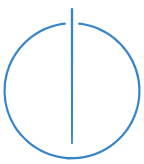
DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Robotics, Cognition, Intelligence

Orthogonal Manifold Foliations for Impedance Control of Redundant Kinematic Structures

Arne Sachtler



DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Robotics, Cognition, Intelligence

Orthogonal Manifold Foliations for Impedance Control of Redundant Kinematic Structures

Orthogonale Blätterungen von Mannigfaltigkeiten zur Impedanzregelung redundanter Roboterkinematiken

Author:	Arne Sachtler
Supervisor:	Prof. Dr. Alin Albu-Schäffer
Advisors:	Korbinian Nottensteiner, Dr. rer. nat. Freek Stulp
Submission Date:	15 April 2020

Declaration

I confirm that this master's thesis with the title *Orthogonal Manifold Foliations for Impedance Control of Redundant Kinematic Structures* is my own work and I have documented all sources and material used.

Arne Sachtler, Munich, April 15, 2020

Abstract

Redundant manipulators have more degrees of freedom then minimally required in order to perform the main manipulation task. This provides more flexibility during the manipulation task as it allows for simultaneous secondary tasks like obstacle avoidance or modification of dynamical properties. However, the overall system becomes underdetermined and methods for redundancy resolution are required.

One technique for redundancy resolution is task space augmentation. Besides the task space coordinates, another set of coordinates is defined to be used to determine the configuration of a manipulator. Linear projection is used to ensure that the main task is not disturbed by the new coordinates. In this thesis a novel kind of task space augmentation is designed, which is based on dynamical decoupling. By construction, these new coordinates are dynamically independent and no projection is required. Controllers in both sets of coordinates can be superimposed without mutual interference.

The additional new set of coordinates is computed by a coordinate function with certain properties. The mapping from joint space to task space can be seen as a foliation of the joint space manifold, where the leaves correspond to the self-motion manifolds. Based thereon, relations of the Jacobian between the task space forward kinematics and the Jacobian of the desired coordinate function are derived. These relations can be described as an underdetermined system of partial differential equations. In order to find an approximate solution to this, a variational principle is employed. In particular, the desired coordinate function is written as a neural network and the derived requirements on the Jacobians are translated to a cost function. Training of the neural network simultaneously finds a concrete instantiation of the PDE as well as a solution to it.

Trained models for different planar robots are evaluated in different settings. Kinematic evaluation shows decoupling of the two sets of coordinates on first-order dynamics, which is generally not provided by traditional augmentation methods. Afterwards, the model is evaluated using simulation of closed-loop dynamics. Impedance controllers in both coordinate sets control a simulated planar robot. In contrast to the kinematic analysis some couplings are observable on actual multi-body dynamics. The majority of couplings is due to Coriolis and centrifugal forces and terms related to the change of the Jacobian. An additional feed-forward controller compensating the major couplings achieves dynamically decoupled coordinates.

The developed method provides a technique to automatically find dynamically decoupled coordinates, which can be used for impedance control of redundant robots. These can also be interpreted as providing potentials and geodetic springs which are advantageous for controller design.

Contents

1 Introduction

Many modern robots have complex kinematic structures with a lot of degrees of freedom. Each joint provides additional freedom of motion in order to interact with humans or to manipulate the environment. Figure 1.1 shows *Justin*, a modern robot developed within the Institute of Robotics and Mechatronics at the German Aerospace Center (DLR). In comparison to traditional industrial robots, this robot is designed using light weight technology. Torque-sensing and advanced control schemes allow for safe interaction with humans. An overview on the control approaches operating this robot can be found in [11].

However, many degrees of freedom also result in kinematically redundant structures. A structure is called redundant when it is equipped with more degrees of freedom than minimally required to perform the main manipulation task [34]. Redundancy provides more flexibility when performing manipulation tasks, but also increases the complexity of control algorithms.

For redundant manipulators, the task space controller does not require all degrees of freedom. This does not necessarily mean that the controller leaves out distinctive joints in the structure, rather it is only using a subspace of the space spanned by all the joints. In other words, the task controller does not control the joints individually, but certain combinations of them. Redundancy resolution techniques allow to control the remaining degrees of freedom in order to achieve desired properties.

Figure 1.1.: Rollin' Justin. A wheeled robot with humanoid upper part. Modern robots are complex structures with many degrees of freedom. Image taken from [4].

Classical approaches use a projective technique [80, 34]. This allows to design arbitrary controllers that compute a joint space torque. A projection matrix then removes all components of that vector that would interfere with the main task. In this thesis a new approach is developed, which by construction does not require projection.

1.1. Motivation

Redundant robots provide more degrees of freedom than required to perform the main manipulation task. As shown in Figure 1.2, the end-effector can be held at the same pose while constantly reconfiguring the elbow of the robot arm. The remaining degrees of freedom provide freedom to perform additional tasks. Imagine, for instance, a camera mounted on the elbow of the robot. Reconfiguration of the arm can move the camera around without affecting the main task performed with the gripper.

Figure 1.2.: Self-motion of a DLR LBR robot arm. While the end-effector pose is kept constant, the robot can smoothly move the arm and reconfigure itself. Image source: [48].

Obstacle avoidance is a typical example for additional tasks for redundant robots. In environments with moving obstacles, the robot can react and avoid collisions without disturbing the manipulation task at the end-effector. Maciejewski and Klein [47] already have shown examples for this in 1985. For parallel or tree-structured robots like Rollin' Justin the redundancy can also be used for self-collision avoidance [15]. Additionally, for multi-arm setups redundancy can be used to avoid collisions between the individual robot arms. For example the DLR MiroSurge robotic system for telesurgery [25] consists of multiple redundant 7-DoF arms. There, collision avoidance between the individual arms is one of the tasks performed using the redundancy. Simultaneously, additional tasks are performed in the remaining degrees of freedom by optimizing kinematic properties.

Kinematic properties are optimized by, for instance, keeping the configuration away from joint limits and kinematic singularities. Hutzl et al. used a redundant robot to optimize configurations such that future motions are not constrained by joint limits [31]. This involves a prediction step for future motions and optimizing the current configuration for those to become feasible. Also the so-called manipulability index can be optimized using redundancy.

The manipulability index measures how well a robot can generate velocities and forces in the Cartesian directions of the end-effector [92]. For example in [85] a manipulability optimization is used in the field of medical robotics.

Also publications using the redundancy in order to optimize the dynamical properties of the robot are available. As early as 1990 Walker showed a concept of reduction of impact and contact effects based on redundancy in [89]. Also more recent publications use redundancy to reduce the reflected mass [49] or to minimize the energy of blunt elastic impacts [69]. Ficuciello et al. optimize the dynamical properties for more intuitive interaction with humans [20]. In a hand writing task executed by a light weight robot, they optimize the simulated inertia of the robot.

1.2. Problem Statement

Clearly, kinematic redundancy provides a lot of advantages and seems superior to non-redundant constructions. Also the human arm has seven degrees of freedom and is, therefore, a kinematically redundant system. However, it also comes with complications to the control algorithms as well as path and trajectory planning for the systems. The application of methods to determine the particular configuration a robot will have while executing a task, is called redundancy resolution. The state of the art approach for controlling such systems provides a quite flexible framework [80]. Basically, it allows to specify arbitrary controllers and control goals, which are then projected in a way that they do not interfere with the main task. This involves computation of a properly weighted projection matrix that ensures that no disturbance of the main task controller occurs [34]. As the projection matrix projects arbitrary vectors into the null space of the Jacobian matrix, this approach is often called null space control in the literature.

While being very flexible, this approach also has some drawbacks. Klein and Huang pointed out that a controller based on the projective null space control method can drive the system to unpredictable configurations [39]. Projection also complicates the process of showing stability. Usually, it is required to assume convergence of the null space controller before stability of the main task can be showed. Additionally, often the approach does not provide a minimal representation of the robot arm configuration in terms of coordinates. Specification of the task coordinate is usually straightforward, often it will be the pose of the end-effector in terms of its position and orientation. However, there are no straightforward "coordinates" for the remaining degrees of freedom. If so, they are hand-designed coordinate functions valid for only one robot and set of task coordinates (e.g. [78]).

The goal of this thesis is to develop a general method to find a minimal representation of the remaining degrees of freedom. This will be a kind of coordinate function, where these additional coordinates together with the task space coordinates fully specify the configuration of the robot. Additionally, this coordinate function shall be designed such that no projection is required for the closed-loop control. By construction, any motion in this additional set of coordinates shall not disturb the task controller and vice versa. Controllers in the new coordinates can be designed using the same control algorithms known from main task control. They simply operate on another set of coordinates. Also stability proofs known for the main task control can directly be adapted to those new coordinates. The non-disturbing nature of the coordinate is achieved by dynamically decoupling them from the task space coordinates.

The controller that operates the main task of the robot will be called task space controller. Usually, that controller will control the pose of the end-effector of the robot. A second controller operates in the new coordinates providing a minimal representation of the remaining degrees of

freedom. As pure motion in those coordinates will only effect the internal motions of the arm without interfering with the task space, it will be called self-motion controller. Similarly, the set of coordinates are called task space coordinates and self-motion coordinates, respectively. With the help of the dynamical decoupling, both controllers can simply be superimposed without interfering with each other. To summarize:

Problem Statement. *Develop a method to automatically find minimal coordinates for the remaining degrees of freedom of redundant manipulators. The target coordinate function shall be dynamically decoupled from the task space coordinates such that controllers in both coordinates can be superimposed without interference.*

1.3. Approach

Using classical pseudo-inverse based control techniques [34], solutions for the velocities tangent to the self-motion manifolds are provided. However, there is no possibility to integrate the desired coordinate function. This argument is based on the non-integrability of the vector field distribution determined by the pseudo-inverse of the Jacobian of the manipulator [39]. It is definitely true that a pseudo-inverse is generally not integrable. In contrast to the state of the art approach, here a decoupling without the pseudo-inverse is targeted. In this thesis an argument is provided that allows to show that decoupled self-motion coordinates exist.

Under a differential geometric perspective on the task space coordinate function, the task space coordinates can be viewed as a foliation of the joint space manifold. Then, integrable vector field distributions do exist such that the integral leads to the desired coordinate function for the self-motion manifold. However, neither the self-motion coordinates, nor the distribution to integrate is known in advance. Consequently, finding the self-motion coordinate function can not be achieved by an integration algorithm on vector fields.

Requirements on the relation between the self-motion coordinates and the task space coordinates are derived. Particularly, the Jacobians of both coordinate functions must satisfy certain conditions in order to achieve the dynamical decoupling. The rows of the Jacobian of the desired coordinate function must be mutually orthogonal to the rows of the task space Jacobian. This corresponds to finding foliations, which are orthogonal to the foliation induced by the task space coordinates.

In order to be dynamically consistent, this orthogonality must hold under a particular metric. Choosing the inverse mass matrix as metric enforces that no accelerations in the other coordinates are generated.

Given the requirements on the Jacobian of the desired function still does not allow to determine the vector fields spanned by the Jacobian analytically. The amount of requirements is not enough to determine the vector fields and the system is underdetermined. The desired coordinate function can be interpreted as the solution to an underdetermined system of partial differential equations. Well known PDE solving techniques like finite elements only work for fully determined systems. In consequence, classical PDE solvers are not directly applicable.

An approach comparable to a variational principle is developed in this thesis. It is assumed that the desired coordinate function can be written as a parametric function. Then, the parameters of that function are optimized such that the derived requirements on its Jacobian are fulfilled as well as possible. In particular, a two-layer neural network is used as parametric model. Neural networks are universal function approximators and therefore promise to be able to adapt to the desired coordinate function. For optimization, a cost function based on the derived

requirements is designed. Another advantage of neural networks is the large amount of available tools and software for training and implementation. TensorFlow is used for efficient and fast evaluation and training of the network as well as for automatic differentiation.

1.4. Contributions and Outline

Coordinate functions that specify the remaining degrees of a robotic manipulator are desirable. Especially, when the coordinate functions are designed in a way that they do not interfere with the task space dynamics, controllers in both spaces can operate without interfering with each other. Those coordinate functions parametrize the foliation induced by the entirety of self-motion manifolds of the task space forward kinematics. The decoupling of the task space coordinates is achieved, when foliations that are orthogonal to the self-motion manifolds in the sense of a specified metric tensor are considered. These will be called orthogonal self-motion foliations (OSMFs). Coordinates on these foliation will be called orthogonal self-motion coordinates (OSMCs).

In this thesis a neural network based method to find such OSMC is developed. The main contributions of the thesis are:

- Providing a foliation view on classical task space forward kinematics. A forward kinematics function can be interpreted as imposing a foliation of the joint space manifold.
- Providing an argument that these coordinate functions exist locally. This is a direct consequence of the foliation view on forward kinematics by applying Frobenius' theorem.
- Development of an algorithm to train a neural network approximation of OSMF. The method parametrizes the foliation by explicitly approximating coordinates in terms of OSMCs on it.
- Concept of a controller scheme using OSMCs to simultaneously control the task space dynamics and the remaining degrees of freedom in a decoupled fashion.
- Experimental validation of closed-loop dynamics using the new control scheme on simulated models of planar manipulators.

Chapter 2 shows an overview of related work in the field of robotics and compares them to the method developed in this thesis. In Chapter 3 a more systematic introduction to differential geometry in robotics required for the later chapters is provided. After this introductory part of the thesis, Chapter 4 shows the concept and rationale on the existence OMSFs. Additionally, the requirements on coordinates (OMSCs) on the foliations and the limitations in terms of (non)-globality of the coordinates are explained. Then, the chapter finishes by showing a control scheme using a coordinate function for the remaining degrees of freedom.

Chapter 4 only establishes the requirements on OMSCs, but no approach to find such a function was provided. Subsequently, Chapter 5 provides a concrete algorithm to find a solution to the requirements in the modeling chapter. The coordinate function is written as neural network and a training algorithm is developed, which aims to update the model parameters such that they fulfill the requirements. For the training, a cost function and a training data sampling strategy are required, which are developed in Chapter 5. After the description of the neural-network, Chapter 6 shows results of various trained models for different planar manipulators. The evaluation is performed on a geometric, kinematic and dynamic level. In low dimensional spaces the models can be visualized as curves and surfaces in the plane or volume. Orthogonality can be visually observed and validated. After that, one example model is evaluated on a kinematic

level using a first order differential equation in closed-loop. Then, another example model is used for dynamic evaluation. The model is used for an impedance controller, which is applied to a simulated multi-body dynamics equation of a manipulator. Chapter 7 compares the proposed method to the state of the art approach and discuses advantages and disadvantages. Finally, Chapter 8 concludes the thesis by summarizing the content and providing an outlook to future work. Throughout the thesis, some links to animations and videos are provided in the footnotes. It is recommended to follow the links for an intuitive understanding of the statements, results and concepts explained.

2
Related Work

In this thesis a novel approach for redundancy resolution is proposed. Several techniques for redundancy resolution exist. The subsequent section summarizes available techniques for redundancy resolution and compare the novel approach of this thesis to the latter. A neural network will be used for the novel approach. To our knowledge there is no other publication using machine learning techniques for the given problem. However, several publications deal with learning of dynamic and kinematics models, which are summarized in section 2.2.

2.1. Redundancy Resolution Techniques

Redundancy resolution is a broadly discussed research topic in the literature. The tutorial by Siciliano [80] groups the approaches into four different categories:

1. Simple Jacobian-based Techniques
2. Gradient Projection Method
3. Task Space Augmentation
4. Inverse Kinematic Functions

which are briefly explained below.

2.1.1. Simple Jacobian-based Techniques

Already in the 1960s articles dealing with this topic were published. In early publications the focus is mostly on a kinematic level only, i.e. the dynamic model of a robot is not taken into account for those studies. Initially, the usage of the Moore-Penrose inverse of the Jacobian for redundancy resolution was proposed by Whitney in 1969 [91]. Using a pseudo-inverse, desired task velocities can be transformed to joint velocities. For redundant robots this mapping is not unique, for a given rectangular matrix infinitely many pseudo-inverses exist. The Moore-Penrose pseudo-inverse is a special choice for an identity-weighted case. This minimizes the Euclidean norm of the resulting joint velocities. Another common weighing matrix is the inverse mass matrix of the robot, which minimizes the kinetic energy of the manipulator. Also proposals on minimization of the joint torques for redundancy resolution have been proposed [29].

The redundancy resolution techniques based on the pseudo-inverse also have some drawbacks. As Klein and Huang [39] state it, the approach can drive the system to unpredictable configurations [54]. This is a direct consequence of the non-integrability of the Moore-Penrose pseudo-inverse. They provide a very intuitive visualization of this in their review [39], which is reproduced in Figure 2.1. A robot with three degrees of freedom is following a square multiple times, where the

robot trajectory is computed on a kinematic level based on the Moore-Penrose pseudo-inverse of the Jacobian. Each time the manipulator passes by the lower-left corner of the square, the configuration is plotted.

Obviously, the configurations at the lower-left corner is different after tracing out the square. As the Moore-Penrose pseudo-inverse is non-integrable, a round trip in the corresponding vector fields does not lead back to the initial state. While tracing out the square, the configurations may or may not converge, reach joint limits or come arbitrary close to singularities [39]. Therefore, the behavior of the system can not be considered deterministic.

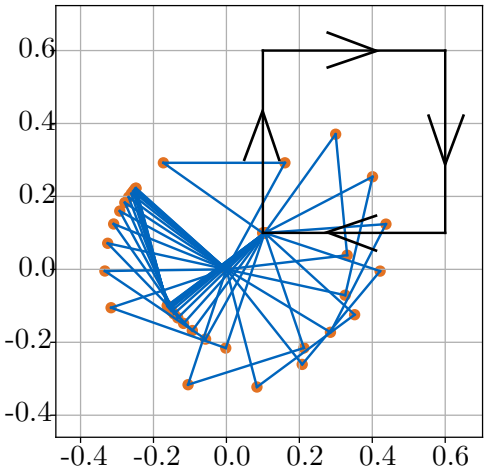


Figure 2.1.: Non-Integrability of the Moore-Penrose pseudo-inverse visualized. The robot configuration is shown after each full cycle of following the square shape with the end-effector. The Moore-Penrose inverse is used for computation of joint velocities. The non-integrability results in different configurations after the full cycles. Animation of the robot here: http://thesis.aaarne.de/square-tracing.gif

Mussa-Ivaldi and Hogan showed that integrable pseudo-inverses exist on simply-connected and singularity-free regions in joint space [54]. For special weighing matrices, the pseudo-inverse becomes integrable. In their approach, they take a function and show that its derivative can be written as a weighted pseudo-inverse. The particular weighing matrix is not constant, but rather a position-dependent function.

2.1.2. Gradient Projection Method

The gradient projection is an extension of simple Jacobian-based techniques. All of the pseudo-inverse based approaches for redundancy resolution allow for direct application of null space control. The projection matrix required for null space control can be straightforwardly computed given the pseudo-inverse. Then all of the projection-based control schemes already mentioned in the introduction can be applied. These, for instance, include obstacle avoidance [36], self-collision avoidance (e.g. [15]), joint limit avoidance [31], singularity avoidance (e.g. [25]) and modification of dynamic properties [85, 89, 49, 69, 20].

Non-integrability may lead to non-deterministic behavior as could be observed in Figure 2.1. Following the square might be possible for only a limited amount of iterations. Without intervention, the robot may at some point run into joint limits or singularities. This can be prevented by adding another controller for joint-limit and singularity avoidance acting in the null space. Basically, using the additional null space controllers the non-deterministic behavior can be forced into a desirable direction. In particular, the robot can be kept manipulable by

avoiding unfavorable configurations and collisions with obstacles.

Gradient projection methods also allow to specify more than one task in a hierarchy. Higher priority tasks will never be disturbed by lower priority tasks. In [13] an overview of projection matrices is provided. A recent publication by Mansfeld et al. also shows techniques for temporarily relaxing the hierarchy [50]. Using this technique, the main task may temporarily be disturbed by the secondary task such that the secondary task can achieve the control goal.

