$ as the fundamental frequency, therefore those two are interesting to compare. The 8-figure trajectory was also performed with this frequency, however, due to the shape of the figure the second joint had a frequency of roughly $2/\pi$, i.e. double the frequency. Last, the cosine trajectory was performed with the values given in 4.1, which resulted in much faster frequencies, therefore, resulting in higher tracking errors. Also, it is interesting to note that the tracking error on the first joint is by far the largest on basically every trajectory, which indicates that this joint did not get excited enough during learning but could also be due to the fact that the first joint experiences the most coupling effects produced by the other joints, especially the second joint. Especially during trajectories with a high frequency, like the cosine curves of Lutter, this is visible.
Both the circle and the 8-figure trajectory, both trajectories of the task space, result in similar tracking errors. This comes to no surprise, as both of those trajectories were performed in about the same work space of the robot and with a similar frequency. They both learn a model that fits their own trajectories well, resulting in a very low error. When performing more complex trajectories the tracking performance however deteriorates. This should be because of two reasons: First, the frequencies of the trajectories are synchronized while they are asynchronous on the cosine trajectory which is then information missing on the data and it is harder to learn the influence of the movements of other joints. Second, the speed required for the swevers and the cosine trajectory is higher and therefore the model is not well suited for higher velocities. Also, the circle trajectory when performed with a lower frequency actually achieves better results for the tracking error of the first joint in both the cosine and the swevers trajectory, however, results in a worse performance of the other joints. This could be due to the fact that the frequency of the movement of the first joint of the trajectory and the cosine one are similar therefore learning a model that fits the cosine trajectory well. Since the other frequencies aren't too similar the tracking error becomes worse for the other joints.
On the other hand, both the cosine trajectory and the swevers one lead to quite good results. The cosine trajectory has similar tracking errors for each trajectory, however performing worse on the task space trajectories. This could be due to the fact that the used work space of the cosine trajectory and that of the task space trajectory is vastly different. It can be also seen that the tracking performance of the second joint on the cosine trajectory is better when the initially used trajectory was not the cosine trajectory. In no other combination does an initial trajectory performs worse on its own trajectory than the other ones. This in combination with the otherwise high tracking error on the first joint could imply that the learned model from the cosine trajectory actually does a tradeoff between the tracking errors of the first and second joint. Since the second joint seems to have strong coupling effects on the first joint those forces are also considered in the learned model.
Another interesting result is that even though the total range of the joint limits used by the cosine and the swevers trajectory are vastly different the learned model performs comparably. This could indicate that a more strict constraint on joint limits could be mitigated by adding higher frequency harmonic terms to the trajectory. It can be seen from 4.7 (d) that using no harmonic terms in the FFS increases the tracking error substantially while the used range of the trajectory stays about the same (see **??**). The same is true for 4.7 (c) where a reduced amplitude results in a higher tracking error. Therefore increasing the use of the range of the available work space during model learning as well as higher-order frequency terms should help in finding better models. From 4.7 (d) it can be also seen that a trajectory optimized under some optimality criterion doesn't necessarily provides the best excitation of the dynamic parameters. This result is however due to the fact that the parametrization of the trajectory is badly chosen, i.e. the form of the model itself, the FFS, the frequency and most importantly the length of the Fourier series, deteriorating the excitation.

The results presented here indicate that several factors influence the performance of model learning: First, it seems to be important that the used range of the work space should be as big as possible to capture the different dynamics of the whole work space. Otherwise, the model might not generalize well to unseen areas. Therefore, trajectories that model a