The projection matrix is computed based on a pseudo-inverse as well. Therefore, the same limitations apply. The projection method does not ensure integrability either. Additionally, the projection can result in local minima when evaluating the closed-loop behavior with controllers on the lower priority tasks. These can lead to suddenly discharging virtual springs upon small variation of the robot configuration.

2.1.3. Task Space Augmentation

All of the aforementioned approaches are based on pseudo-inverses of the task Jacobian and have one thing in common: they are inherently local. Task space augmentation techniques augment the task coordinates by additional so-called constraint tasks, that employ at most all redundant degrees of freedom [80]. The constraint tasks have a Jacobian with respect to the joint angles. As many constraint tasks are added as needed to form a square-shaped Jacobian matrix for the augmented task space. Then, the inverse Jacobian can be used for redundancy resolution. However, it is not easy to find a constraint tasks such that a manipulator can achieve both tasks exactly. Then, least squares techniques can be used and the gradient projection method is used for least squares solution of the constraint tasks. Also, there is no methodology for finding such constraint tasks. They are hand-designed for each use case and task.

Egeland used a task space augmentation method for a macro-micro manipulator [17] and in [74] the approach is used for narrow-passage passing of a highly redundant robot.

2.1.4. Global Inverse Kinematic Functions

Another approach is to use global inverse kinematic functions for redundancy resolution [90]. The inverse kinematics function maps a task space pose to joint angles. Once an inverse kinematics function exists, one task pose always maps to the same joint angles, so no redundancy resolution is required. Basically, the manipulator behaves non-redundantly when viewed through the global inverse kinematic function. Of course, infinitely many of those functions can exist. For redundant manipulators the forward kinematics pre-image are always infinitely many configurations. If a global inverse kinematic function is selected, only one of those is taken. This would be very restrictive as the redundancy can not be used. Hence, the global inverse kinematic function can be written as parametric functions. Some parameters allow to select which of the infinitely many pre-images the global inverse kinematics function will select.

Wampler [90] restricted the work space of a redundant manipulator to a region, where a global inverse kinematic function exists. Shimizu et al. showed a method for a 7-DoF robot in [78]. They use the angle between the plane spanned by the elbow angle and a reference angle as a parameter to the global inverse kinematic function. Also by fixing some of the joint angles a global inverse kinematics function can be obtained. This type of redundancy resolution is implemented on the KUKA LBR iiwa. The angle of the third axis is fixed for the global inverse kinematic function. Then, there are only countably many solutions left. For instance, also when fixing one of the joints, the robot still has elbow-up and elbow-down configurations. Additionally, fixing the signs of some angles yields a global inverse kinematics function [76].

The programming API also provides the redundancy resolution based on the angle of the elbow plane.

2.1.5. Classification of the Novel Approach

In this thesis a method is developed that finds a coordinate function for the remaining the degrees of freedom. Therefore, it is a task space augmentation method. However, in contrast to existing approaches, the idea is to design dynamically (and kinematically) decoupled coordinates. The Jacobian of the new coordinates must be in the null space of the task Jacobian. In other words, applying the projection would have no effect, because by construction the coordinates have no effect on the task space.

2.2. Learning of Kinematic and Dynamic Models

The approach in this thesis uses a machine learning technique in order to find the self-motion coordinates. To my best knowledge there is so far no publication showing this approach. However, for many other use cases in the control field of robotics, machine learning techniques are widely researched and many results have been published.

The dynamics of the robot is usually modeled as a differential equation arising from multi-body dynamics known in mechanics. Therefore, for this model to be valid the entire robot has to be known. This includes the kinematics as well as inertial properties. Given the structure of the differential equation, standard system identification techniques (e.g. [44]) can be used on order to fit the unknown parameters of the robot. Many physical phenomena of the actual robot usually remain unmodelled. This includes joint friction, delays, unmodelled nonlinearities and the dynamics of the drives [71]. Machine learning techniques allow to learn the dynamics model without the strict structure enforced by the multi-body dynamics model. Therefore, a machine learning technique can potentially also learn the features, which are not modelled in classical multi-body dynamics. Some publications show machine learning based system identification of forward dynamic models. This can be achieved using classical regression techniques. Consider [84] for a comparative overview on regression. For instance, classical neural networks [57] and radial basis function networks [86] have been used to perform regression of a forward model.

In contrast to the forward dynamic and kinematic models, the inverses generally have no closed-form analytic solution. Therefore, the motivation to learn inverse models is significantly larger than learning forward models. This can be observed also in the amount of publications, the majority of articles focus on learning of inverse models. D'Souca et al. [9] use locally weighted projection regression (LWPR) [88] in order to learn inverse kinematics of a redundant manipulator. Especially, learning of inverse dynamic models is researched extensively. Inverse dynamic models can be directly used as feed-forward control input of the robot [55]. LWPR-based techniques are often used for inverse dynamics models. Schaal et al. [72] used learned LWPR-based models that were trained for specific tasks like devil sticking and pole balancing. They also showed that it is possible to learn the inverse dynamic model of manipulators using LWPR. In [64] the inverse dynamics learning is extended to an operational space framework for redundant manipulators. Holonomic mobile robots are usually equipped with omniwheels in order to be able to maneuver in every direction. Omniwheels provide flexibility, but are also inherently hard to model as the resulting forces are highly dependent on ground friction. In [59] different models for adaptive feed-forward control of mobile robots with omniwheels are compared. Also Gaussian processes [65] are frequently used for inverse dynamics regression. For instance, Shon et al. [79] use Gaussian process regression to perform imitation learning

of motions captured from humans. In [56] use local Gaussian processes for real-time online learning of inverse dynamics. Furthermore, Calinon et al. [7] show results of Gaussian mixture models (GMM) on humanoid robots.

On the one hand, learned inverse models can be directly fed to the robot and the output can be used as feed-forward control signal, on the other hand, the model is almost certainly not perfect and errors will occur. Additionally, many quantities can be computed analytically, which is faster and exact. Salaün et al. [71] combine learned models with the analytical operational space framework for control of redundant manipulators

3

Differential Geometry in Robotics

This chapter introduces and wraps up fundamentals of robot kinematics, dynamics and control presented in a differential geometric perspective. Fundamental equations and concepts needed to derive the orthogonal foliation control approach are presented here.

Generally, a robotic manipulator is a set of links connected by joints. Joints connect adjacent links in order to form a kinematic structure. In this thesis only serial structures without kinematic loops or parallelisms are considered. Figure 3.1 shows a schematic drawing of such a serial interconnection of links. Joints occur in a variety of appearances, prismatic and rotational joints being most common examples of those. The configuration of a robot can be uniquely parametrized given the values of all joint parameters, where the required amount of parameters per joint corresponds to that joint's degrees of freedom [53, Chapter 3].

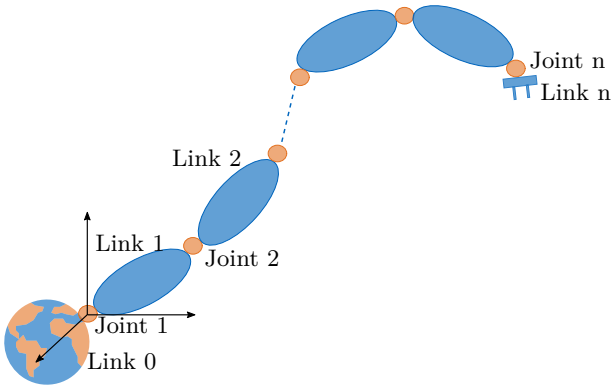


Figure 3.1.: A robotic manipulator is a set of links connected by joints. Figure adapted from [35, Section 1.2]

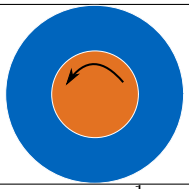
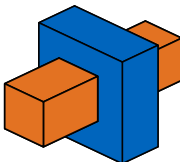
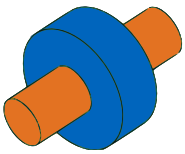
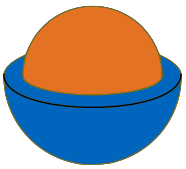
Tasks are usually specified in Cartesian coordinates. Robot configurations, however, are described in terms of joint configurations like joint angles for example. In order to perform tasks with the manipulator, a relationship between joint- and task space is required. Therefore, a coordinate function, which maps from joint- to task space will be defined. Considering only static configurations of the manipulator, the study of that relationship is known as kinematics. The subsequent section shows concepts from kinematics and instantaneous kinematics required to understand how positions, velocities and forces transform between those spaces. Additionally, the terms non-redundant and redundant manipulators will be defined. Afterwards, interpreting the interconnection of links and joints as a multi-body system, the dynamics of the manipulator is formulated using multi-body dynamics. As for the static cases, also the manipulator dynamics can be transformed between spaces. Given the dynamics of the manipulator, control schemes will be presented. The focus is on the impedance controller scheme, which simulates a virtual

spring-damper system in respective spaces. It will be shown that task space impedance control leads to left-over (uncontrolled) dynamics of redundant manipulators. Finally, the terms vector field, integrability and foliation will be introduced. The chapter concludes stating the important Frobenius' theorem on integrability, a necessary and sufficient condition for existence of solutions to underdetermined systems of partial differential equations.

3.1. Coordinates, Kinematics and Jacobians

Links of a manipulator are connected by joints, which come in a variety of types. Joint types can be characterized according to their configuration space. Table 3.1 summarizes a selection of joint types [45]. The joints connect two links, which are depicted in blue and orange, respectively. Dependent on the joint type, the relative position between the two connected links is parametrized by different parameters. Additionally, different types may also leave more or less degrees of freedom. The space of all parameters required to describe the configuration of the joint is called configuration space of the joint. For instance, the revolute joint in Table 3.1 needs one such parameter, namely the angle between the two links. Because the joint configuration of the revolute joint is specified by an angle, the topology of the configuration space is the one-sphere $\mathbb{S}^1$. On the contrary, the state of the prismatic joint is the shift between the two links and the topology of the configuration space is simply the Euclidean space of the real numbers $\mathbb{R}$. Based on this, the degrees of freedom of one joint are defined as the dimensionality of its configuration space. The spherical joint has three degrees of freedom, because the topology of the configuration space is a three-sphere. From now on, only joints with one degree of freedom will be considered. Note that joints with more than one degree of freedom can be transformed to multiple single degree of freedom joints by adding virtual links.

Table 3.1.: Example joint types. Different joint types have different configuration spaces with different topologies. The dimensionality of the configuration space is called the degrees of freedom. Drawings are inspired by [41, Chapter 3].

	Revolute	**Prismatic**	**Cylindrical**	**Spherical**
Drawing				
Configuration Space	$\mathcal{Q}_r = \mathbb{S}^1$	$\mathcal{Q}_p = \mathbb{R}$	$\mathcal{Q}_c = \mathbb{R} \times \mathbb{S}^1$	$\mathcal{Q}_s = \mathbb{S}^3$
Degrees of Freedom	$\dim \mathcal{Q}_r = 1$	$\dim \mathcal{Q}_p = 1$	$\dim \mathcal{Q}_c = 2$	$\dim \mathcal{Q}_s = 3$

Manipulators consist of multiple joints. The configuration space of the manipulator $\mathcal{Q}$ is simply the Cartesian product of all configuration spaces of the joints $\mathcal{Q}_i$ [53, Chapter 2]

$$\mathcal{Q} = \bigtimes_{i=1}^{n} \mathcal{Q}_i \,. \tag{3.1}$$

The configuration space of a manipulator is also referred to as the manipulator's *joint space*. This term will be used from now on. Most robotic manipulators consist of rotational joints only, where each joint i has a one-sphere topology. The product of n one-spheres is an n-dimensional hypertorus denoted

$$\mathbb{T}^n = \bigtimes_{i=1}^{n} \mathbb{S}^1 \,. \tag{3.2}$$

It is important to notice that the Cartesian product of two one-spheres $\mathbb{S}^1$ is not the two-sphere $\mathbb{S}^2$, but a torus $\mathbb{T}^2$. Toroidal topologies are easier to handle, especially sampling schemes using uniform distributions are easy on toroidal topologies and complicated on spheres. Therefore, usual joint spaces are hypertori. If the robot also has linear actuators and prismatic joints the overall joint space will be $\mathcal{Q} = \mathbb{T}^r \times \mathbb{R}^r$ for r rotational and p prismatic joints [53].

3.1.1. Task Space

When performing tasks with a robotic manipulator, the specification of motions, forces and desired dynamic properties is usually not described in the joint space directly. Rather, robotic tasks are usually specified in a Cartesian coordinate system. Let the task space $\mathcal{M}$ of a manipulator be the space where tasks are specified. In most of the cases, this will be the Cartesian pose of the end-effector [45, Chapter 4]. Therefore, in most of the cases the task space will be the Special Euclidean Group $\mathcal{M} = SE(n)$ with $n = 2$ for planar and $n = 3$ for non-planar robots. The Special Euclidean Groups specify rigid body transformations in Euclidean spaces. Basically, elements of these groups describe poses (position and orientation) of objects in $\mathbb{R}^2$ or $\mathbb{R}^3$, respectively. [53, Chapter 3] is a reference for further information. Note that this choice is not complete, a task might be any holonomic constraint. For instance, distances from obstacles can also be regarded as task space coordinates, which are not $SE(n)$.

Task spaces are only for pure translational quantities Euclidean. As soon as orientations are incorporated for the choice of the task space coordinates, the task space $\mathcal{M}$ is described by a manifold, which is only locally Euclidean. Also, the special Euclidean groups $SE(n)$ are manifolds embedded in Euclidean space.

Figure 3.2 shows a visualization of this concept with an exemplary robotic manipulator. The robot consists of two rotational joints and has a 2-torus joint space $\mathcal{Q} = \mathbb{T}^2$. Here, the choice for the task space $\mathcal{M}$ is only the translational component of the endeffector position specified by x_1- and x_2-coordinates. In other words, the orientation is irrelevant for the task space coordinate here. For example, this task space choice would be sufficient to draw with a pen on paper with the robot.

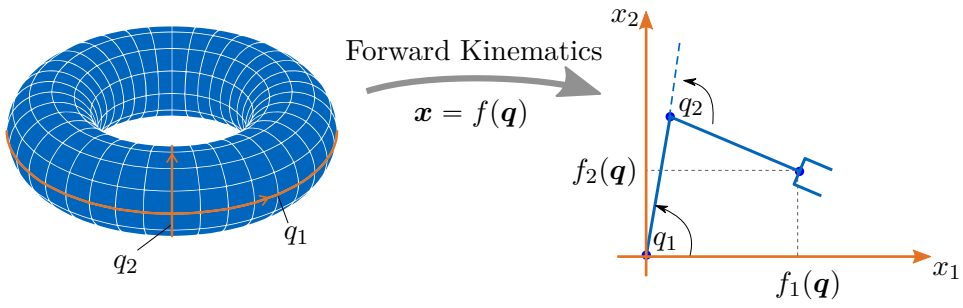


Figure 3.2.: An exemplary planar robot with its configuration- and task space. The joint space represents the space of all possible joint configurations, while tasks and forces are usually specified in task space. The mapping from joint to task space is known as forward kinematics.

3.1.2. Forward Kinematics

In order to perform tasks with the manipulator a relationship between joint- and task space coordinates is required. Therefore, the forward kinematics function is defined [53, Chapter 3].

Definition 1 (Forward Kinematics)**.** *Let* $\boldsymbol{q} \in \mathcal{Q}$ *and* $\boldsymbol{x} \in \mathcal{M}$*, the function* $f : \mathcal{Q} \to \mathcal{M}$

$$\boldsymbol{x} = f(\boldsymbol{q}) \tag{3.3}$$

that maps joint space configurations to task space coordinates is called forward kinematics.

Given a joint configuration $\boldsymbol{q}$ of the robot, the function computes the position $\boldsymbol{x}$ of the robot in task space. Usually, the forward kinematics is derived by chaining rigid-body transformations between the links. Refer to e.g. [8, Chapter 3] for further information on the derivation of the forward kinematics function. The function f is generally not bijective, as multiple joint configurations may result in the same task space coordinates.

For the example robot introduced in Figure 3.2 the forward kinematics function f is plotted in Figure 3.3. Isolines show configuration with constant x_1 and x_2 components, respectively. Additionally, two configurations with the same task space coordinate are shown.

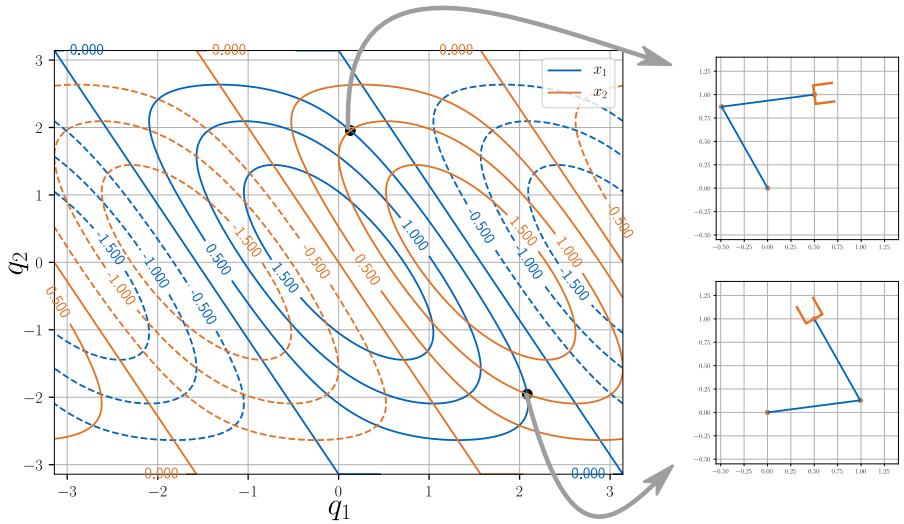


Figure 3.3.: Forward kinematics of the example robot. Each joint configuration is mapped to the position of the endeffector in Cartesian space. Blue lines denote isolines of the x_1 component and orange lines of the x_2 component, respectively. The mapping is not bijective. The black dots denote an example of configuration with different joint configurations, but the same endeffector position.

3.1.3. Tangent Spaces, Vectors, Jacobians

Next, concepts from instantaneous kinematics are introduced, which are required to transform vectors and covectors between the tangent and cotangent spaces of the joint- and task space manifolds $\mathcal{Q}$ and $\mathcal{M}$. Forward kinematics describe only the static and stationary configuration of a manipulator. Let again $\mathcal{M}$ be the task space manifold, the tangent space of $\mathcal{M}$ at the point $\boldsymbol{x}$ is denoted $\mathcal{T}_x\mathcal{M}$ [32, Section 1.2]. The tangent space is dependent on the location $\boldsymbol{x} \in \mathcal{M}$ on the manifold (Figure 3.4). In contrast to the task space manifold, its tangent space is a vector space and all vector space properties apply. The elements of the tangent space $\mathcal{T}_x\mathcal{M}$ are called *vectors*.

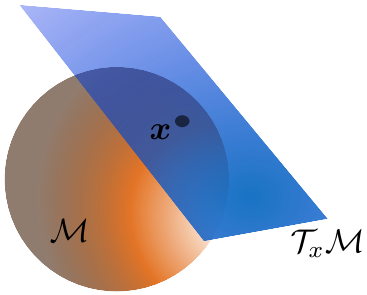


Figure 3.4.: A manifold $\mathcal{M}$ and its tangent space $\mathcal{T}_x\mathcal{M}$ at $\boldsymbol{x} \in \mathcal{M}$. If $\mathcal{M}$ represents the task space manifold the columns of the forward kinematics Jacobian span the tangent space under the condition that the configuration is non-singular.

Further, the definition of tangent bundles is required in order to analyze integrability in the later section. The tangent space $\mathcal{T}_x\mathcal{M}$ is a local vector space tangent to the manifold $\mathcal{M}$ at the point $\boldsymbol{x} \in \mathcal{M}$. According to [58, Chapter 2] the tangent bundle is defined as

Definition 2 (Tangent Bundle)**.** *The union of all tangent spaces for all $\boldsymbol{x} \in \mathcal{M}$ is called the tangent bundle $\mathcal{TM}$ of the manifold $\mathcal{M}$*

$$\mathcal{TM} = \bigcup_{\boldsymbol{x}\in\mathcal{M}} \mathcal{T}_x\mathcal{M}\,. \tag{3.4}$$

Given the vectorial tangent spaces, vector quantities such as velocities for a robotic manipulator can be defined. Velocities are vectors and live in the tangent spaces $\mathcal{T}_x\mathcal{M}$ of the manifold. For the choice of Cartesian task spaces $SE(2)$ and $SE(3)$ the tangent spaces at $\boldsymbol{x}$ are $\mathcal{T}_xSE(2) = \mathbb{R}^3$ and $\mathcal{T}_xSE(3) = \mathbb{R}^6$, respectively. The term twist is defined for that case [19]

Definition 3 (Twist)**.** *The elements of the tangent spaces $\mathcal{T}_xSE(3)$ of the special Euclidean group $SE(3)$ are called* twists*. They describe instantaneous screw motions.*

Usually the task space manifold is $SE(3)$. Consequently, often the term twist is used when task space velocities are considered.

Joint- and task space are both manifolds with tangent spaces each. Consider a velocity in joint space $\dot{\boldsymbol{q}} \in \mathcal{T}_q\mathcal{Q}$. For a manipulator consisting of rotational joints only, this would be a vector of angular rates for each joint. Next, the question arises how the effect of this velocity is in task space. In other words, how does the joint space vector transform to a task space velocity $\dot{\boldsymbol{x}} \in \mathcal{T}_x\mathcal{M}$? The answer is the Jacobian matrix of the manipulator, which is obtained by differentiating the forward kinematics $\boldsymbol{x} = f(\boldsymbol{q})$ [75, Section 4.5]:

Definition 4 (Jacobian Matrix)**.** *The matrix of all partial derivatives of the forwards kinematics function $f : \mathcal{Q} \to \mathcal{M}$ is called Jacobian matrix*

$$\boldsymbol{J}(\boldsymbol{q}) = \left[\frac{\partial f_i}{\partial q_j}\right] = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial q_1} & \cdots & \frac{\partial f_m}{\partial q_n} \end{bmatrix}, \tag{3.5}$$

where n denotes the number of joints and $\dim(\mathcal{T}_x\mathcal{M}) = m$.

The Jacobian $\boldsymbol{J} : \mathcal{Q} \to \mathbb{R}^{m \times n}$ is dependent on the joint configuration $\boldsymbol{q}$. The Jacobian of a differentiable mapping (as the forward kinematics function is) transforms vectors from the tangent space of the input manifold to the tangent space of the output manifold [21, Chapter 2]. In plain differential geometry, the concept of a Jacobian is called a *push-forward* associated with f [42, Chapter 3], because the push vectors in $\mathcal{T}_q\mathcal{Q}$ forward to vectors in $\mathcal{T}_x\mathcal{X}$. This can be directly applied to robotics. Velocities are vectors and are therefore transformed according to a linear transformation with the Jacobian matrix

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}\,. \tag{3.6}$$

Consider Figure 3.5 for a schematic viewing of this concept. Because vectors transform according to this rule, they are said to be transformed contravariantly [42, Chapter 4]. While the forward kinematics $\boldsymbol{x} = f(\boldsymbol{q})$ is generally nonlinear, the transformation between velocities and other vector quantities is a *linear* transformation with $\boldsymbol{J}(\boldsymbol{q})$. Note however, that the Jacobian itself depends, potentially nonlinearly, on the joint configuration $\boldsymbol{q}$. Provided that the Jacobian is a square matrix and is non-singular, the transformation between and task space velocities can also be inverted

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1}(\boldsymbol{q})\dot{\boldsymbol{x}}\,. \tag{3.7}$$

This is an important property of the linear transformation that is heavily used for iterative inverse kinematics or the generation of desired task space velocities.

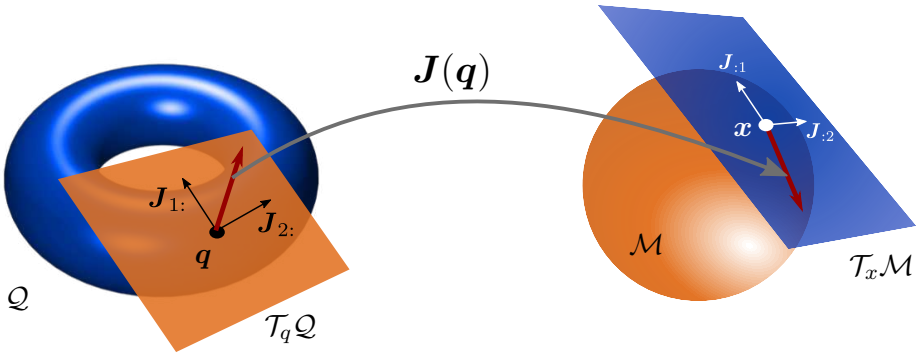


Figure 3.5.: The Jacobian matrix of a forward kinematics function maps vectors between the tangent spaces of the manifolds. While the rows of the Jacobian span a subspace of the tangent space of the joint space manifold, the columns span the tangent space of the task space provided that the Jacobian is nonsingular.

3.1.4. Cotangent Spaces, Covectors, Wrenches

Recall the definition of dual spaces known in linear algebra

Definition 5 (Dual Space of a Vector Space)**.** *Let V be a vector space. The space of all linear functionals $\eta : V \to \mathbb{R}$ is called dual space of V and denoted V^*. Its elements $\eta \in V^*$ are called covectors. The application of a covector $\eta \in V^*$ to a vector $v \in V$ is called* duality pairing *and is also denoted $\langle \cdot, \cdot \rangle : V^* \times V \to \mathbb{R}$.*

Tangent spaces $\mathcal{T}_x\mathcal{M}$ of a manifold $\mathcal{M}$ are vector spaces and therefore the definition of dual spaces can be applied to those as well. In differential geometry, the dual spaces of tangent spaces are called cotangent spaces and defined as [42, Chapter 4]

Definition 6 (Cotangent Space)**.** *Let $\mathcal{M}$ be a smooth manifold. The space of all linear functionals $\eta : \mathcal{T}_x\mathcal{M} \to \mathbb{R}$ at a point $x \in M$ is called* cotangent space *at x and is denoted $\mathcal{T}_x^*\mathcal{M}$. The cotangent space $\mathcal{T}_x^*\mathcal{M}$ is dual to the tangent space $\mathcal{T}_x\mathcal{M}$. Its elements are called* tangent covectors*, or simply* covectors *at x.*

For the robotic manipulator, covectors represent forces and torques. For example, for a robot with only rotational joints, a covector would consist of all joint torques at the individual joints. On the other hand, for a choice of Cartesian task coordinates, a task space covector could be translational forces along and torques about the principal axes. As for velocities, if the task space choice is $SE(3)$, covectors have special properties and a special term is used [19]

Definition 7 (Wrench)**.** *The elements of the cotangent space $\mathcal{T}_x^*SE(3)$ are called* wrenches*.*

While twists live in the tangent spaces of $SE(3)$ and represent velocities based on instantaneous screws, the analogous covectors in the cotangent spaces of $SE(3)$ are called wrenches and represent forces by virtually applying torques to screws with specific pitches.

For scalar quantities, the product of a force and the velocity is a power. The same is true for multiple dimensions, where the duality pairing of a force covector and a velocity vector computes a scalar power. Let $\boldsymbol{\tau} \in \mathcal{T}_q^*\mathcal{Q}$ be a joint space force and $\boldsymbol{f} \in \mathcal{T}_x^*\mathcal{M}$ a task space force. Then, $\boldsymbol{\tau}^\top \dot{\boldsymbol{q}} = P_{joint}$ computes the power exerted on or by the robotic manipulator. Similarly, $\boldsymbol{f}^\top \dot{\boldsymbol{x}} = P_{task}$ also computes the power. By the principle of conservation of power it is known that $P_{joint} = P_{task}$. Using the principle of conservation of power, the transformation of covectors can be derived

$$\begin{aligned} P_{joint} = \boldsymbol{\tau}^\top \dot{\boldsymbol{q}} &= \boldsymbol{f}^\top \dot{\boldsymbol{x}} = P_{task} \\ \boldsymbol{\tau}^\top \dot{\boldsymbol{q}} &= \boldsymbol{f}^\top \boldsymbol{J}(\boldsymbol{q}) \dot{\boldsymbol{q}} \\ \boldsymbol{\tau} &= \boldsymbol{J}^\top(\boldsymbol{q}) \boldsymbol{f} \end{aligned} \tag{3.8}$$

While $\boldsymbol{J}(\boldsymbol{q})$ maps vectors from joint- to task space, $\boldsymbol{J}^\top(\boldsymbol{q})$ transforms covectors in the opposite direction. Therefore, this mapping is also called a *pullback* in differential geometry [42, Chp. 4].

This type of transformation is called covariant transformation [21]. Again, if the Jacobian $\boldsymbol{J}(\boldsymbol{q})$ is a square matrix and non-singular, the transformation can be inverted in order to compute task space forces

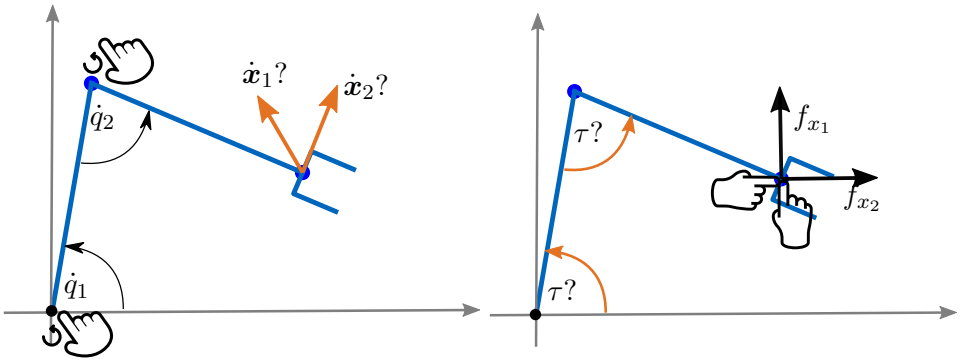
$$\boldsymbol{f} = \boldsymbol{J}^{-\top}(\boldsymbol{q}) \boldsymbol{\tau} \,. \tag{3.9}$$

The vector and covector transformation is visualized for the planar example manipulator in Figure 3.6. While the i-th column of $\boldsymbol{J}(\boldsymbol{q})$ represents the velocity of the endeffector when rotating the i-th joint at unit speed, the j-th row of $\boldsymbol{J}(\boldsymbol{q})$ represents the joint torques when pushing with a unit force in the j-th principal direction of the endeffector coordinates.

3.1.5. Singularities

In order to compute the required joint space velocity to generate a desired task space velocity $\dot{\boldsymbol{x}}$, the inverse of the Jacobian $\boldsymbol{J}^{-1}(\boldsymbol{q})$ is required. At singular configurations, this inversion is not possible, because the Jacobian matrix becomes singular. The manipulator is said to loose degrees of freedom with respect to the task space in singular configurations.

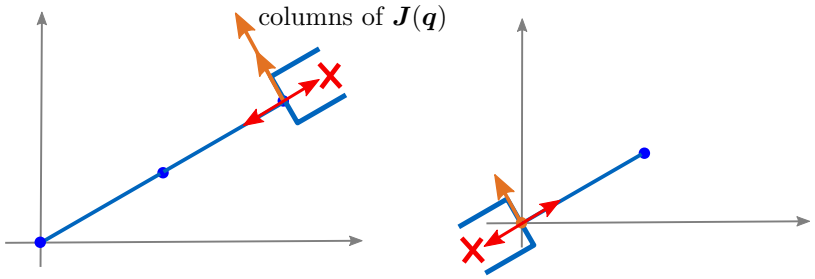
Singularities arise when columns of the Jacobian become collinear or vanish entirely. Figure 3.7 shows two example configurations, which are singular with respect to the position of the end effector. At the first configuration, the columns of the Jacobian are linearly dependent. Therefore,



(a) Vector mapping of $\boldsymbol{J}(\boldsymbol{q})$ (b) Covector mapping of $\boldsymbol{J}^\top(\boldsymbol{q})$

Figure 3.6.: Example of the usage of the Jacobian matrix to transform vectors and covectors between joint- and task space. (a) Mapping of velocities. Rotating the joint at unit speed, the columns of $\boldsymbol{J}(\boldsymbol{q})$ specify the velocity vector in task space for the respective joint. (b) Mapping of forces. Pushing the endeffector with a unit force into the principal direction of the task space, the rows of $\boldsymbol{J}(\boldsymbol{q})$ compute the torques on the joints.

the Jacobian only spans a one-dimensional vector space in this configuration. Components of task space velocities corresponding to the direction denoted in red can not be generated. Similarly, task space forces acting in that direction do not generate any torque $\boldsymbol{\tau}$. At the second singular configuration shown in (b), one of the rows even vanishes.



(a) Configuration 1 (b) Configuration 2

Figure 3.7.: Singular configuration of an exemplary planar robot with two degrees of freedom.

For now, it is assumed that the number of task space coordinates matches the degrees of freedom of the manipulator. The next section generalizes this to redundant manipulators. Formally, singularities can be defined in terms of the determinant of the Jacobian

Definition 8 (Singularity)**.** *A joint configuration $\boldsymbol{q} \in \mathcal{Q}$ is called singular with respect to the task space coordinates, if*

$$\det \boldsymbol{J}(\boldsymbol{q}) = 0\,. \tag{3.10}$$

3.2. Kinematic Redundancy

Kinematic redundancy of a manipulator depends on the number and interconnection of joints as well as on the choice of the task space coordinates. From now on the task space coordinates are assumed to be minimal, meaning that there are no dependencies. For example a coordinate function computing the x-component and half the x-component of the endeffector pose of a manipulator would not be considered minimal as it computes only one independent quantity. Then, the term kinematic redundancy can be defined [34].

Definition 9 (Kinematic Redundancy)**.** *A manipulator is kinematically redundant under a choice of task space coordinates, when there are more degrees of freedom in the joint space n than there are dimensions m in the task space coordinates*

$$n = \dim \mathcal{Q} > \dim \mathcal{M} = m\,. \tag{3.11}$$

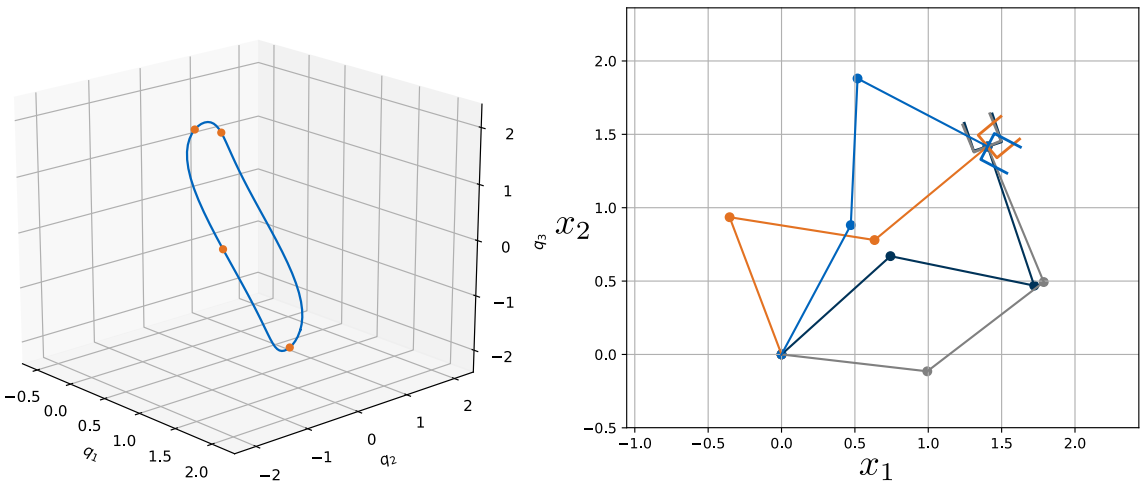
The difference $r = n - m$ is called the degree of redundancy of the manipulator.

3.2.1. Self-Motion Manifolds

For non-redundant robots where the degrees of freedom matches the dimensionality of the task space coordinates, isolated points in joint space may map to the same task space coordinates. This was shown in Figure 3.3 for the exemplary planar robot. For redundant manipulators however, joint space configuration that map to the same task space coordinates lie on a continuum. These continua form one or more r-dimensional manifolds in the joint space. Because moving in these manifolds will change the robot configuration without changing the task space coordinates, these manifolds are called *self-motion manifolds*. Formally [6]:

Definition 10 (Self-motion Manifold)**.** *Each of the disjoint r-dimensional manifolds in the inverse kinematic preimage will be termed a* self-motion manifold.

For example, when another rotational joint is added to the example planar manipulator, while keeping the choice of using the x_1 and x_2-component of the position of the endeffector as task space coordinates, the manipulator becomes kinematically redundant. There are $n = 3$ degrees of freedom and $m = \dim \mathbb{R}^2 = 2$ independent task space coordinates. The degree of redundancy is $r = n - m = 1$. Figure 3.8 (left) shows the one-dimensional self-motion manifold for the planar manipulator with three degrees of freedom. On the right, four robot configurations corresponding to four sampled points on the self-motion manifold are shown. Similarly, when choosing only a one-dimensional forward kinematics function $m = \dim \mathcal{M} = 1$, the degree of redundancy becomes $r = 3 - 1 = 2$ and the self-motion manifold will be two-dimensional. Choosing only the x_1-component of the endeffector position as coordinate function, the self-motion manifold of the planar three degree of freedom looks like the surface in Figure 3.9.



(a) Self-motion manifold

(b) Robot configurations.

Figure 3.8.: Self-motion manifold of a planar robot with three rotational joints and unit-length links. The chosen task space coordinates are the x and y component of the translational position of the end-effector of the robot. The orientation of the end-effector is not considered as task space coordinate. On the left, the self-motion manifold for $x_d = [1.404, 1.417]^\top$ is shown. The blue line represents the one-dimensional self-motion manifold. Four points are selected depicted with an orange dot. Those configurations are shown on the right. Animated robot here: http://thesis.aaarne.de/animated.gif

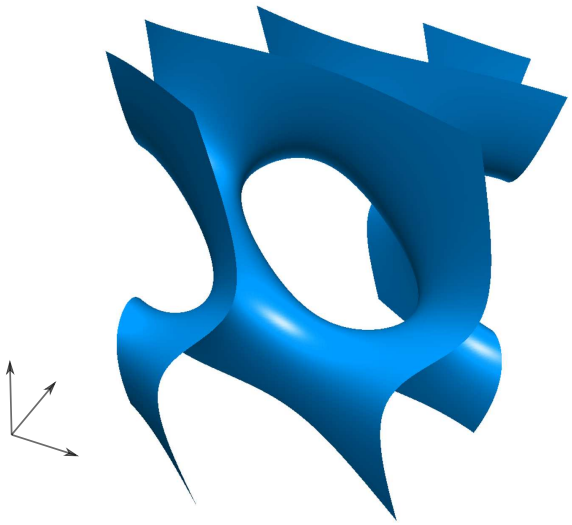


Figure 3.9.: Manifold for $x = 0$ for a single dimensional taskspace of a 3DoF planar robot without joint limits. The surface is created using the marching cubes algorithm described in [43] on the forward kinematics. 3D view of the surface here: http://thesis.aaarne.de/self-motion-manifold.stl

3.2.2. Null Space of the Jacobian Matrix

For redundant manipulators, the Jacobian matrix $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{m \times n}$ is no longer a square matrix. As there are more degrees of freedom than task space coordinates, joint space velocities that do

not generate any task space velocity are possible. These velocity vectors lie in the null space $\ker \boldsymbol{J}(\boldsymbol{q})$ of the Jacobian. All joint velocities $\dot{\boldsymbol{q}}_0 \in \ker \boldsymbol{J}(\boldsymbol{q})$ vanish in task space coordinates and do not result in any task space velocity. Consider Figure 3.10 for a schematic drawing.