certain figure in the task space, like the circle or the 8-figure, should be avoided, unless the learned model is used for a very similar task in the same work space. Second, it seems that adding harmonic terms to the trajectories also improves the performance. Those harmonic terms could also be used to mitigate reduced performance induced by a constraint on the usable work space during learning. Third, the velocity of the movement should be in a similar area as it is during the execution of the task that the model is used for.
Still, finding the right trajectory that maximizes model learning performance is a hard task as there are infinitely many possibilities to define a trajectory. The trajectory could be a trigonometric function in the joint or task space as presented here, but could also be some other trajectory, e.g. a kinematic movement from point A to point B or some other polynomial function. Next, the trajectory has many parameters, even when only considering the case of trigonometric functions in the joint space: Choice of cosine or sine trajectories or a combination of both, the choice of using only a fundamental frequency or additional harmonic frequencies, amplitude and time that the trajectory is performed. Each of those choices influences the data generated by the trajectory and therefore directly the performance of the learned model. Finding a trajectory that actually maximizes the model performance, therefore, seems to be impossible right now as the theoretical background of the influence of each parameter has not been completely researched yet. In [29] it is stated that "[a] full understanding of the important aspects of a good solution is still lacking", referring to optimal experiment design for nonlinear systems. Additionally, it is not entirely possible to define a criterion of maximized model performance as no perfect model is known and performance always needs to measured implicitly. An extensive empirical evaluation on a real robot is additionally expensive due to wear.
Therefore the results of the carried out evaluation are a first little step towards better understanding the influence of different trajectories for model learning.

5.2 Robust Controller

The robust controller has been successfully used for the initial data generation for model learning. However, the practical use of the controller was limited, as it could not be used for the evaluation of the test trajectories due to high computational requirements. The controller requires the inertia matrix $\mathbf{H}(\mathbf{q})$, the matrix of coriolis and centripetal forces $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ and the vector of gravitational forces $\mathbf{g}(\mathbf{q})$, which are, depending on the used model learning algorithm, not explicitly available. With the system identification used in chapter 4, those values had to be computed using the inverse dynamics, which lead to six sequential evaluations of the product of the regression matrix and the dynamic parameters. This resulted in a maximum control frequency of 150Hz on the physical WAM, which was too slow to conduct the evaluation on the robot. Therefore a feedforward control had to be chosen for the test trajectories. An additional problem is that instead of the matrix $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ often only the vector $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}$ is available, such that the given proof in chapter 3 does not hold true in practice. However, since the values of coriolis and centripetal forces are small in general and therefore this problem can be neglected.
An interesting aspect to use the controller in, is a model learning algorithm where an initial model, possibly random, is given, e.g. such as in [1], where a neural network is used to learn $\mathbf{H}(\mathbf{q})$, $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})$ and $\mathbf{g}(\mathbf{q})$ and the initial configuration of the network is used for the first model.

The joint limit avoidance has been employed during all experiments, however at no point during those experiments it was necessary that the joint limit avoidance had to be activated. Still, the robustness properties were important to ensure that even when trajectories of low excitement were used, no joint limits were violated even when the used task space was unknown from the previous data.

6 Conclusion and Outlook

In this thesis, I presented the foundations of online model learning and implemented two robustness measures, a robust controller using sliding mode control and a joint limit avoidance, on the physical Barret WAM 4DoF. Both systems have been successfully tested and their stability properties have been shown. Lastly, different trajectories used in the model learning community have been evaluated regarding their excitation of dynamic parameters while using the robust controller and the joint limit avoidance. The tracking performance of the resulting model after execution of those trajectories have been measured and the results have been discussed. Additionally, the tracking performance of the analytical model has been evaluated as well.

For the future, it would be interesting to further evaluate the ideas proposed here. The robust controller and the joint limit avoidance could be used for future experiments carried out on the WAM, taking advantage of the discussed robustness properties. Additionally, it could be of interest to also look into adaptive control or more complex methods of robust control, as sliding mode control is one of the simplest forms of robust control. While in the end it is important that the robot is sufficiently stabilized during the model learning process, more sophisticated methods like H_∞ control may lead to better performance for model learning. Additional experiments could be carried out where the dynamic parameters are changed during the model learning process, e.g. by adding a mass at the end effector to test the limits of the robustness.

The analytical model of the WAM hasn't looked too exact. It should be verified whether the given parameters correspond to the real values. Alternatively, a model could be created using model learning.

The evaluation of different excitation trajectories should be extended to include more different parameters. It would be interesting to test whether the found properties of excitation trajectories also hold true when applied for different model learning algorithms, e.g. DeLaN [1] or Local Gaussian Process Regression [26], or if the trajectories are used on different robots, especially when the amount of DoFs is different from that of the WAM. Other properties to research could be the impact of the total time that a trajectory is performed. In the trajectories used here, several periods were performed. It could be of interest, whether the data at the same point of time in a period is providing additional information or not. This could potentially decrease the time needed for the model learning if the data from one period is processed well. The influence of coupling effects, inferred by different frequencies of the movements of the joints could also be further investigated. While the frequency in [22] is *a priori* set for all joints, it is part of the optimization in [46]. It would also be interesting to examine the impact of the initial data by using other performance measures than the MSE, e.g. also taking into account the velocity error or the jerk, the derivative of the acceleration, of the performed trajectory.