For non-redundant cases the null space of the Jacobian $\ker \boldsymbol{J}(\boldsymbol{q})$ is trivial and only contains the origin, provided the Jacobian is nonsingular. At kinematic singularities however, also the Jacobian of non-redundant manipulators locally has a non-trivial null space. In contrast, for kinematically redundant manipulators, the null space $\ker \boldsymbol{J}(\boldsymbol{q})$ is always non-trivial and at least $\dim(\ker \boldsymbol{J}(\boldsymbol{q})) \geqslant r$. Similar for the non-redundant case, the notion of kinematic singularities can be defined for redundant manipulators.

Definition 11 (Singularities of Redundant Manipulators)**.** *Let r be the degree of redundancy of a manipulator. For given kinematics and task space coordinates, a configuration is called non-singular, when* $\dim(\ker \boldsymbol{J}(\boldsymbol{q})) = r$ *and singular when* $\dim(\ker \boldsymbol{J}(\boldsymbol{q})) > r$.

In other words, a configuration $\boldsymbol{q}$ is singular, when the Jacobian $\boldsymbol{J}(\boldsymbol{q})$ is row-rank deficient.

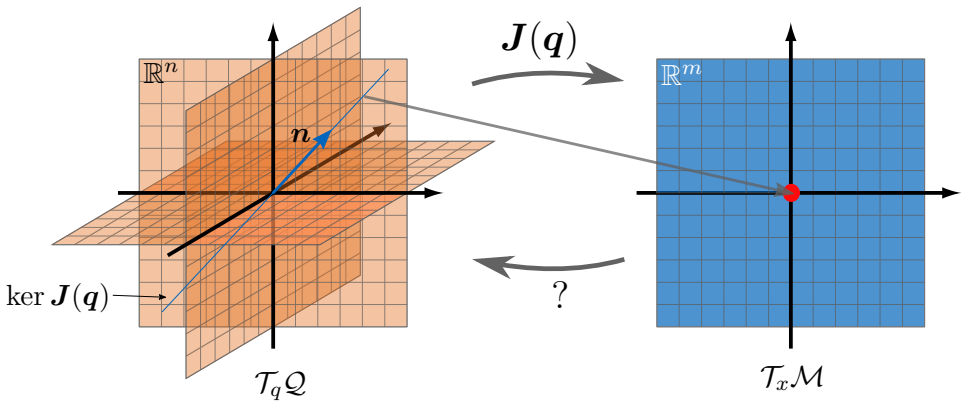


Figure 3.10.: The Jacobian matrix maps from the n-dimensional tangent space $\mathcal{T}_q\mathcal{Q}$ to the m-dimensional task space tangent space $\mathcal{T}_x\mathcal{M}$. For kinematic redundancies $n > m$ the mapping is projective and leaves a nontrivial kernel of the Jacobian matrix. Vectors in $\ker \boldsymbol{J}(\boldsymbol{q})$ are mapped to zero.

While transformation of vectors still works from joint to task space (3.6) the inversion is no longer possible, since the inverse of the Jacobian is no longer defined. Equivalently, covectors can still be transformed from task- to joint space (3.8), but not the other way around. Straightforwardly, pseudo-inverses can be used to perform the inverse transformations. Let $\boldsymbol{A}^{\#}$ denote a pseudo-inverse of the matrix $\boldsymbol{A}$. Then the inverse transformation of vectors is [39]

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\boldsymbol{q})\dot{\boldsymbol{x}} \tag{3.12}$$

and for covectors

$$\boldsymbol{f} = \boldsymbol{J}^{\#\top}(\boldsymbol{q})\boldsymbol{\tau}\,. \tag{3.13}$$

The mapping from task space vectors to joint space vectors is underdetermined. Therefore, there exists an infinite amount of pseudo-inverses for a rectangular matrix. One such matrix is the well-known Moore-Penrose pseudo-inverse (e.g. [63]):

Definition 12 (Moore-Penrose Pseudo-Inverse)**.** *The Moore-Penrose pseudo-inverse* $\boldsymbol{A}^+ \in \mathbb{R}^{n\times m}$ *of the matrix* $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ *is the result of*

$$\boldsymbol{A}^+ = \boldsymbol{A}^\top \left(\boldsymbol{A}\boldsymbol{A}^\top\right)^{-1} . \tag{3.14}$$

A generalization of the Moore-Penrose pseudo-inverse is the weighted pseudo-inverse [16]:

Definition 13 (Weighted Pseudo-Inverse)**.** *Let* $\boldsymbol{W} = \boldsymbol{W}^\top \succcurlyeq 0$ *be a positive-definite and symmetric weighing matrix. The weighted pseudo-inverse* $\boldsymbol{A}^{\boldsymbol{W}\#}$ *of the matrix* $\boldsymbol{A}$ *is*

$$\boldsymbol{A}^{\boldsymbol{W}\#} = \boldsymbol{W}^{-1}\boldsymbol{A}^\top \left(\boldsymbol{A}\boldsymbol{W}^{-1}\boldsymbol{A}^\top\right)^{-1} . \tag{3.15}$$

For $\boldsymbol{W} = \boldsymbol{I}$ the weighted pseudo-inverse equals the Moore-Penrose pseudo-inverse. For the transformation of vectors either of the pseudo-inverses can be used. However, the inverse mapping is underdetermined and different generalized inverses optimize the mapping for different things. One common choice is to choose the weighing matrix equal to the mass matrix of the manipulator $\boldsymbol{W} = \boldsymbol{M}(\boldsymbol{q})$. Then, the inverse mapping computes a joint space velocity $\dot{\boldsymbol{q}} = \boldsymbol{J}^{\boldsymbol{M}\#}(\boldsymbol{q})\dot{\boldsymbol{x}}$ with the minimal kinetic energy $\frac{1}{2}\dot{\boldsymbol{q}}^\top \boldsymbol{M}\dot{\boldsymbol{q}}$ of the manipulator.

3.3. Multi-Body Dynamics of a Manipulator

So far, only static configurations and kinematics of manipulators were considered. Dynamics describes the relationship between forces, velocities and acceleration and also depend on inertial properties of the links. In particular, the moments of inertia of the links are additionally required to derive the dynamics. An interconnection of rotating multiple bodies also introduces effects like Coriolis and centrifugal forces. Let n be again the degrees of freedom of the manipulator. The dynamics of the serial interconnection of links and joints can be written as an n-dimensional system of second-order differential equations [53, Chapter 4]

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + g(\boldsymbol{q}) = \boldsymbol{\tau} , \tag{3.16}$$

where $\boldsymbol{q}$ represents the vector of joint position and $\boldsymbol{\tau}$ the vector of applied joint torques, i.e. the sum of motor and external torques. Further, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ represent joint velocity and acceleration vectors. The remaining terms are

- $\boldsymbol{M}(\boldsymbol{q})$: The symmetric and positive-definite mass or inertia matrix. The mass matrix generally depends on the configuration $\boldsymbol{q}$ of the robot as moments of inertia change with changing angles between links. The off-diagonal coefficients of $\boldsymbol{M}(\boldsymbol{q})$ introduce second-order couplings between the joint dynamics.
- $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$: Matrix of Coriolis and centrifugal forces. This matrix also introduces nonlinear couplings between the joint dynamics. The choice of $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ is generally not unique. Here it is assumed to be chosen such that $\dot{\boldsymbol{M}}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})^\top$ as in [11].
- $g(\boldsymbol{q})$: The effect of gravity on the joint torques.

For the derivation of the quantities the kinematic model as well as inertial properties of the links are required. Further information on the derivation and properties of the dynamics can be found in [45, Chapter 8]. In this thesis, the dynamics model of the manipulator (3.16) is assumed to be known. Rewriting the dynamics a block diagram of the multi-body dynamics can be derived

$$\ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{q})\left[\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - g(\boldsymbol{q})\right] , \tag{3.17}$$

which is shown in Figure 3.11. This model can be used for simulation of manipulator dynamics and will be used in the evaluation chapter for simulating the closed-loop behavior of control schemes with a manipulator in the loop.

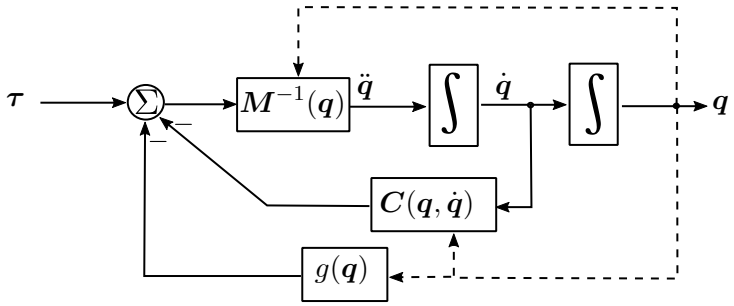


Figure 3.11.: Block diagram of the multi-body dynamics of a robotic manipulator.

For the planar example robot in Figure 3.2, the dynamics equation is a two-dimensional second-order differential equation. Without external torques $\boldsymbol{\tau} = 0$ the dynamics equals the one of an unforced double pendulum.

3.3.1. Task Space Dynamics

The differential equation (3.16) expresses the dynamics of the manipulator in joint space by relating torques, velocities and positions of the manipulator in joint space. The consequent next step is to consider the dynamics model in task space. This model then shows how the robot mass and coupling forces feel when observed in task space. Therefore, the dynamics model can be transformed to task space (e.g. [34])

$$\boldsymbol{M}_x(\boldsymbol{q})\ddot{\boldsymbol{x}} + \boldsymbol{C}_x(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{x}} + g_x(\boldsymbol{q}) = \boldsymbol{f}\,, \tag{3.18}$$

where $c_x(\boldsymbol{q}, \dot{\boldsymbol{q}})$ denotes end-effector Coriolis and centrifugal forces and $g_x(\boldsymbol{q})$ the effect of gravity to the end-effector. For a derivation see [33]. The main observation here is that the mass observed when applying forces to the end-effector is

$$\boldsymbol{M}_x(\boldsymbol{q}) = \left[\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{J}^\top(\boldsymbol{q})\right]^{-1}\,, \tag{3.19}$$

which naturally depends on the joint configuration $\boldsymbol{q}$.

3.3.2. Task Space Dynamics and Redundant Manipulators

Above, the equations for task space dynamics were derived. For non-redundant robots the relation between forces and joint torques is $\boldsymbol{\tau} = \boldsymbol{J}^\top(\boldsymbol{q})\boldsymbol{f}$. For redundant manipulators, however, this relation becomes incomplete. The set of task space coordinates is insufficient to completely describe the dynamical behavior of the manipulator. Consequently, the relation between task space forces and joint torques can be completed by adding null space torques [34]

$$\boldsymbol{\tau} = \boldsymbol{J}^\top(\boldsymbol{q})\boldsymbol{f} + \left[\boldsymbol{I} - \boldsymbol{J}^\top(\boldsymbol{q})\boldsymbol{J}^{\#\top}(\boldsymbol{q})\right]\boldsymbol{\tau}_0\,, \tag{3.20}$$

where $\boldsymbol{\tau}_0$ is an arbitrary joint torque vector. The matrix $\left[\boldsymbol{I} - \boldsymbol{J}^\top(\boldsymbol{q})\boldsymbol{J}^{\#\top}(\boldsymbol{q})\right] := \boldsymbol{P}(\boldsymbol{q})$ is a projector matrix that projects the vector $\boldsymbol{\tau}_0$ into the null space of $\boldsymbol{J}^{\#\top}$. The interpretation of this, is that any vector $\boldsymbol{\tau}_N$ projected by $\boldsymbol{\tau}_N = \boldsymbol{P}(\boldsymbol{q})\boldsymbol{\tau}_0$ will not interfere with any torque $\boldsymbol{\tau} = \boldsymbol{J}^\top(\boldsymbol{q})\boldsymbol{f}$ generated by a task space force. These torques $\boldsymbol{\tau}_N$ are also called *nondriving torques*, those joint torques that cannot be balanced by any task space force [16]. Any projector satisfying this is called statically consistent:

Definition 14 (Static Consistency)**.** *A null space projector matrix $\boldsymbol{P}(\boldsymbol{q})$ is called statically consistent if any joint torque does not interfere with torques generated by a task space force. It must hold that*

$$\boldsymbol{J}^{\#\top}(\boldsymbol{q})\boldsymbol{P}(\boldsymbol{q}) = 0\,. \tag{3.21}$$

As before, there are infinitely many pseudo-inverses $\boldsymbol{J}^{\#\top}(\boldsymbol{q})$ of the transposed Jacobian. Only one of those is consistent with the system dynamics meaning that no task space accelerations $\ddot{\boldsymbol{x}} = 0$ are generated. If so, the projector is called dynamically consistent [37]:

Definition 15 (Dynamical Consistency)**.** *A null space projector $\boldsymbol{P}(\boldsymbol{q})$ is called dynamically consistent if it is statically consistent and never generates task space accelerations*

$$\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{P}(\boldsymbol{q}) = 0\,. \tag{3.22}$$

When using the weighted pseudo-inverse $\boldsymbol{J}^{\boldsymbol{W}\#\top}(\boldsymbol{q})$ for the null space projector

$$\boldsymbol{P}_2(\boldsymbol{q}) = \boldsymbol{I} - \boldsymbol{J}^{\top}(\boldsymbol{q})\boldsymbol{J}^{\boldsymbol{W}\#\top}(\boldsymbol{q}) \tag{3.23}$$

and choosing the configuration dependent weighing matrix $\boldsymbol{W} = \boldsymbol{M}(\boldsymbol{q})$ results in a dynamically consistent null space projector [33].

3.4. Joint- and Task Space Impedance Control

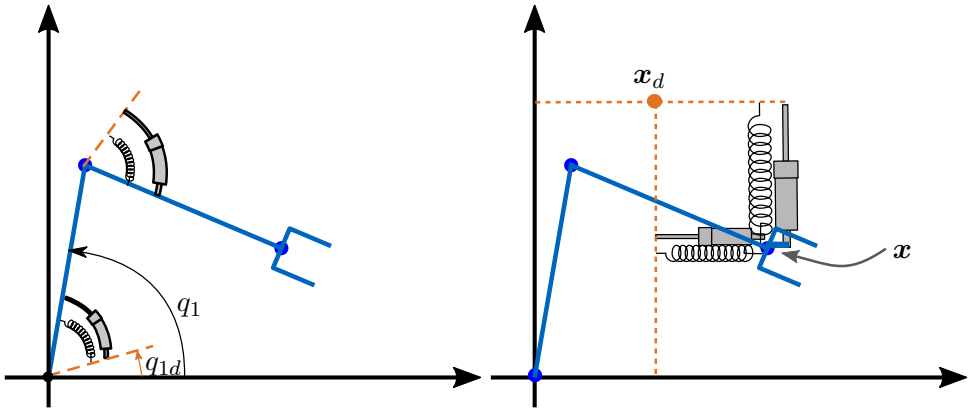
Impedance controllers impose desired dynamic behavior by virtually attaching linear springs and dampers to the manipulator. The idea is to have a framework, where springs and dampers with tunable stiffnesses and damping coefficients can be adapted to match the desired dynamic behavior for different tasks [28]. Then, the control goal is to change the manipulator dynamics such that it simulates the virtual system of springs and dampers. When programming the robotic manipulator, not only the desired positions are commanded, but additionally the desired dynamical properties. Springs and dampers can either be specified in joint or task space.

Figure 3.12 shows both cases for the example manipulator. On the left, the springs are attached to the robot joints, i.e. each joint simulates dynamics with linear rotational springs and dampers. In contrast, for most actual tasks the impedance of the end-effector in task space shall be controlled. On the right, the springs are attached in task space and the control goal is then to actuate the robot in a way such that the dynamics of the end-effector match the desired task space impedance.

The impedance controllers summarized here, set desired joint torques for the system. Another, lower level, controller then controls the motors in order to achieve the desired torque imposed by the outer level impedance controller. Feedback torque controllers require the torques in the joints, which need to be measured directly or estimated from other quantities. The DLR light weight robot [27], for instance, is a 7DoF robot with rotational joints, where each of the joints is equipped with torque sensors. Harmonic drive gears, which are used as gearing between motors and joints, induce flexibility which needs to be compensated in the torque controllers. An overview on torque controllers for modern robots is given in [2].

3.4.1. Joint Impedance Control

Let $\boldsymbol{q}_d$ be the desired joint position and $\dot{\boldsymbol{q}}_d$ be the desired joint velocity. When only regulation and no trajectory tracking is desired, the desired velocity is set to zero $\dot{\boldsymbol{q}}_d = 0$. For joint



(a) Joint Impedance Control (b) Task Space Impedance Control

Figure 3.12.: Impedance controllers impose dynamics of a manipulator by controlling the mechanical impedance of the structure. The robot simulates a virtual spring-damper system, where the stiffnesses and damping coefficients are usually task-specific design parameters. For joint impedance control each joint simulates a system with a rotational spring and damper, while for task space impedance control task space springs and dampers are defined. The springs and dampers are spanned between the desired and actual positions of the manipulator.

impedance control, the desired dynamic behavior of the robot is

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{D}(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) + \boldsymbol{K}(\boldsymbol{q} - \boldsymbol{q}_d) = \boldsymbol{\tau}_{ext}\,, \tag{3.24}$$

where $\boldsymbol{K}$ and $\boldsymbol{D}$ are positive definite and symmetric stiffness and damping matrices. Usually, these are diagonal and parametrize the stiffness and damping of each of the joints as shown in Figure 3.12a. The control approach is a PD controller with additional gravity compensation

$$\boldsymbol{\tau} = -\boldsymbol{K}(\boldsymbol{q} - \boldsymbol{q}_d) - \boldsymbol{D}(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) + g(\boldsymbol{q})\,. \tag{3.25}$$

Then, for the case of regulation $\dot{\boldsymbol{q}}_d = 0$ the achieved dynamic behavior of the system is

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + (\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{D})\,\dot{\boldsymbol{q}} + \boldsymbol{K}(\boldsymbol{q} - \boldsymbol{q}_d) = \boldsymbol{\tau}_{ext} \tag{3.26}$$

3.4.2. Task Impedance Control

Let $\boldsymbol{e}_x = \boldsymbol{x} - \boldsymbol{x}_d$ be the deviation of the desired pose from the actual pose in task space coordinates. The desired dynamics of the end-effector are

$$\boldsymbol{M}_x(\boldsymbol{q})\ddot{\boldsymbol{e}}_x + \boldsymbol{D}_x\dot{\boldsymbol{e}}_x + \boldsymbol{K}_x\boldsymbol{e}_x = \boldsymbol{f}_{ext}\,, \tag{3.27}$$

where $\boldsymbol{M}_x(\boldsymbol{q})$ denotes the transformed mass matrix (3.19) and $\boldsymbol{K}_x$, $\boldsymbol{D}_x$ are the desired task space stiffness and damping parameters, respectively. Again, here only the case of regulation $\dot{\boldsymbol{x}} = 0$ is considered, but it can easily be extended to trajectory tracking. First, the controller imposing the desired task space impedance is completely formulated in task space coordinates

$$\boldsymbol{f}_c = -\boldsymbol{K}_x(f(\boldsymbol{q}) - \boldsymbol{x}_d) - \boldsymbol{D}_x\dot{\boldsymbol{e}}_x\,. \tag{3.28}$$

When choosing second order dynamics, the damping ratio $\zeta \in [0..1]$ is a good parameter to tweak for the user. The mass matrix is not constant, but depends on the configuration of

the robot. Observing the dynamics equation (3.18), it becomes obvious that the damping matrix $\boldsymbol{D}_x$ must be selected dependent on the configuration $\boldsymbol{q}$ as the mass is not constant. Hence, it will be written $\boldsymbol{D}_x(\boldsymbol{q})$. One approach to choose the task space damping $\boldsymbol{D}_x(\boldsymbol{q})$ is the factorization design [1]. Let $\boldsymbol{M}_x^{1/2}(\boldsymbol{q})$ and $\boldsymbol{K}_x^{1/2}$ be the matrix square roots of the matrices $\boldsymbol{M}_x(\boldsymbol{q})$ and $\boldsymbol{K}_x$. Further, let $\boldsymbol{D}_\zeta = \mathrm{diag}\{\zeta_i\}$ be a diagonal matrix of the chosen damping ratios in the task space principal directions. Then, the configuration dependent damping matrix is computed using [1]

$$\boldsymbol{D}_x(\boldsymbol{q}) = \boldsymbol{M}_x^{1/2}(\boldsymbol{q})\boldsymbol{D}_\zeta\boldsymbol{K}_x^{1/2} + \boldsymbol{K}_x^{1/2}\boldsymbol{D}_\zeta\boldsymbol{M}_x^{1/2}(\boldsymbol{q})\,. \tag{3.29}$$

Given the task space controller the output force is transformed to joint torques and fed to the lower level torque controller. The overall task space impedance controller is then (using $\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}$)

$$\begin{aligned} \boldsymbol{\tau} &= \boldsymbol{J}^\top(\boldsymbol{q})\boldsymbol{f}_c + g(\boldsymbol{q}) && (3.30)\\ &= \boldsymbol{J}^\top(\boldsymbol{q})\left[-\boldsymbol{K}_x\left(f(\boldsymbol{q}) - \boldsymbol{x}_d\right) - \boldsymbol{D}_x(\boldsymbol{q})\left(\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}_d\right)\right] + g(\boldsymbol{q})\,, && (3.31)\end{aligned}$$

where again a term for gravity compensation is added. Figure 3.13 shows a schematic view of this control approach.