Bibliography

[1] M. Lutter, C. Ritter, and J. Peters, "Deep lagrangian networks: Using physics as model prior for deep learning," in *International Conference on Learning Representations (ICLR)*, 2019.

[2] C. G. Atkeson, C. H. An, and J. M. Hollerbach, "Estimation of inertial parameters of manipulator loads and links," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 101–119, 1986.

[3] M. E. Moran, "The da vinci robot," *Journal of Endourology*, vol. 20, no. 12, pp. 986–990, 2006. PMID: 17206888.

[4] E. Hoffmann, *Der Sandmann*. Stuttgart: Reclam, 1991.

[5] S. Y. Nof, *Handbook of Industrial Robotics*. New York, NY, USA: John Wiley & Sons, Inc., 2nd ed., 1999.

[6] B. Singh, N. Sellappan, and P. Kumaradhas, "Evolution of industrial robots and their applications," 2013.

[7] B. Gates, "A robot in every home," *Scientific American*, vol. 296, no. 1, pp. 58–65, 2007.

[8] R. W. H. II and E. L. Hall, "Survey of robot lawn mowers," in *Intelligent Robots and Computer Vision XIX: Algorithms, Techniques, and Active Vision* (D. P. Casasent, ed.), vol. 4197, pp. 262 – 269, International Society for Optics and Photonics, SPIE, 2000.

[9] V. Milanés, D. F. Llorca, B. M. Vinagre, C. González, and M. A. Sotelo, "Clavileño: Evolution of an autonomous car," in *13th International IEEE Conference on Intelligent Transportation Systems*, pp. 1129–1134, Sep. 2010.

[10] S. Jung and H. Kim, "Analysis of amazon prime air uav delivery service," *Journal of Knowledge Information Technology and Systems*, vol. 12, pp. 253–266, 04 2017.

[11] D. Koert, S. Trick, M. Ewerton, M. Lutter, and J. Peters, "Online learning of an open-ended skill library for collaborative tasks," in *Proceedings of the International Conference on Humanoid Robots (HUMANOIDS)*, 2018.

[12] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, 07 2015.

[13] S. K. Lim, D. I. Park, Y. K. Kwak, B. Kim, and S. Jeon, "Variable geometry single-tracked mechanism for a rescue robot," in *IEEE International Safety, Security and Rescue Rototics, Workshop, 2005.*, pp. 111–115, June 2005.

[14] A. A. Transeth and K. Y. Pettersen, "Developments in snake robot modeling and locomotion," in *2006 9th International Conference on Control, Automation, Robotics and Vision*, pp. 1–8, Dec 2006.

[15] M. Shoham, M. Burman, E. Zehavi, L. Joskowicz, E. Batkilin, and Y. Kunicher, "Bone-mounted miniature robot for surgical procedures: Concept and clinical applications," *IEEE Transactions on Robotics and Automation*, vol. 19, pp. 893–901, Oct 2003.

[16] H. Zhuang, H. Gao, Z. Deng, L. Ding, and Z. Liu, "A review of heavy-duty legged robots," *Science China Technological Sciences*, vol. 57, pp. 298–314, Feb 2014.

[17] M. W. Spong, *Robot Dynamics and Control*. New York, NY, USA: John Wiley & Sons, Inc., 1st ed., 1989.

[18] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters, "Learning inverse dynamics models in o(n) time with lstm networks," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 811–816, Nov 2017.

[19] P. Fisette, B. Raucent, and J. C. Samin, "Minimal dynamic characterization of tree-like multibody systems," *Nonlinear Dynamics*, vol. 9, pp. 165–184, Feb 1996.

[20] P. Song, P. Kraus, V. Kumar, and P. Dupont, "Analysis of Rigid-Body Dynamic Models for Simulation of Systems With Frictional Contacts ," *Journal of Applied Mechanics*, vol. 68, pp. 118–128, 06 2000.

[21] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.