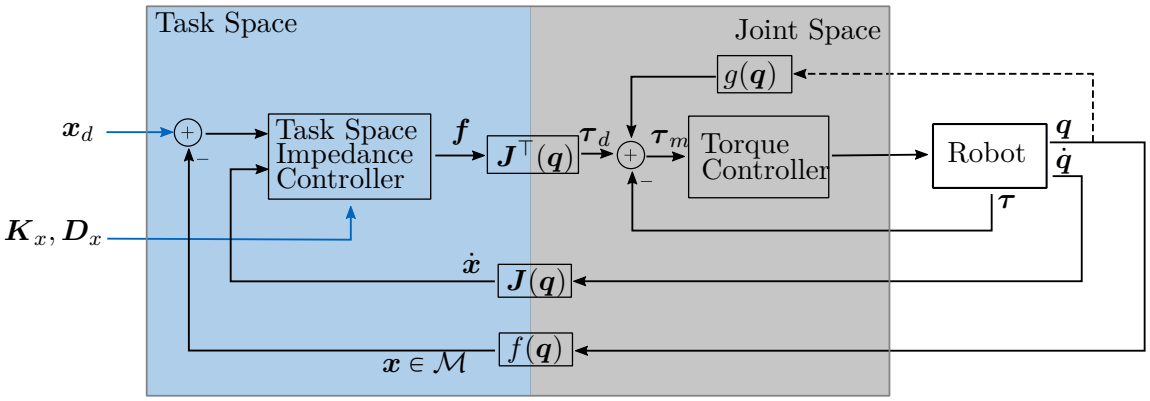


Figure 3.13.: Task space impedance control scheme. The task space impedance controller imposes a task space impedance by computing a task space force, which is transformed to joint space torques using the transposed Jacobian. The desired torque computed by the outer loop impedance controller is then fed to the inner loop torque controller. Measured or estimated torques are compared to the desired values and the motors are driven in order to achieve the torques set by the outer loop. Forward kinematics and its Jacobian are needed in the outer loop feedback closure for computation of the task space deviation and velocity. Gravity torques are also compensated in the control scheme.

3.4.3. Self-Motion Control aka Null Space Control

For redundant manipulators the set of task space coordinates is insufficient to completely describe the dynamics of a manipulator. Similarly, the task space impedance controller (3.30) does only control part of the manipulator dynamics. Recall that the completed relation between joint torques and task space forces is

$$\boldsymbol{\tau} = \underbrace{\boldsymbol{J}^\top(\boldsymbol{q})\boldsymbol{f}}_{\boldsymbol{\tau}_x} + \underbrace{\left[\boldsymbol{I} - \boldsymbol{J}^\top(\boldsymbol{q})\boldsymbol{J}^{\#\top}(\boldsymbol{q})\right]\boldsymbol{\tau}_0}_{\boldsymbol{\tau}_N}\,. \tag{3.32}$$

When choosing the dynamical consistent null space projector, there are two independent sets of control vector fields:

1. $\boldsymbol{\tau}_x$: joint torques corresponding to forces acting at the end-effector and
2. $\boldsymbol{\tau}_N$: joint torques that only affect internal motions.

The task space impedance controller (3.30) only controls the dynamics corresponding to $\boldsymbol{\tau}_x$ and leaves uncontrolled dynamics corresponding to $\boldsymbol{\tau}_N$. The redundancy can be used to add subtasks with lower priority. Subtasks are then projected into the null space of the task space Jacobian and not interfere with the task space impedance controller. Typical goals of such subtask controllers are avoidance of singularities [77], obstacles [47], joint limits and self-collisions [15]. Also approaches to minimize the reflected mass [49], to alter contact effects[89] or to minimize the energy of impacts [69] have been used.

The basic idea is to design another controller $\boldsymbol{\tau} = \Phi(\boldsymbol{q}, \dot{\boldsymbol{q}})$ for the subtask. For collision and joint limit avoidance tasks this is usually derived using repulsive potentials. Then, the output of that controller is projected to the null space using the null space projector matrix. The overall controller is then

$$\boldsymbol{\tau} = \underbrace{\boldsymbol{J}^\top(\boldsymbol{q})\left[-\boldsymbol{K}_x(f(\boldsymbol{q}) - \boldsymbol{x}_d) - \boldsymbol{D}_x(\boldsymbol{q})\left(\dot{\boldsymbol{x}} - \dot{\boldsymbol{x}}_d\right)\right]}_{\text{task space controller}} + \underbrace{\left[\boldsymbol{I} - \boldsymbol{J}^\top(\boldsymbol{q})\boldsymbol{J}^{\#\top}(\boldsymbol{q})\right]\Phi(\boldsymbol{q}, \dot{\boldsymbol{q}})}_{\text{self-motion controller}} + g(\boldsymbol{q})\,. \quad (3.33)$$

Note that this approach can also be continued recursively to generate a hierarchy of tasks, where lower priority tasks never interfere with higher priority tasks. [13] gives an overview on the approach for more than one subtask.

3.5. Vector Fields, Foliations and Integrability

This section wraps up some concepts from differential geometry and the connection to dynamical systems based on [82, 58]. The notions of vector fields and distributions on manifolds are introduced in order to state Frobenius' theorem. Finally, foliations are introduced, which are a fundamental concept in this thesis.

Vector fields on a manifold are defined in [42, Chapter 3]. Figure 3.14 shows an exemplary vector field on a 2-sphere.

Definition 16 (Vector Field)**.** *Let $\mathcal{M}$ be a smooth manifold. A vector field $\boldsymbol{f} : \mathcal{M} \to \mathcal{T}_x\mathcal{M}$ on $\mathcal{M}$ is a section of its tangent bundle $\mathcal{TM}$. It can be viewed as an assignment of a vector to each point $x \in \mathcal{M}$ such that the assigned vector lies in the tangent space at that point $\mathcal{T}_x\mathcal{M}$.*

Given a set of multiple vector fields on a manifold they form a distribution defined as

Definition 17 (Distribution)**.** *Let $\mathcal{M}$ be a manifold and suppose we have k vector fields $\boldsymbol{f}_i$. At every point the span of the vectors $\boldsymbol{f}_i$ forms a subspace of $\mathcal{T}_x\mathcal{M}$. This assignment of subspaces of $\mathcal{T}_x\mathcal{M}$ for every point $x \in \mathcal{M}$ is called distribution and is denoted*

$$\Delta = \operatorname{span}\{\boldsymbol{f}_1, ..., \boldsymbol{f}_k\}\,. \quad (3.34)$$

For instance, for the robotic manipulator each of the rows of the Jacobian matrix are vector fields on $\mathcal{Q}$. At every configuration $\boldsymbol{q} \in \mathcal{Q}$, a row of the Jacobian matrix $\boldsymbol{J}(\boldsymbol{q})$ assigns a vector to

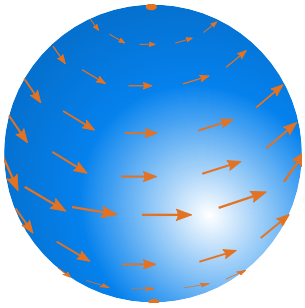


Figure 3.14.: Schematic drawing of a vector field on a manifold. The vector fields assign a vector to each point on the manifold such that the assigned vector lies within the tangent space at that point.

the point $\boldsymbol{q}$. These vectors lie in the tangent space $\mathcal{T}_q\mathcal{Q}$ and represent the direction of greatest ascent in the respective task space coordinates. The span of all the rows of the Jacobian at every point $\boldsymbol{q} \in \mathcal{Q}$ forms a distribution on $\mathcal{Q}$. Because the rows of the Jacobian are the gradient of a function, it is known that the distribution is integrable. This implies that the the system of partial differential equations given by the distribution of the vector fields in $\boldsymbol{J}(\boldsymbol{q})$ can be integrated in order to get the forward kinematics back. However, in general not every distribution is integrable.

Single vector fields can be integrated, if they are a gradient to a scalar function. In vector calculus it is known that only curl-free vector fields represent a gradient [51]. Curl-free vector fields are also called conservative vector fields and can be integrated. For integrability of distributions of vector fields another property called involutivity of the distribution is required. In order to define involutivity, first the definition of the Lie bracket is required.

Definition 18 (Lie Bracket)**.** *Let $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$ be vector fields. The Lie Bracket $[\cdot,\cdot]$ of $\boldsymbol{f}$ and $\boldsymbol{g}$ computes a third vector field defined as*

$$[\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{g}(\boldsymbol{x})] = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}) - \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x})\,. \tag{3.35}$$

Using the definition of Lie brackets, involutivity can be defined in the next step [82].

Definition 19 (Involutivity of a Distribution)**.** *A distribution Δ of k vector fields is called* involutive *if applying the Lie bracket to each pair of the vector fields in Δ implies that the resulting vector field lies also within Δ. Formally,*

$$\forall i, j \in [1..k] : \boldsymbol{f}_i, \boldsymbol{f}_j \in \Delta \implies [\boldsymbol{f}_i, \boldsymbol{f}_j] \in \Delta\,. \tag{3.36}$$

Frobenius' theorem provides a sufficient and necessary condition for a distribution to be integrable [42, Theorem 14.5]

Theorem 1 (Frobenius)**.** *Every involutive distribution is completely integrable.*

Complete integrability here means that it is possible to find a function whose gradients correspond to the vector fields in the distribution. Another interpretation is based on foliations. The term foliation is defined as in [42].

Definition 20 (Foliation)**.** *A collection $\mathcal{F}$ of disjoint, connected r-dimensional submanifolds of $\mathcal{M}$, whose union is $\mathcal{M}$, is called r-dimensional foliation of $\mathcal{M}$. The submanifolds in $\mathcal{F}$ are called leaves of the foliation.*

Figure 3.15 shows an example foliation of the solid torus. Important is that the leaves of the foliation never intersect and that the union of all of the leaves reconstructs the torus.

Complete integrability of a distribution means that there exists a foliation of integral manifolds. Take any of the leaves of that foliation. Then at every point on the leaf, the tangent space corresponds to the vector fields of the distribution. Starting at any point on one of the leaves and integrating along the vector fields will never leave the leaf.

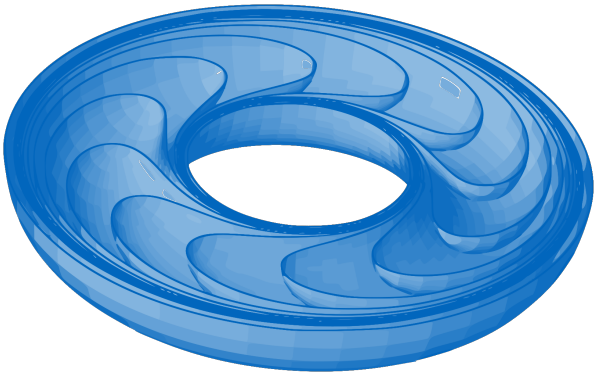


Figure 3.15.: Reeb foliation of a torus. Only some of the leaves of the foliation are shown. The union of all of the leaves of the foliation restores the entire torus. Figure inspired by [30].

4

Dynamically Decoupled Coordinates

This chapter introduces the new approach for foliation-based impedance control of redundant robots. Forward kinematics $f : \mathcal{Q} \to \mathcal{M}$ map from the n-dimensional joint- to the m-dimensional task space manifold. Task space impedance controllers do not control the entire dynamics and do not use all of the degrees of freedom of a manipulator. Here, a task space augmentation approach [80] with a special set of minimal coordinates is proposed for redundancy resolution. Task space augmentation means that besides the task coordinates an additional set of coordinate is chosen to control the remaining degrees of freedom. This new set of coordinates will be termed *self-motion coordinates* and denoted $\boldsymbol{\xi} \in \mathcal{S}$. For now, the manifold $\mathcal{S}$ abstractly represents the manifold of self-motion coordinates. Then, the coordinates are minimal if $\dim \mathcal{S} = n - m = r$. Figure 4.1 shows an example with the planar three DoF manipulator for a choice of only translational task space coordinates.

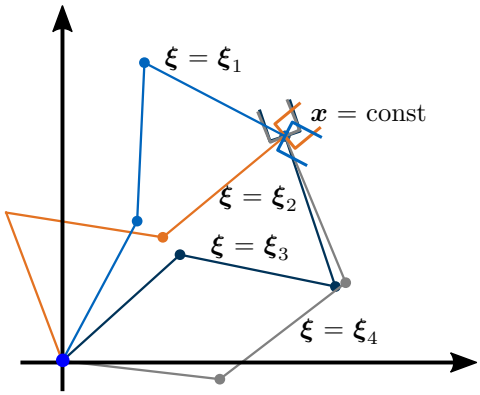


Figure 4.1.: Concept of coordinates for self-motions. The additional coordinate $\boldsymbol{\xi}$ assigns a self-motion coordinate to each of the configuration.

Similar to task space coordinates, these coordinates also have a "forward kinematics" function

$$\boldsymbol{\xi} = \varphi(\boldsymbol{q})\,. \tag{4.1}$$

There exist already many choices for task space augmentations like [76], [73] or the one implemented on the KUKA iiwa [10]. In this thesis, the main contribution is to design a set of self-motion coordinates which are orthogonal with respect to a given metric tensor. Choosing the inertia tensor as metric, dynamical decoupling from task space control is achieved. Dynamic decoupling refers to the fact, that any motion in $\boldsymbol{\xi}$ shall by construction not interfere with the task space controller. Vice versa, task space motions shall not affect the $\boldsymbol{\xi}$-coordinates. In particular, these statements shall hold when considering the pure dynamics of the manipulator without external forces. This implies that using the task space impedance controller (3.30)

without any self-motion control should not affect the $\boldsymbol{\xi}$ coordinates, when no external forces are present. Additionally, the function $\varphi(\boldsymbol{q})$ shall be smooth or at least continuously differentiable. Then, it also has a Jacobian and all transformations, equations and controllers known from task space can directly be applied to $\boldsymbol{\xi}$-coordinates.

Let $\boldsymbol{N}(\boldsymbol{q})$ denote the Jacobian of $\varphi(\boldsymbol{q})$ computed by the partial derivatives

$$\boldsymbol{N}(\boldsymbol{q}) = \left[\frac{\partial \varphi_i}{\partial q_j}\right] = \begin{bmatrix} \frac{\partial \varphi_1}{\partial q_1} & \cdots & \frac{\partial \varphi_1}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \varphi_r}{\partial q_1} & \cdots & \frac{\partial \varphi_r}{\partial q_n} \end{bmatrix}. \tag{4.2}$$

By simple pattern matching, the transformation of vectors can be derived. Let $\dot{\boldsymbol{\xi}} \in \mathcal{T}_{\xi}\mathcal{S}$ be a generalized self-motion velocity. It is transformed according to

$$\dot{\boldsymbol{\xi}} = \boldsymbol{N}(\boldsymbol{q})\dot{\boldsymbol{q}}\,. \tag{4.3}$$

Similarly, generalized forces $\boldsymbol{\eta} \in \mathcal{T}^*_{\xi}\mathcal{S}$ in $\boldsymbol{\xi}$-coordinates are transformed to joint torques by

$$\boldsymbol{\tau} = \boldsymbol{N}^{\top}(\boldsymbol{q})\boldsymbol{\eta}\,. \tag{4.4}$$

Equation (3.32) showed the decomposition of two dynamically decoupled control vectors in the relationship between joint torques and task forces [34]. This can be resembled without the null space projector

$$\boldsymbol{\tau} = \boldsymbol{J}^{\top}(\boldsymbol{q})\boldsymbol{f} + \boldsymbol{N}^{\top}(\boldsymbol{q})\boldsymbol{\eta}\,. \tag{4.5}$$

By construction, the self-motion coordinates are designed dynamically decoupled from the task space coordinates. Therefore, equation 4.5 is also dynamically decoupled and controllers in both coordinates can act independently.

The remainder of this chapter will develop the further insights needed to design these coordinates and give an argument why these coordinates can exist. First, a foliation-based perspective on task space forward kinematics is presented. The close relation between foliations and integrability of a distribution will be used to conclude the existence of such decoupled self-motion coordinates. In order to achieve the desired decoupling between the coordinate functions, the Jacobians must be in a certain relation. This relation is derived afterwards. First, only the kinematic level and static case are considered. Subsequently, the approach is extended for dynamical consistency using the multi-body dynamics equation. Finally, an overall controller using the coordinate function is designed.

All the statements and insights in the subsequent sections are valid under the assumption that the task space Jacobian $\boldsymbol{J}(\boldsymbol{q})$ has full-rank for all $\boldsymbol{q} \in \mathcal{Q}$. For actual task space coordinates this is not true. Still, it is always possible to find regions in the joint space, which are singularity-free and the approach developed here is valid in these. By the end of this chapter, it will be shown how all the singularity-free regions can be combined to apply the concept in the entire joint space.

4.1. Foliation Perspective on Forward Kinematics

Distributions of vector fields that correspond to the tangent bundle of a foliation are integrable. For redundant manipulators, task space coordinates determined by the forward kinematics function can essentially be interpreted as a certain type of foliation. This will be derived in this

section. Starting with only one task space coordinate, it will be shown that the choice of that forward kinematics function corresponds to a $(n-1)$-dimensional foliation of the joint space manifold $\mathcal{Q}$, where $n = \dim \mathcal{Q}$ represents the degrees of freedom of the manipulator. Then, it will be shown that for each additional task space coordinates, a joint space foliation with one dimension less is obtained. Finally, the complete forward kinematics $f : \mathcal{Q} \to \mathcal{M}$ creates a r-dimensional foliation, where r denotes the degree of redundancy of the robot.

Consider a smooth one-dimensional task space forward kinematics $x = f_1(\boldsymbol{q})$. It assigns a scalar to every joint configuration $\boldsymbol{q} \in \mathcal{Q}$. The submanifolds, where $f(\boldsymbol{q}) = const$ are called self-motion manifolds (definition 10). Here, because the task space coordinate is a scalar, the self-motion manifolds are $(n-1)$-dimensional. For the scalar forward kinematics, the Jacobian is a simple row vector of partial derivatives $\boldsymbol{J}_1(\boldsymbol{q}) = \operatorname{grad}^\top f(\boldsymbol{q})$. The gradient is normal to the self-motion manifold. Figure 4.2 shows a schematic drawing of this.

The function $f(\boldsymbol{q})$ is defined everywhere on the joint space, i.e. every $\boldsymbol{q} \in \mathcal{Q}$ is assigned a task space coordinate. Because the manipulator is redundant, every task space coordinate $x \in \mathcal{M}_1$ corresponds to a self-motion manifold. Additionally, the self-motion manifolds can never intersect[1]. Combining these insights, it follows that the entirety of self-motion manifolds creates a foliation of the joint space manifold. Let $\mathcal{F}_1$ denote the foliation corresponding to the self-motion manifolds of f_1.

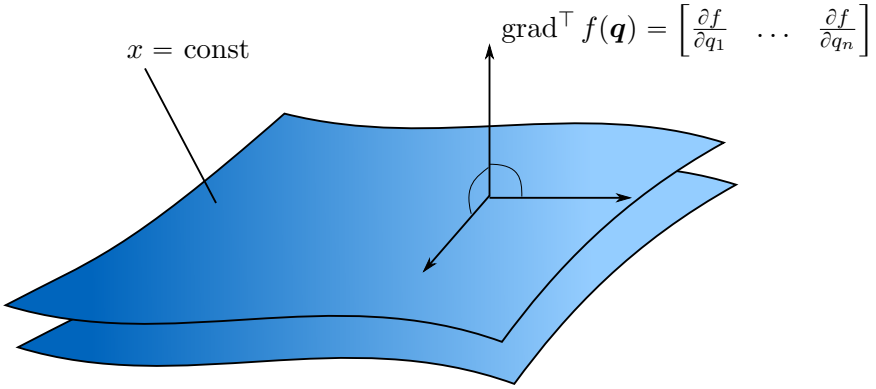


Figure 4.2.: Schematic drawing of self-motion manifolds patches for a scalar task space coordinate. The evaluation of the forward kinematics yields constant values for the points on one leaf. The Jacobian is normal to the surface. For the scalar case, the Jacobian corresponds to the transposed gradient. Drawing inspired by [40].

This concept is shown for the planar manipulator with three degrees of freedom. Choosing only the x-component of the end-effector pose of the example manipulator from Figure 4.1 creates such a scalar coordinate function. The degree of redundancy is $r = 3 - 1 = 2$. Consequently, the self-motion manifolds are two-dimensional surfaces in the joint space manifold. Figure 4.3 shows self-motion manifolds computed for different values of the scalar task space coordinate x. The surfaces are extracted using marching-cubes [43]. On the right, the entire joint space is shown from $-\pi$ to π is shown. Note the toroidal topology of the space. The cube shown in the figure can be smoothly continued in every direction by stacking exact copies of it in every direction. On the left, the front-top subcube of the joint space is shown. By rotating it (left bottom) it can be seen that the foliation structure continues in the inside of the cube. The colors encode the value of the task space coordinate $x_1 = f_1(\boldsymbol{q})$.

In the next step, consider a second coordinate function $y = f_2(\boldsymbol{q})$, which creates another foliation $\mathcal{F}_2$ of the joint space manifold. The leaves of the foliations generated by the self-motion

[1] This can be seen by contradiction using the intermediate value theorem.

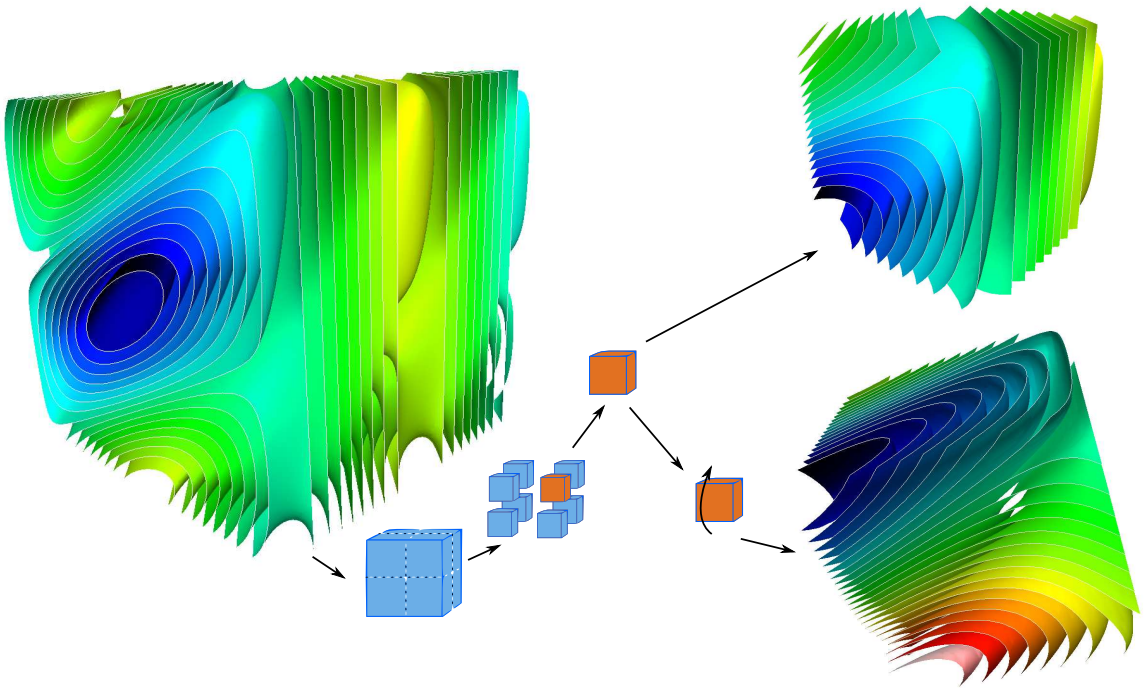


Figure 4.3.: Two-dimensional foliation of the joint space of the example manipulator. On the left the foliation is shown for the entire joint space from $-\pi$ to π for each of the three joint angles. Each of the leaves corresponds to one value for $x = f(\boldsymbol{q}) = \text{const}$. The front-top subcube with edge length $\frac{\pi}{2}$ is cut out and shown on the right. Rotating this subcube shows that the foliation continues in the inside of the joint space. Colors denote the value of the scalar coordinate x. Therefore, each leaf has a constant color.

manifolds of f_1 and f_2 intersect. Figure 4.4(a) shows this schematically. For now, it is assumed that the Jacobian is non-singular. This implies that the leaves of the foliation never touch tangentially, but intersect at every point. As shown in the figure, the intersection of the leaves creates another manifold of lower dimension. This lower dimensional manifold corresponds to a self-motion manifold of

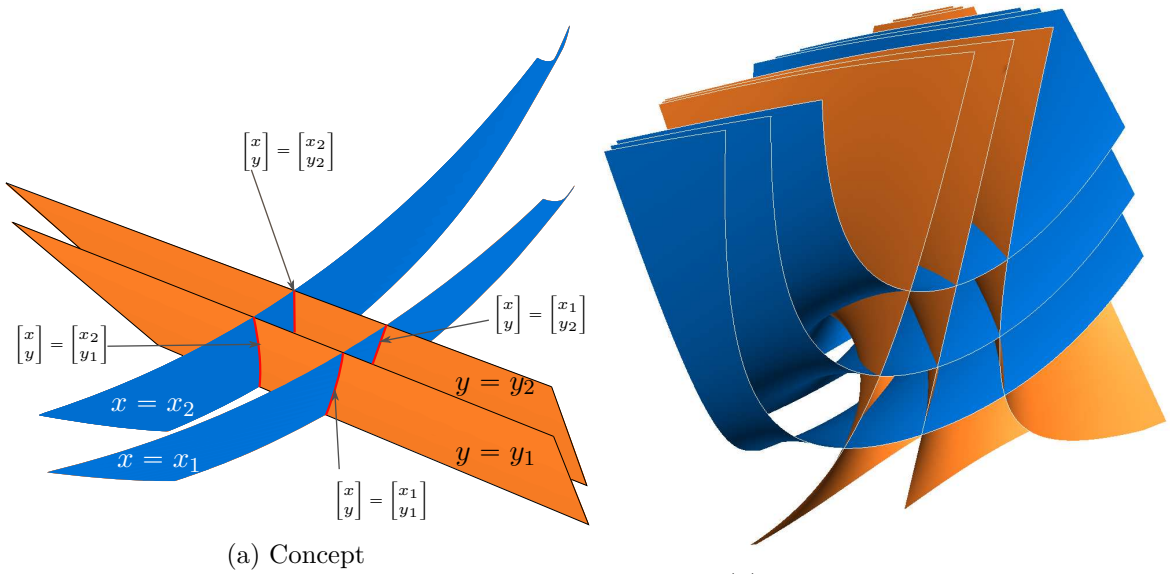
$$f_{12}(\boldsymbol{q}) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{q}) \\ f_2(\boldsymbol{q}) \end{bmatrix} . \tag{4.6}$$

Again, this is shown for a small patch of the joint space of the example manipulator in Figure 4.4(b).

The self-motion manifolds of f_{12} are also dense in $\mathcal{Q}$ and never intersect. Therefore, intersecting all the leaves of the foliations creates another foliation $\mathcal{F}_{12} = \mathcal{F}_1 \cap \mathcal{F}_2$ of $\mathcal{Q}$. This new foliation is now $(n-2)$-dimensional.

Also this can be verified using the example manipulator. Figure 3.8 already showed a single self-motion manifold. In order to observe the foliation structure, Figure 4.5 shows many of those corresponding to different values $\boldsymbol{x}_{12} = f_{12}(\boldsymbol{q})$. Further information on the geometry of those manifolds can be found in [6, Section 3]. Note, that all the curves are closed when considering the toroidal structure of $\mathcal{Q}$. Only the view in the Cartesian coordinate system shows open lines, but they all can be smoothly continued on the opposite side of the cube.

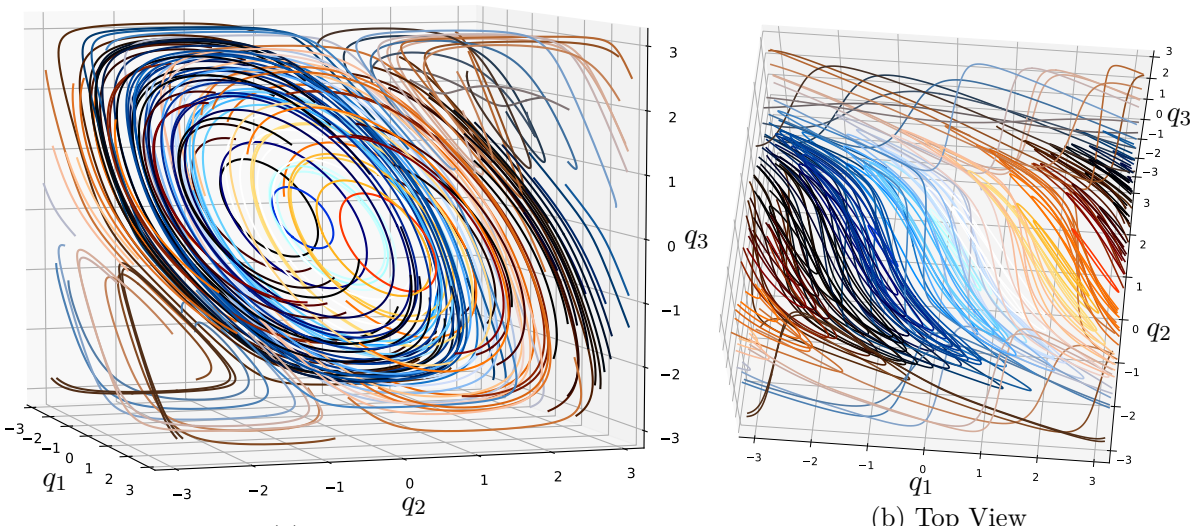
Imagine continuing the process until all task space coordinates are contained in the coordinate function f. The task space coordinates $f : \mathcal{Q} \to \mathcal{M}$ correspond to a r-dimension foliation of



(a) Concept

(b) Patch for example robot

Figure 4.4.: Intersection of the leaves of the self-motion manifolds corresponding to different coordinate functions creates manifolds of lower dimension. These correspond to self-motion manifolds of a combined coordinate function, where both coordinates stay constant.



(a) Front View

(b) Top View

Figure 4.5.: One-dimensional foliation of the example manipulator using the end-effector position as task space coordinates. The lines create a foliation of the joint space $\mathcal{Q}$.

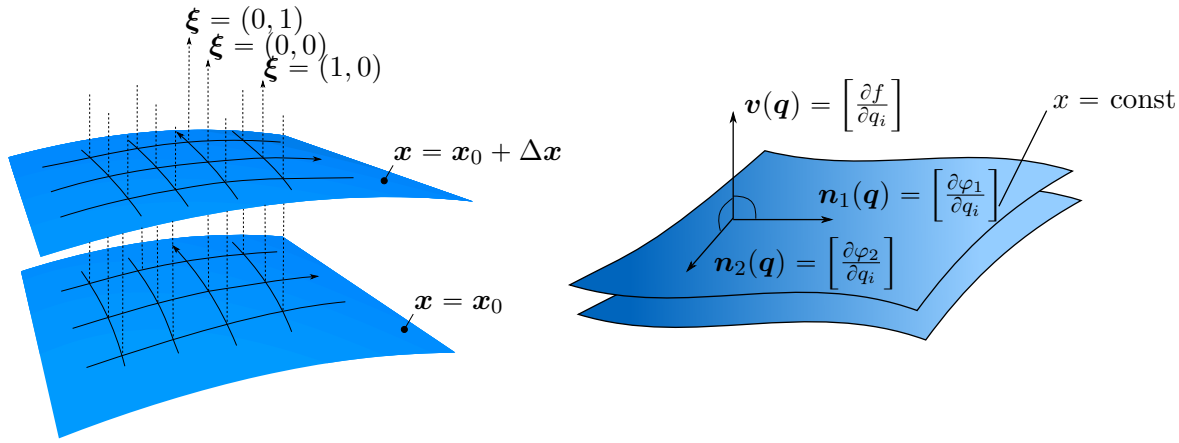
the joint space, where the leaves of the foliation are the self-motion manifolds. This leads to:

Observation 1. *The entirety of self-motion manifolds corresponding to the task space coordinates $\boldsymbol{x} = f(\boldsymbol{q})$ creates an r-dimensional foliation of $\mathcal{Q}$, where r is the degree of redundancy of the manipulator.*

4.2. Orthogonal Foliations

The goal is to design a coordinate function $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$ that does not interfere with the task space coordinates. Based on the foliation view on the forward kinematics, this section derives the required relation between the functions f and φ on a kinematic level, i.e. for now only kinematics are considered and the dynamics of the robot is not taken into account.

Consider the drawing on the left in Figure 4.6. Basically, the idea of the self-motion coordinates $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$ is to assign a coordinate chart onto the self-motion manifolds of the foliation induced by the task space coordinates. Additionally, when moving from one leaf to the next one the self-motion coordinates shall not jump, but vary smoothly. This is indicated by the dashed lines between the grids.



(a) Coordinate chart on manifolds

(b) Orthogonal Jacobians

Figure 4.6.: Coordinate on the self-motion manifolds. This drawings are valid if the Euclidean metric is used to define orthogonality. On the left the basic concept of assigning coordinates on the self-motion manifolds is presented. The coordinates shall be consistent when moving from one to the next leaf of the foliation induced by the task space coordinates. On the right, it is shown that the Jacobian of the self-motion coordinates must be in the tangent space of the manifold. Only then the effect on the task coordinates vanishes.

The effect of joint space displacements $\Delta\boldsymbol{q}$ on task space displacements $\Delta\boldsymbol{x}$ is

$$\Delta\boldsymbol{x} = \boldsymbol{J}(\boldsymbol{q})\Delta\boldsymbol{q}\,. \tag{4.7}$$

In order to avoid any task space displacement $\Delta\boldsymbol{x}$, the projection of the rows of the Jacobian $\boldsymbol{N}(\boldsymbol{q})$ of φ onto the rows of $\boldsymbol{J}(q)$ must vanish. In other words, the rows of $\boldsymbol{N}(\boldsymbol{q})$ must be orthogonal to the rows of $\boldsymbol{J}(\boldsymbol{q})$. Orthogonality between the rows of $\boldsymbol{N}(\boldsymbol{q})$ and $\boldsymbol{J}(\boldsymbol{q})$ can only be defined with respect to a given metric tensor as two vectors cannot be directly multiplied with each other. The rows $\boldsymbol{n}_i^\top(\boldsymbol{q})$ of $\boldsymbol{N}(\boldsymbol{q})$ are vectors in the tangent space of the self-motion manifolds. Figure 4.6(b) depicts this graphically for the choice of the Euclidean metric to define orthogonality. This leads to

Observation 2. *The rows $\boldsymbol{n}_i^\top(\boldsymbol{q})$ of $\boldsymbol{N}(\boldsymbol{q})$ are vector fields that map into the tangent bundle of the self-motion manifolds.*

Essentially, this leads to orthogonal foliations. The task space foliation and the foliations induced by the new $\boldsymbol{\xi}$-coordinates need to intersect orthogonally in order to fulfill the conditions

on the Jacobian. Consider the trivial example in Figure 4.7. The blue leaves represent a trivial foliation of a solid cube sliced into axis-aligned planes. Then, the other two foliations depicted in orange and gray are orthogonal at every point. Note, that only some leaves of the foliation are shown. The orthogonality condition, however, is also true at every point in the cube when considering the three leaves intersecting it.

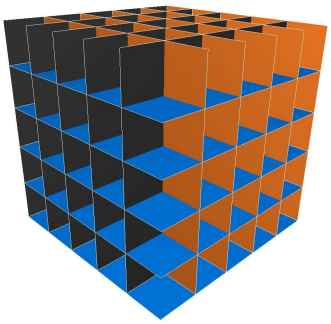


Figure 4.7.: Trivial example for orthogonal foliations. The surfaces represent trivial foliations of a solid cube by slicing it into axis-aligned planes. The image only shows some of the infinitely many leaves. At each point in the solid cube, three leaves intersect orthogonally.

When the base foliation is not that trivial, also the orthogonal foliations become more complicated. Figure 4.8 shows a foliation of the solid cube of the left. The solid unit cube is foliated into paraboloids. On the right, an approximate orthogonal foliation is shown by the leaves in gray and orange. The orthogonal foliations denoted in gray and orange were generated using the neural network based method developed later in this thesis.