[22] J. Swevers, C. Ganseman, J. D. Schutter, and H. V. Brussel, "Experimental robot identification using optimised periodic trajectories," *Mechanical Systems and Signal Processing*, vol. 10, no. 5, pp. 561 – 577, 1996.

[23] D. Nguyen-Tuong and J. Peters, "Using model knowledge for learning inverse dynamics," in *2010 IEEE International Conference on Robotics and Automation*, pp. 2677–2682, May 2010.

[24] D. Nguyen Tuong and J. Peters, "Model learning in robotics: a survey," no. 4, 2011.

[25] D. Nguyen Tuong and J. Peters, "Learning inverse dynamics: a comparison," in *Proceedings of the European Symposium on Artificial Neural Networks (ESANN 2008)*, 2008.

[26] D. Nguyen-Tuong, M. Seeger, and J. Peters, "Model learning with local gaussian process regression," *Advanced Robotics*, vol. 23, pp. 2015–2034, 10 2009.

[27] O. Herbort, M. Butz, and G. Pedersen, *The SURE-REACH Model for Motor Learning and Control of a Redundant Arm: From Modeling Human Behavior to Applications in Robotics*, vol. 264, pp. 85–106. 12 2009.

[28] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," *Artificial Intelligence Review*, vol. 11, pp. 75–113, Feb 1997.

[29] J. Schoukens and L. Ljung, "Nonlinear system identification: A user-oriented roadmap," *CoRR*, vol. abs/1902.00683, 2019.

[30] K. S. NARENDRA and A. M. ANNASWAMY, "Persistent excitation in adaptive systems," *International Journal of Control*, vol. 45, no. 1, pp. 127–160, 1987.

[31] D. Nguyen–Tuong and J. Peters, "Incremental sparsification for real-time online model learning," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 557–564, PMLR, 13–15 May 2010.

[32] J. Schrimpf, "Sensor-based real-timecontrol of industrial robots," 2013.

[33] J. C. Doyle, K. Glover, P. P. Khargonekar, and B. A. Francis, "State-space solutions to standard h/sub 2/ and h/sub infinity / control problems," *IEEE Transactions on Automatic Control*, vol. 34, pp. 831–847, Aug 1989.

[34] J. Slotine and W. Li, *Applied Nonlinear Control*. Prentice Hall, 1991.

[35] B. Rooks, "The harmonious robot," *Industrial Robot: An International Journal*, vol. 33, no. 2, pp. 125–130, 2006.

[36] D. Nguyen-Tuong, M. W. Seeger, and J. Peters, "Computed torque control with nonparametric regression models," *2008 American Control Conference*, pp. 212–217, 2008.

[37] D. Nguyen-Tuong, J. Peters, M. Seeger, and B. Schölkopf, "Learning inverse dynamics: A comparison," pp. 13–18, 01 2008.

[38] Y. Choi, S. Cheong, and N. Schweighofer, "Local online support vector regression for learning control," in *2007 International Symposium on Computational Intelligence in Robotics and Automation*, pp. 13–18, June 2007.

[39] O. Koc, G. Maeda, and J. Peters, "Optimizing execution of dynamic goal-directed robot movements with learning control," *CoRR*, vol. abs/1807.01918, 2018.

[40] A. Karakasoglu, S. I. Sudharsanan, and M. K. Sundareshan, "Identification and decentralized adaptive control using dynamical neural networks with application to robotic manipulators," *IEEE Transactions on Neural Networks*, vol. 4, pp. 919–930, Nov 1993.

[41] S. S. GE, T. H. LEE, and E. G. TAN, "Adaptive neural network control of flexible joint robots based on feedback linearization," *International Journal of Systems Science*, vol. 29, no. 6, pp. 623–635, 1998.

[42] M. Obayashi, N. Nakahara, K. Yamada, T. Kuremoto, K. Kobayashi, and L. Feng, *A Robust Reinforcement Learning System Using Concept of Sliding Mode Control for Unknown Nonlinear Dynamical System*. 04 2011.

[43] D. Romeres, M. Zorzi, R. Camoriano, S. Traversaro, and A. Chiuso, “Derivative-free online learning of inverse dynamics models,” *CoRR*, vol. abs/1809.05074, 2018.

[44] R. Camoriano, S. Traversaro, L. Rosasco, G. Metta, and F. Nori, “Incremental semiparametric inverse dynamics learning,” pp. 544–550, 05 2016.

[45] U. Pattacini, F. Nori, L. Natale, G. Metta, and G. Sandini, “An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots,” pp. 1668–1674, 10 2010.

[46] G. Calafiore, M. Indri, and B. Bona, “Robot dynamic calibration: Optimal excitation trajectories and experimental parameter estimation,” *Journal of Robotic Systems*, vol. 18, pp. 55–68, 2 2001.

[47] S. Vijayakumar, A. D’Souza, and S. Schaal, “Incremental online learning in high dimensions,” *Neural Computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[48] D. Romeres, M. Zorzi, R. Camoriano, and A. Chiuso, “Online semi-parametric learning for inverse dynamics modeling,” in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 2945–2950, Dec 2016.

[49] A. Jaritz and M. W. Spong, “An experimental comparison of robust control algorithms on a direct drive manipulator,” *IEEE Trans. Contr. Sys. Techn.*, vol. 4, pp. 627–640, 1996.

[50] H. Wang and Y. Xie, “Adaptive inverse dynamics control of robots with uncertain kinematics and dynamics,” *Automatica*, vol. 45, no. 9, pp. 2114 – 2119, 2009.

[51] A. Mitra, N. Das, R. N. Samant, and L. Behera, “Control of a 4 dof barrett wam robot — modeling, control synthesis and experimental validation,” in *2016 IEEE First International Conference on Control, Measurement and Instrumentation (CMI)*, pp. 397–402, Jan 2016.

[52] J.-J. E. Slotine, “Sliding controller design for non-linear systems,” 1984.

[53] J. M. Hollerbach, “A recursive lagrangian formulation of maniputator dynamics and a comparative study of dynamics formulation complexity,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 10, pp. 730–736, Nov 1980.

[54] J. Jin and N. Gans, “Parameter identification for industrial robots with a fast and robust trajectory design approach,” *Robotics and Computer-Integrated Manufacturing*, vol. 31, pp. 21 – 29, 2015.

[55] F. Ghorbel, A. Fitzmorris, and M. W. Spong, “Robustness of adaptive control of robots: Theory and experiment,” in *Advanced Robot Control* (C. Canudas de Wit, ed.), (Berlin, Heidelberg), pp. 1–29, Springer Berlin Heidelberg, 1991.

[56] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

A Trajectory Tracking

To visualize the movements of the robot in the task space a simple tool was developed to track the end effector. This tracking is especially helpful for interpreting the tracking performance of a learned model.
First, the end effector has been painted in red to create a point that is easily trackable. A video of the trajectory is then converted into the HSV color space. After that, the video is transformed into a binary video by thresholding the video frame by frame. The threshold is chosen such that red pixels in the video turn white and all others black. The resulting frames are then eroded and dilated to remove any outliers. Using the OpenCV library [56], the biggest remaining part is found and its center coordinates are saved.
Once all those coordinates have been found, n many coordinates are connected in the final video, resulting in a trail behind the performed trajectory. This trail can be seen in A.3 where a circle trajectory is tracked. It is visible that the circle is not a perfect circle but is rather edgy.
The tool is fully parameterizable: The length of the trail in frames can be specified, as well as the width of the trail and the color of the trail can be changed after a set time to visualize changes during the trajectory. This behavior is shown in A.1. Also, it is possible to track other colors than red.

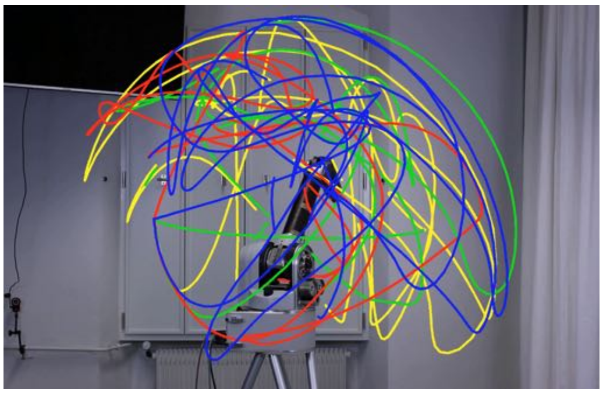


Figure A.1.: The four cosine trajectories given in 4.2.1 plotted in different colors in the task space.