These systems of curved surfaces in three-dimensional space are also subject of research in math (e.g. [3]) and called *triply orthogonal systems*. In three dimensions, Dupin's theorem provides a statement about the lines of intersection of the surfaces [83]:

Theorem 2 (Dupin)**.** *Given three mutually orthogonal surfaces, the line of intersection of any two surfaces of different families is a line of curvature for the surfaces.*

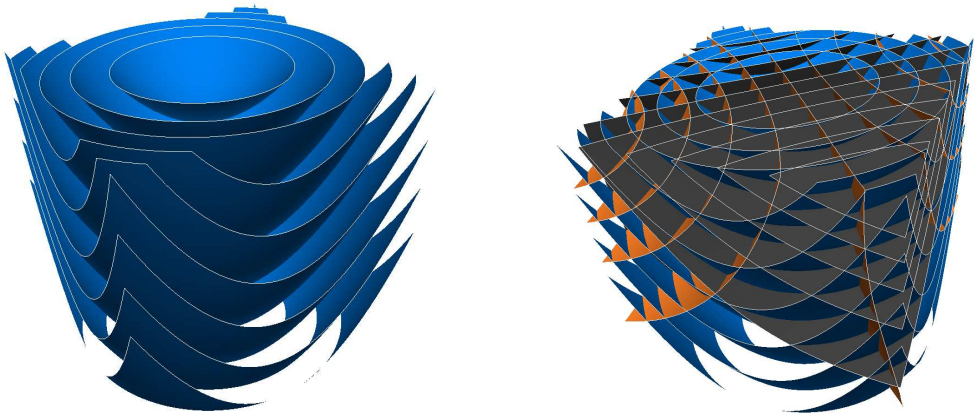
4.2.1. Existence of Orthogonal Self-Motion Foliations

This section combines the two observations of the previous section in order to conclude that orthogonal self-motion foliations (OSMFs) exist. Task space forward kinematics correspond to a $(n-m)$-dimensional foliation of $\mathcal{Q}$. The Jacobian $\boldsymbol{N}(\boldsymbol{q})$ is unknown, but it is known that each of the rows of $\boldsymbol{N}(\boldsymbol{q})$ are orthogonal to the rows of the task space Jacobian $\boldsymbol{J}(\boldsymbol{q})$ under a metric tensor. Hence, in order to fulfill the (for now kinematically) decoupled coordinates $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$, the Jacobian of φ must map into the tangent bundle of the self-motion manifolds.

Using the following theorem [42, Lemma 14.12] (rewritten according to [62])

Theorem 3. *If $\mathcal{F}$ is a r-dimensional foliation of $\mathcal{Q}$, then the collection of tangent spaces to the leaves of $\mathcal{F}$ form an involutive distribution.*

it can be concluded that the rows of $\boldsymbol{N}(\boldsymbol{q})$ form an involutive distribution. Additionally, Frobenius' theorem states that every involutive distribution is integrable (Thm. 1). Therefore,



(a) Cube foliation with paraboloids

(b) Triply orthogonal system

Figure 4.8.: Less trivial example of orthogonal foliation by a paraboloid-based foliation of a solid cube (blue). The orange and gray surface correspond to leaves of the two orthogonal foliations. 3D view of the orthogonal foliations on the right here: http://thesis.aaarne.de/triply-orthogonal-paraboloid.stl

Observation 3. *Orthogonal self-motion foliations (OSMFs) exist.*

The OSMFs provide a per se coordinate-free structure, i.e. a topological space. However, for computations in this space coordinates are required. For instance, they can be used for motion planning and impedance controllers. The arguments above show that the OSMFs exist. However, it does not provide statements about coordinates on the foliation. These coordinates will be called orthogonal self-motion coordinates (OSMCs).

Although it is known that the Jacobian $\boldsymbol{N}(\boldsymbol{q})$ of the OSMCs spans the tangent spaces of the self-motion manifolds, there are infinitely many instantiations of the r independent row-vector in $\boldsymbol{N}(\boldsymbol{q})$ in each tangent space. The huge majority of choices will not lead to integrable vector fields. This is the case for example for null space basis vectors generated by pseudo-inverses.

Consequently, the main question is how to obtain integrable vector fields $\boldsymbol{N}(\boldsymbol{q})$ in the integrable distribution and how to find the integrals in terms of the coordinate function of the OSMCs.

4.2.2. Dynamical Decoupling

So far, the relation between the coordinate functions $f : \mathcal{Q} \to \mathcal{M}$ and $\varphi : \mathcal{Q} \to \mathcal{S}$ was only analyzed on a kinematic level. Implicitly, it was assumed that the metric for orthogonality was the identity matrix, i.e.

$$\boldsymbol{n}_i^\top(\boldsymbol{q})\boldsymbol{I}\boldsymbol{v}_j(\boldsymbol{q}) = 0\,, \tag{4.8}$$

where $\boldsymbol{n}_i^\top(\boldsymbol{q})$ denotes the i-th row of $\boldsymbol{N}(\boldsymbol{q})$ and $\boldsymbol{v}_j^\top$ denotes the j-th row of $\boldsymbol{J}(\boldsymbol{q})$. In what follows the required metric for dynamical consistency will be derived.

Recall the multi-body dynamics equation of the manipulator

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + g(\boldsymbol{q}) = \boldsymbol{\tau} + \boldsymbol{\tau}_{ext}\,. \tag{4.9}$$

The mass matrix is generally non-diagonal and introduces second-order couplings between the differential equations. In the following, the explicit dependency of the matrices on the joint

configuration $\boldsymbol{q}$ is dropped in order to improve readability. The goal is to achieve decoupled task- and self-motion dynamics (cmp. to Khatib's version [34, eq. (19)])

$$\boldsymbol{\tau} = \boldsymbol{J}^\top \boldsymbol{f} + \boldsymbol{N}^\top \boldsymbol{\eta}\,. \tag{4.10}$$

Consider the case without external forces and assume the gravity torques are already compensated by the controller. Equations 4.9 and 4.10 can be matched to obtain

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{C}\dot{\boldsymbol{q}} = \boldsymbol{J}^\top \boldsymbol{f} + \boldsymbol{N}^\top \boldsymbol{\eta}\,. \tag{4.11}$$

Left-multiplication by $\boldsymbol{J}\boldsymbol{M}^{-1}$ yields

$$\boldsymbol{J}\ddot{\boldsymbol{q}} + \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{C}\dot{\boldsymbol{q}} = \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^\top \boldsymbol{f} + \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^\top \boldsymbol{\eta}\,. \tag{4.12}$$

Using $\boldsymbol{J}\ddot{\boldsymbol{q}} = \ddot{\boldsymbol{x}} - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}$ this can be rewritten to

$$\ddot{\boldsymbol{x}} - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{C}\dot{\boldsymbol{q}} = \underbrace{\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{J}^\top}_{\boldsymbol{M}_x^{-1}} \boldsymbol{f} + \underbrace{\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^\top}_{\overset{!}{=}0} \boldsymbol{\eta}\,. \tag{4.13}$$

The effect of $\boldsymbol{\eta}$ to the task space acceleration $\ddot{\boldsymbol{x}}$ is $\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^\top\boldsymbol{\eta}$. Consequently, the required metric for orthogonality between $\boldsymbol{J}$ and $\boldsymbol{N}$ is the inverse mass matrix:

Observation 4. *For dynamical consistency, the required metric for defining the scalar product between the rows of the Jacobians $\boldsymbol{J}(\boldsymbol{q})$ and $\boldsymbol{N}(\boldsymbol{q})$ is the inverse mass matrix*

$$\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{N}^\top(\boldsymbol{q}) = 0\,. \tag{4.14}$$

Here, the condition of dynamical consistency (definition 15) of null space projectors reappears. Additionally, the Jacobian $\boldsymbol{N}(\boldsymbol{q})$ is also statically consistent (definition 14). Using the dynamically consistent pseudo-inverse in the definition of static consistency yields

$$\left(\boldsymbol{J}^{M\#}\right)^\top \boldsymbol{N}^\top = \left(\boldsymbol{J}^\top \boldsymbol{M}^{-1}\boldsymbol{J}\right)^{-1} \underbrace{\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^\top}_{=0} = 0\,. \tag{4.15}$$

The same analysis can be done for the acceleration $\ddot{\boldsymbol{\xi}}$. Analogous to the previous derivation, the resulting dynamics is

$$\ddot{\boldsymbol{\xi}} - \dot{\boldsymbol{N}}\dot{\boldsymbol{q}} + \boldsymbol{N}\boldsymbol{M}^{-1}\boldsymbol{C}\dot{\boldsymbol{q}} = \underbrace{\boldsymbol{N}\boldsymbol{M}^{-1}\boldsymbol{J}^\top}_{\overset{!}{=}0} \boldsymbol{f} + \underbrace{\boldsymbol{N}\boldsymbol{M}^{-1}\boldsymbol{N}^\top}_{\boldsymbol{M}_\xi^{-1}} \boldsymbol{\eta}\,. \tag{4.16}$$

The mass matrix is symmetric, and inverses of symmetric matrices are symmetric again. Therefore,

$$\left[\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^\top\right]^\top = \boldsymbol{N}\boldsymbol{M}^{-1}\boldsymbol{J}^\top \tag{4.17}$$

and it can be concluded that ensuring $\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^\top = 0$ also enforces $\boldsymbol{N}\boldsymbol{M}^{-1}\boldsymbol{J}^\top = 0$. In consequence, also a task space force $\boldsymbol{f}$ does not result in any immediate acceleration $\ddot{\boldsymbol{\xi}}$. Comparing equations 4.13 and 4.15 and rewriting them yields

$$\ddot{\boldsymbol{x}} = \boldsymbol{M}_x^{-1}\boldsymbol{f} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} - \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{C}\dot{\boldsymbol{q}} \tag{4.18}$$

$$\ddot{\boldsymbol{\xi}} = \boldsymbol{M}_\xi^{-1}\boldsymbol{\eta} + \dot{\boldsymbol{N}}\dot{\boldsymbol{q}} - \boldsymbol{N}\boldsymbol{M}^{-1}\boldsymbol{C}\dot{\boldsymbol{q}}\,. \tag{4.19}$$

For the static equilibrium $\dot{\boldsymbol{q}} = 0$, the simplified dynamics is $\boldsymbol{M}_x \ddot{\boldsymbol{x}} = \boldsymbol{f}$ and $\boldsymbol{M}_\xi \ddot{\boldsymbol{\xi}} = \boldsymbol{\eta}$. However, the matrices $\boldsymbol{J} = \boldsymbol{J}(\boldsymbol{q})$, $\boldsymbol{N} = \boldsymbol{N}(\boldsymbol{q})$ and $\boldsymbol{M} = \boldsymbol{M}(\boldsymbol{q})$ are not constant, but depend on the configuration $\boldsymbol{q}$. For $\dot{\boldsymbol{q}} \neq 0$, the accelerations also depend on the change of the Jacobians $\dot{\boldsymbol{J}}$ and $\dot{\boldsymbol{N}}$ and on the Coriolis and centripetal forces.

It can be concluded, that enforcing $\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^\top = 0$ leads to decoupled coordinates. Any force in one space does not generate any immediate acceleration in the other space. In other words, the immediate effect of a generalized force $\boldsymbol{\eta}$ on the task space acceleration $\ddot{\boldsymbol{x}}$ is zero. However, a force $\boldsymbol{\eta}$ still accelerates the manipulator and accumulates a velocity $\dot{\boldsymbol{q}}$. This velocity $\dot{\boldsymbol{q}}$ may later lead to a task space acceleration via $\ddot{\boldsymbol{x}} = \left(\dot{\boldsymbol{J}} - \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{C}\right)\dot{\boldsymbol{q}}$. The same holds for the effect of a task space force $\boldsymbol{f}$ on the acceleration $\ddot{\boldsymbol{\xi}}$. Note that also the classical approach with the null space projectors has this problem.

4.2.3. Globality with Euclidean Parametrization of Self-Motion Manifolds?

The OSMCs live on an r-dimensional manifold, that was denoted $\mathcal{S}$ so far. Due to the foliation-version of Frobenius' theorem it is always possible to integrate the OSMFs. Suppose, that we can only design a function $\varphi : \mathcal{Q} \to \mathbb{R}^r$, i.e. a function that maps into Euclidean space. In fact, the method developed in the subsequent chapter will be based on a neural network, which inherently maps into an Euclidean space.

In the remainder of this section it will be shown that it is not always possible to express the r-dimensional OSMCs by Euclidean spaces of dimension not larger than r. However, locally it is always possible provided that the task space coordinate function is smooth. Recall the definition of a diffeomorphism, which creates a smooth an invertible maps between manifolds:

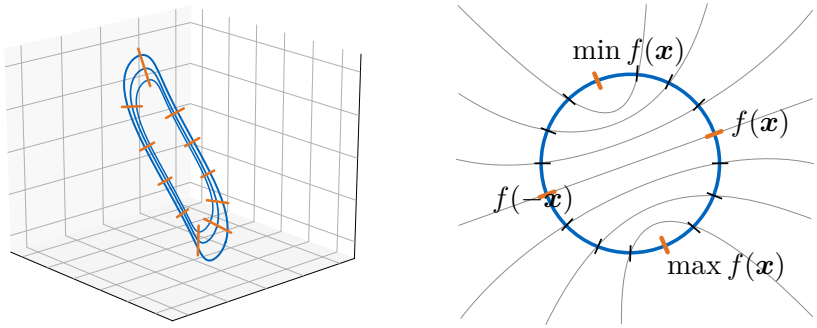
Definition 21 (Diffeomorphism)**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be smooth manifolds. An invertible function $f : \mathcal{X} \to \mathcal{Y}$ is called diffeomorphism if both, the function and its inverse are smooth.*

This is a special case of a homeomorphism, where the smoothness condition has been added. Here, the stronger version of a homeomorphism is required, because the Jacobian of φ shall be smooth. In order to find a global function $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$, where $\varphi : \mathcal{Q} \to \mathbb{R}^r$ there must exist a diffeomorphism between $\mathcal{S}$ and $\mathbb{R}^r$. If no such diffeomorphism exists, a function φ mapping into Euclidean space can only be locally valid. Note that this is no contradiction with the statement on integrability and the existence of self-motion coordinates. Potentially, it is always possible to find the coordinate function φ, however it is generally not always possible to do this with a function mapping into an Euclidean space.

As soon as any of the single self-motion manifolds is not topologically equivalent (or homeomorphic) to the Euclidean space $\mathbb{R}^r$, no globally valid function $\varphi : \mathcal{Q} \to \mathbb{R}^r$ with the desired properties can exist. Still, locally valid functions can always be found.

Consider Figure 4.9(a). The blue lines show three self-motion manifolds for the example manipulator with three joints and the end-effector positions as task space coordinates. Then, $r = 1$ and there is only one OSMC. The required function φ needs to find a parametrization of an orthogonal foliation such that the self-motion manifolds are normal to the leaves of the $\boldsymbol{\xi}$-foliation. Topologically, the three depicted self-motion manifolds are homeomorphic to the 1-sphere $\mathbb{S}^1$. Essentially, the OSMC assigns a number to each point on those manifolds, which is indicated by the orange ticks in the figure. This corresponds to parametrizing the unit circle with the real number line, which is not globally possible.

The Borsuk-Ulam theorem provides an even more general explanation why this is not globally possible [70, Chapter 6]:



(a) Grid on lines (b) Non-bijective mapping

Figure 4.9.: Coordinate grid on lines.

Theorem 4 (Borsuk-Ulam)**.** *Given a continuous function $f : \mathbb{S}^n \to \mathbb{R}^n$, there exists $\boldsymbol{x} \in \mathbb{S}^n$ such that $f(\boldsymbol{x}) = f(-\boldsymbol{x})$.*

In consequence, it is never possible to find a diffeomorphism between the n-sphere $\mathbb{S}^n$ and n-dimensional Euclidean space $\mathbb{R}^n$, because at least two points are mapped to the same image under the mapping.

The right half of Figure 4.9 shows this graphically. Suppose, one of the self-motion manifolds on the left has been mapped to the circle on the right. This is always possible, because they are homeomorphic. Then, there are no globally unique coordinates. Two points map to the same value by Borsuk-Ulam. Consequently, there arises a minimum and a maximum. No unique mapping is possible. This problem can be counteracted when using two coordinates, i.e. the 1-sphere is then embedded into a higher dimensional space. For example, the sine and cosine of an angle-parametrization of the circle provides a singularity-free and global mapping using two parameters.

As a second example, the self-motion manifolds of the same manipulator are used, but this time the y-component of the end-effector position is dropped from the task space coordinate function. Each of the links has a length of $\frac{1}{3}$. Then, two-dimensional self-motion manifolds are obtained. Figure 4.10 shows those manifolds for different ranges of the task space coordinate. Recall that the configuration space has toroidal topology and only the embedding into Euclidean space is shown here. For task space values smaller than $-\frac{1}{3}$ and larger than $\frac{1}{3}$ the self-motion manifolds show 2-sphere topology, i.e. they are closed, have no holes and the surface is two-dimensional. Between those values, the self-motion manifolds have one hole and are no longer homeomorphic to the 2-sphere. Therefore, the self-motion manifolds even change the topology in different regions.

Another restriction arises when a coordinate grid on even-dimensional spheres shall be designed. For instance, the self-motion manifolds on the leftmost and rightmost leaves in Figure 4.10 have $\mathbb{S}^2$ topology. Previously, for the 1-sphere mapping, the embedding into the Euclidean space could be repaired by adding another parameter and embedding the 1-sphere in $\mathbb{R}^2$ instead. For spheres of even-dimension the hairy ball theorem adds another restriction [66, Thm. 9.3]:

Theorem 5 (Hairy Ball Theorem)**.** *The sphere $\mathbb{S}^n$ admits a nowhere-zero tangent vector field if and only if n is odd.*

This implies that it is not possible to find a non-vanishing vector field on the 2-sphere. On $\mathbb{S}^1$ it is no problem to find a non-vanishing smooth vector field, for instance Figure 4.11(a)

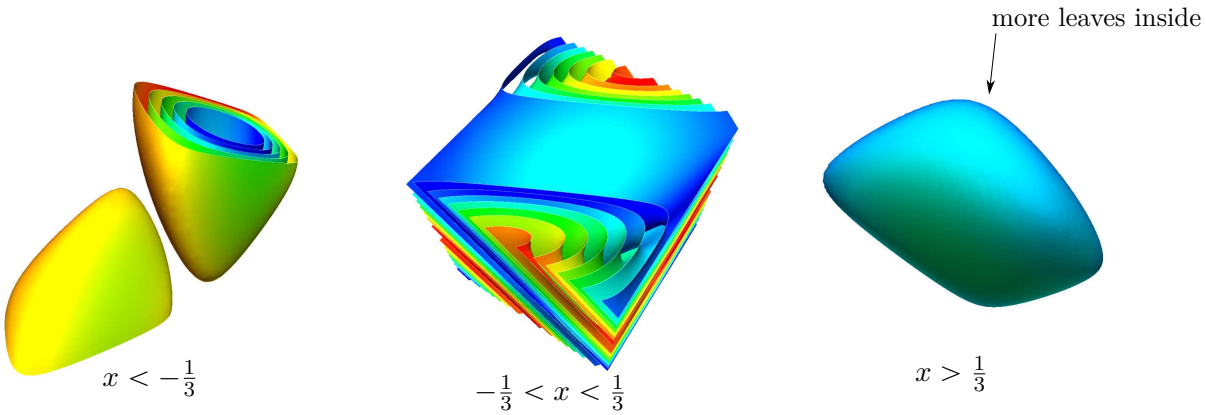
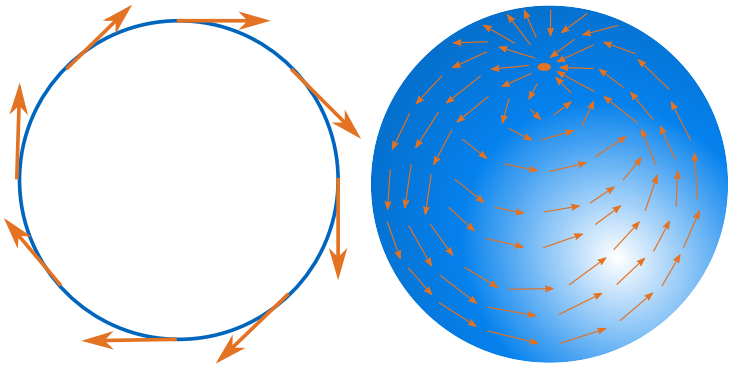


Figure 4.10.: Topology change of the self-motion manifolds. For specific range in task space coordinates, the self-motion manifolds show different topologies when embedding in Euclidean space.

shows a vector field with clock-wise rotating vectors. However, on $\mathbb{S}^2$ no such field is possible. Figure 4.11(b) shows a vector field on $\mathbb{S}^2$ with one pole. Naturally, that is not the only possible vector field on the 2-sphere, but a vector field without any pole can never be constructed as stated by the hairy ball theorem. As the self-motion manifolds on the left and right of Figure 4.10 are homeomorphic to the 2-sphere, non-vanishing vector field cannot be found on them. In consequence, in addition to the parametrization problem of the manifold, it is not possible to find a singularity-free vector field distribution on them. When considering the entire self-motion manifolds with 2-sphere topology, each of the rows of $\boldsymbol{N}(\boldsymbol{q})$ must vanish at least at one point as implied by the hairy ball theorem.