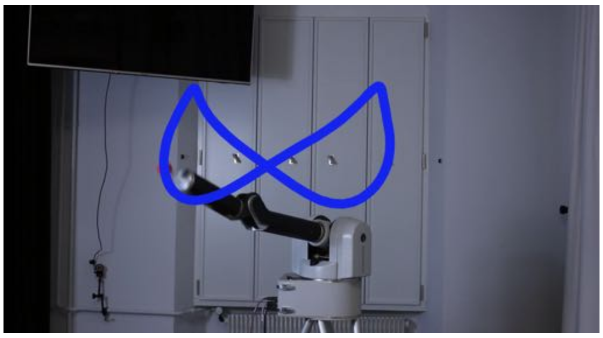


Figure A.2.: The 8-figure plotted with a thick line.

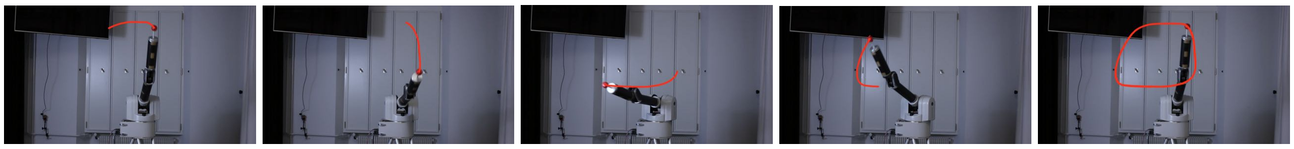


Figure A.3.: The tracking of a circle trajectory from left to right. In the right picture the whole trajectory can be seen.

B Additional Figures

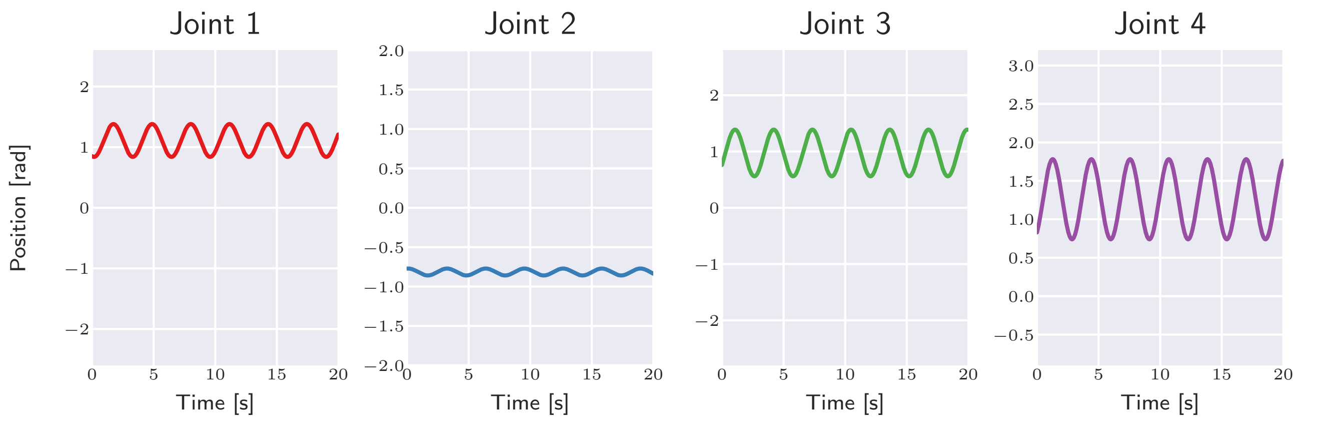
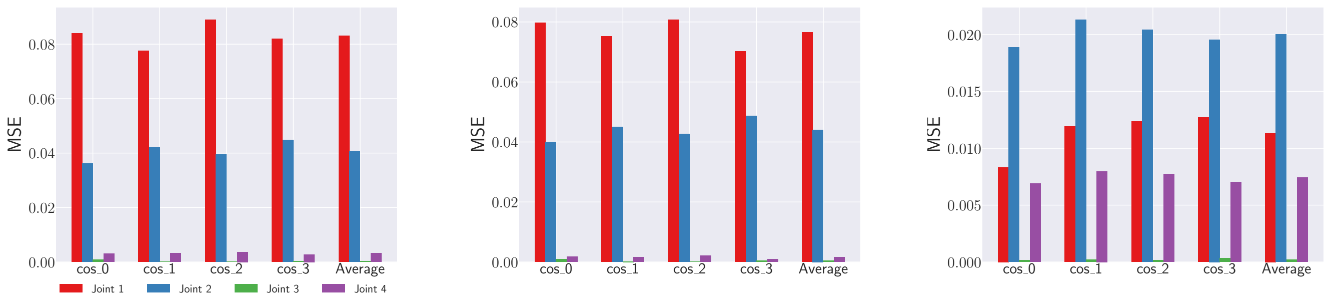


Figure B.1.: The results of feedforward control using the analytical model in simulation when scaling up the feedforward torques by a factor of three.



a: circle **b:** 8-figure **c:** swevers

Figure B.2.: The tracking errors for the different cosine curves used in Lutter et al. [1] with the parametrizations given in figure 4.1. (a): Using each of the four cosine classes to learn a model and then performing the circle trajectory. (b) + (c): The tracking error when testing on the 8-figure and the swevers trajectory respectively.

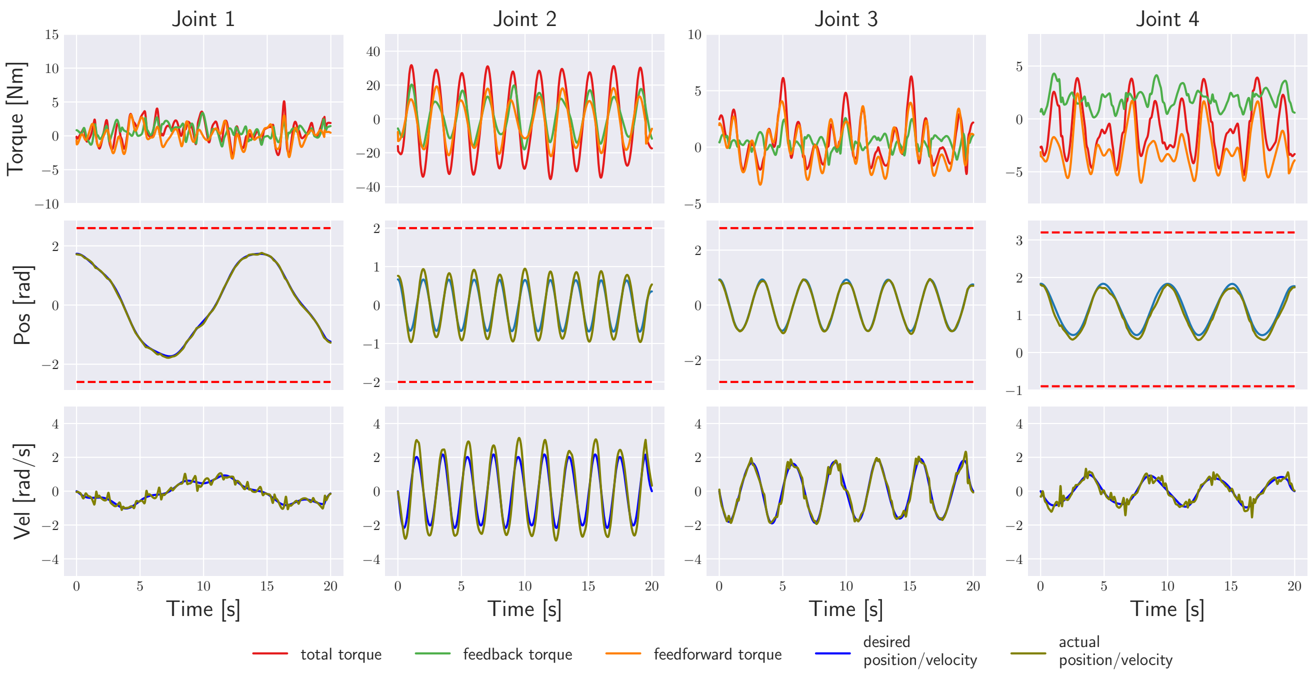


Figure B.3.: The results of feedforward control using the analytical model in simulation when scaling up the feedforward torques by a factor of three.