(a) Vector field on $\mathbb{S}^1$ (b) Vector field on $\mathbb{S}^2$

Figure 4.11.: Hairy ball theorem. While odd-dimensional spheres admit nowhere-zero vector fields, even-dimensional spheres do not. On the left a nowhere-zero vector field is shown on the 1-sphere. The vector fields on the right has one pole. (b) is inspired by [67].

It can be concluded that it is generally not possible to find singularity-free OSMCs. Only in exceptional cases, when the topology of every self-motion manifold admits this, singularity-free coordinates can be designed. This is, among others, possible when $r = 1$. On one-dimensional manifolds non-vanishing vector fields always exists. This can be summarized in the following observation:

Observation 5. *Generally, it is not possible to find a coordinate function for OSMCs, which is singularity-free, i.e. its Jacobian will be rank-deficient at least at one point per self-motion*

manifold. This does not hold when $r = 1$.

Additionally, the topology of the self-motion manifolds is generally not homeomorphic to $\mathbb{R}^r$. Therefore, the embedding of the self-motion manifolds into $\mathbb{R}^r$ is not possible. For instance, the self-motion manifolds shown in Figure 4.9(a) can not be globally embedded in $\mathbb{R}^1$. As shown in Figure 4.9(b) the embedding still works locally, where the 1-sphere is split into two hemispheres. This leads to

Observation 6. *Embedding of the r-dimensional OSMCs into $\mathbb{R}^r$ is generally not globally possible as closed surfaces are not homeomorphic to $\mathbb{R}^r$. When embedding into $\mathbb{R}^r$ is required, this leads to only locally unique coordinates. Globally, they may be ambiguous.*

The classical null space projection method has the same problem. The pseudo-inverse would lead to vanishing joint space components for specific settings. This is an intrinsic topological problem that arises due to the generally non-Euclidean manifolds. On general manifolds, an atlas of coordinate charts is required to cover the entire manifold with singularity-free coordinates:

Observation 7. *An atlas of coordinate charts is required to navigate on a non-Euclidean manifold. This holds for the earth as well as for the self-motion manifolds of redundant manipulators.*

4.3. Overall Controller

The OSMCs are chosen such that they do not interfere with the task space coordinates. Consequently, the control approach is to add two independent impedance controllers. The task space part is controlled using

$$\boldsymbol{f}_c = -\boldsymbol{K}_x \left(f(\boldsymbol{q}) - \boldsymbol{x}_d \right) - \boldsymbol{D}_x(\boldsymbol{q}) \dot{\boldsymbol{x}} \tag{4.20}$$

and the self-motion part is controlled by another impedance controller with individual stiffness and damping matrices

$$\boldsymbol{\eta}_c = -\boldsymbol{K}_\xi \left(\varphi(\boldsymbol{q}) - \boldsymbol{\xi}_d \right) - \boldsymbol{D}_\xi(\boldsymbol{q}) \dot{\boldsymbol{\xi}} \,. \tag{4.21}$$

Then, the joint torque is computed by adding up the joint torques resulting from the transformed generalized forces from the individual impedance controllers. Additionally, the term for gravity compensation is added. The overall controller is

$$\boldsymbol{\tau}_d = \boldsymbol{J}^\top(\boldsymbol{q}) \boldsymbol{f}_c + \boldsymbol{N}^\top(\boldsymbol{q}) \boldsymbol{\eta}_c + g(\boldsymbol{q}) \,. \tag{4.22}$$

Figure 4.12 shows the block diagram of this control approach.

As already seen for the task space impedance controller, the damping matrix can not be constant. The effective task space mass is

$$\boldsymbol{M}_x(\boldsymbol{q}) = \left[\boldsymbol{J}(\boldsymbol{q}) \boldsymbol{M}^{-1}(\boldsymbol{q}) \boldsymbol{J}^\top(\boldsymbol{q}) \right]^{-1} \,. \tag{4.23}$$

In order to achieve the same damping behavior for every configuration, the damping matrix $\boldsymbol{D}_x(\boldsymbol{q})$ must depend on the effective mass. It is chosen using the damping design equation (3.29). Similarly, the effective mass for the self-motion part is

$$\boldsymbol{M}_\xi(\boldsymbol{q}) = \left[\boldsymbol{N}(\boldsymbol{q}) \boldsymbol{M}^{-1}(\boldsymbol{q}) \boldsymbol{N}^\top(\boldsymbol{q}) \right]^{-1} \tag{4.24}$$

and the damping matrix $\boldsymbol{D}_\xi(\boldsymbol{q})$ is also chosen using damping design (3.29) using the mass matrix $\boldsymbol{M}_\xi(\boldsymbol{q})$ and the stiffness $\boldsymbol{K}_\xi$.

In the framework of task space augmentation [80] this approach can be written as new coordinates $\boldsymbol{x}_a$

$$\boldsymbol{x}_a = f_a(\boldsymbol{q}) = \begin{bmatrix} f(\boldsymbol{q}) \\ \varphi(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{bmatrix}. \tag{4.25}$$

Then, the resulting Jacobian of the new coordinates is

$$\boldsymbol{J}_a(\boldsymbol{q}) = \begin{bmatrix} \boldsymbol{J}(\boldsymbol{q}) \\ \boldsymbol{N}(\boldsymbol{q}) \end{bmatrix}, \tag{4.26}$$

where the first and the second block of $\boldsymbol{J}_a(\boldsymbol{q})$ are mutually orthogonal. The impedance controller can then be written as

$$\boldsymbol{f}_a = -\boldsymbol{K}_a \left(\begin{bmatrix} f(\boldsymbol{q}) \\ \varphi(\boldsymbol{q}) \end{bmatrix} - \begin{bmatrix} \boldsymbol{x}_d \\ \boldsymbol{\xi}_d \end{bmatrix} \right) - \boldsymbol{D}_a(\boldsymbol{q}) \boldsymbol{J}_a \dot{\boldsymbol{q}}. \tag{4.27}$$

Again, the damping matrix $\boldsymbol{D}_a(\boldsymbol{q})$ can be designed via damping design, where the mass matrix $\boldsymbol{M}_a(\boldsymbol{q}) = \left[\boldsymbol{J}_a \boldsymbol{M}^{-1} \boldsymbol{J}_a^\top \right]^{-1}$ and the stiffness $\boldsymbol{K}_a$ are used in the damping design equation (3.29):

$$\boldsymbol{D}_a(\boldsymbol{q}) = \boldsymbol{M}_a^{1/2} \boldsymbol{D}_\zeta \boldsymbol{K}_a^{1/2} + \boldsymbol{K}_a^{1/2} \boldsymbol{D}_\zeta \boldsymbol{M}_a^{1/2}. \tag{4.28}$$

The overall controller is then obtained by transforming the force $\boldsymbol{f}_a = \begin{bmatrix} \boldsymbol{f}^\top & \boldsymbol{\eta}^\top \end{bmatrix}^\top$ to joint space

$$\boldsymbol{\tau} = \boldsymbol{J}_a^\top \boldsymbol{f}_a + g(\boldsymbol{q}). \tag{4.29}$$

The Jacobian $\boldsymbol{J}_a(\boldsymbol{q})$ is a square-shaped $n \times n$ matrix and can be, except for singularities, inverted. This can be useful for numerical inverse kinematics, computation of forces and for design of feed-forward controllers as will be shown later.

4.4. Summary

This section summarizes the concept for control of redundant kinematic structures using orthogonal foliations. Basically, the idea is to find coordinates $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$ that specify the remaining free choices of the configuration of a robot arm when only the desired task space coordinate is given. Then, any motion in those new coordinates shall by construction not interfere with the task in task space coordinates. Pure motion in those coordinates will only affect the configuration of the robot, but not the task space position. Therefore, the coordinate are called orthogonal self-motion coordinates (OSMCs). Throughout the chapter some important observation were highlighted. These are summarized here.

In the beginning of the chapter it was shown that choosing a task space coordinate function corresponds to implicitly imposing a foliation of the n-dimensional joint space manifold $\mathcal{Q}$. For redundant manipulators, the preimages of the task space forward kinematics lie on r-dimensional manifolds, where $r = n - m$ denotes the degree of redundancy of the manipulator and m the number of task space coordinates. Because, these so-called self-motion manifolds are dense within $\mathcal{Q}$ and they can never intersect, they form a foliation. Suppose, we have r linearly independent vector fields in the tangent spaces of the self-motion manifolds. Then, the distribution of these vector fields is integrable because of Frobenius' theorem. These r vector fields correspond to the rows of the Jacobian $\boldsymbol{N}(\boldsymbol{q})$ of the OSMCs φ. In order to achieve the

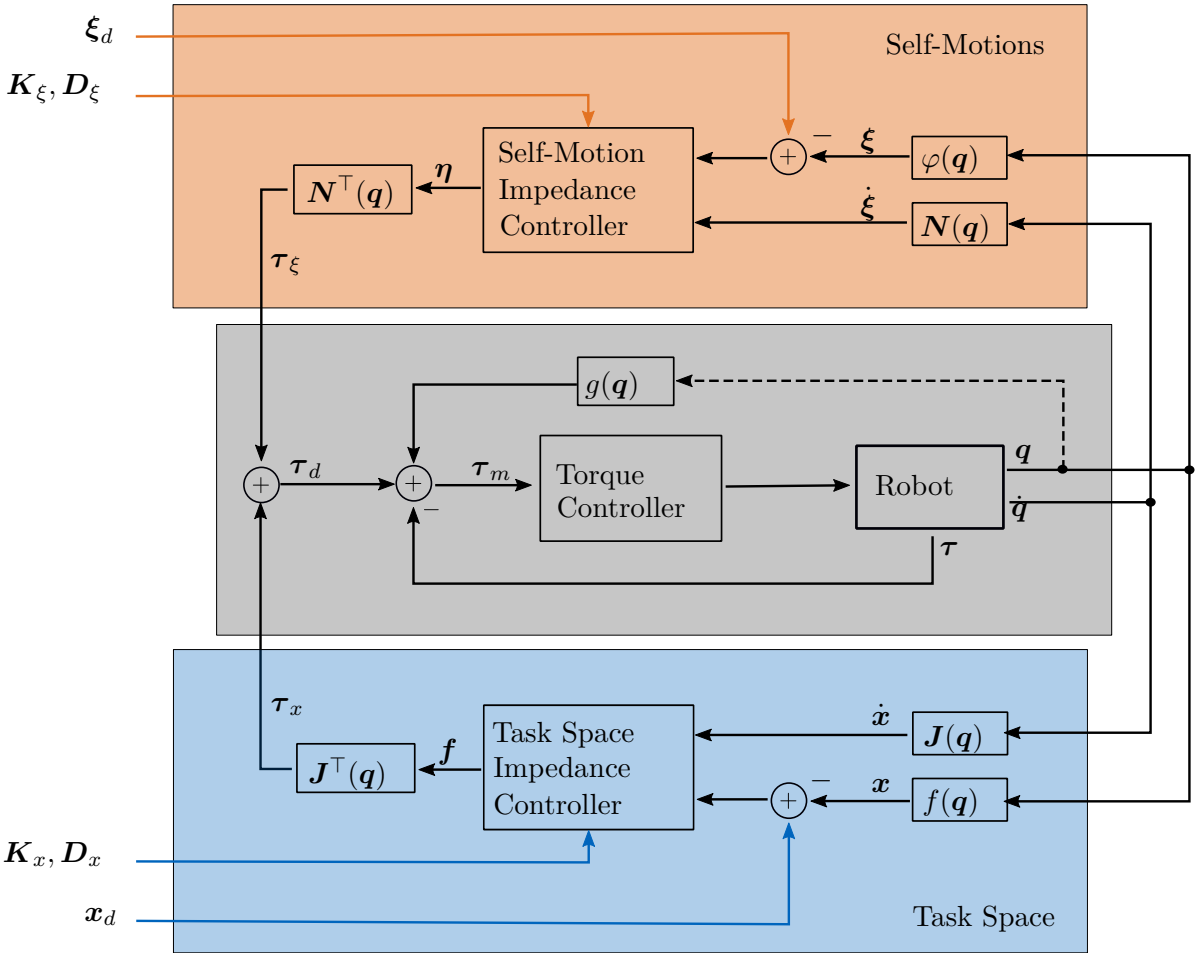


Figure 4.12.: Overall controller. Two independent impedance controllers compute forces in the respective spaces. Using the transpose of the Jacobian, the forces are transformed to joint torques. The torques are added up and fed to the inner control loop controlling the actual joint torques of the joints using the electrical drives. For appropriate coordinate functions f and φ the generated torques do not interfere with each other.

desired decoupling between the r vector field in the rows of $\boldsymbol{N}(\boldsymbol{q})$ must be orthogonal to the rows of $\boldsymbol{J}(\boldsymbol{q})$. Consequently, the function φ creates r orthogonal foliations, where *orthogonal* refers to the fact that every leaf of the foliations is orthogonal to the task space Jacobian $\boldsymbol{J}(\boldsymbol{q})$. For dynamical consistency, the inverse mass matrix $\boldsymbol{M}^{-1}(\boldsymbol{q})$ needs to be chosen for the metric for that orthogonality condition. Then, any motion in one of the coordinates will not generate accelerations in the opposite coordinate space.

Generally, it is not possible to find an OSMC function $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$, which is globally singularity-free. Additionally, the r-dimensional self-motion manifolds can generally not be embedded in $\mathbb{R}^r$. When this embedding into r-dimensional Euclidean space is required, this leads to ambiguous coordinates, i.e. more than one point of the self-motion manifolds may map to the same $\boldsymbol{\xi}$.

The next chapter will develop a neural network based approach to approximately find such a function φ for OSMCs.

5

Neural Networks for Orthogonal Foliations

The goal of this thesis is to design a coordinate function $\varphi : \mathcal{Q} \to \mathcal{S}$ for the OSMCs, where $\mathcal{Q}$ denotes the n-dimensional joint space manifold and $\dim \mathcal{S} = r$. In the previous section, conditions on the Jacobian $\boldsymbol{N}(\boldsymbol{q}) \in \mathbb{R}^{r \times n}$ of that function were derived. Particularly, it is required that the rows of $\boldsymbol{N}(\boldsymbol{q})$ are orthogonal to the rows of $\boldsymbol{J}(\boldsymbol{q})$ under some metric $\boldsymbol{A}(\boldsymbol{q})$. Here, $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{n \times n}$ refers to the Jacobian of the task space forward kinematics $f : \mathcal{Q} \to \mathcal{M}$, where $\mathcal{M}$ denotes the m-dimensional task space manifold. This orthogonality condition can be written as

$$\boldsymbol{J}(\boldsymbol{q})\boldsymbol{A}(\boldsymbol{q})\boldsymbol{N}^\top(\boldsymbol{q}) = 0\,. \tag{5.1}$$

Two different choices for the metric $\boldsymbol{A}(\boldsymbol{q}) \in \mathbb{R}^{n \times n}$ will be used later for the experiments. For $\boldsymbol{A}(\boldsymbol{q}) = \boldsymbol{I}$ the coordinates are decoupled on a kinematic level and the orthogonality conditions can be geometrically interpreted as orthogonality between vectors in the common, Euclidean sense. On the other hand, when incorporating the dynamics of the manipulator it was shown that the metric $\boldsymbol{A}(\boldsymbol{q}) = \boldsymbol{M}^{-1}(\boldsymbol{q})$ needs to be chosen for dynamically consistent coordinates, i.e. coordinates that do not generate interfering accelerations in the opposite space.

This problem can be reinterpreted as finding a solution to a system of partial differential equations with varying coefficients. When explicitly writing the Jacobian $\boldsymbol{N}(\boldsymbol{q})$ in terms of the elements this can be clearly observed:

$$\boldsymbol{J}(\boldsymbol{q})\boldsymbol{A}(\boldsymbol{q}) \begin{bmatrix} \frac{\partial \varphi_1}{\partial q_1} & \cdots & \frac{\partial \varphi_r}{\partial q_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial \varphi_1}{\partial q_n} & \cdots & \frac{\partial \varphi_r}{\partial q_n} \end{bmatrix} = 0\,, \tag{5.2}$$

which essentially describes a system of r independent partial differential equations of the form

$$\boldsymbol{J}(\boldsymbol{q})\boldsymbol{A}(\boldsymbol{q})\nabla\varphi_j(\boldsymbol{q}) = 0\,, \quad \forall j \in \{1, \dots r\}\,. \tag{5.3}$$

Each of the r equations describes an independent partial differential equation with the same varying coefficients $\boldsymbol{J}(\boldsymbol{q})\boldsymbol{A}(\boldsymbol{q})$. Therefore, the PDE in (5.2) would be fulfilled when choosing $\varphi_1(\boldsymbol{q}) = \ldots = \varphi_r(\boldsymbol{q})$. Additionally, choosing a constant function $\varphi_j(\boldsymbol{q}) = \text{const}$ also trivially fulfills the PDE. However, these two settings are not desirable. For the control approach, the coordinates need to be non-constant and in order to control every remaining degree of freedom the rank of $\boldsymbol{N}(\boldsymbol{q})$ needs to be r. By requiring $\forall \boldsymbol{q} \in \mathcal{Q} : \operatorname{rank} \boldsymbol{N}(\boldsymbol{q}) = r$, both of the problems can be avoided.

The target function φ is a solution to an underdetermined system of partial differential equations. This can be observed because the coefficient matrix $\boldsymbol{J}(\boldsymbol{q})\boldsymbol{A}(\boldsymbol{q}) \in \mathbb{R}^{m \times n}$ is rectangular for redundant manipulators. Additionally, when considering how the Jacobian and mass matrix are

usually computed, it can be seen that the coefficients $\boldsymbol{J}(\boldsymbol{q})\boldsymbol{A}(\boldsymbol{q})$ are complicated combinations of weighted sines and cosines of different angles and other smooth terms. Consequently, the matrix $\boldsymbol{J}(\boldsymbol{q})\boldsymbol{A}(\boldsymbol{q})$ behaves smoothly, but writing down the functions generating the coefficients leads to page-filling expressions for common robots.

In interconnection and damping assignment passivity-based control (IDA-PBC) (e.g. [60]) also underdetermined systems of partial differential need to be solved with additional constraints, like positive-definiteness, on the solution. For example, in the non-parametrized approach [61] to solve IDA-PBC, the remaining rows of the coefficient matrix are chosen by educated guesses such that the coefficient matrix is full-rank and the system is transformed to a square-shaped system. Then, the system becomes determined up to boundary conditions and a solution can be obtained by standard PDE methods. However, this approach requires finding a function computing the remaining rows of $\boldsymbol{J}(\boldsymbol{q})\boldsymbol{A}(\boldsymbol{q})$. For the problem at hand it is not clear how to do this, because of the complicated expressions leading to the Jacobian and inverse mass.

For the approach of this thesis, it is sufficient when any solution to the PDE is obtained, provided that it fulfills the condition on the row rank of the Jacobian $\boldsymbol{N}(\boldsymbol{q})$. This chapter will develop a method to find a solution to the underdetermined system of partial differential equations given by (5.2). The basic idea is to use a variational principle via a neural network (Figure 5.1). The model parameters are optimized in order to satisfy the PDE and the additional conditions on the rank of the Jacobian. Training of the model corresponds to simultaneously finding a solution φ and a concrete instantiation of the PDE.

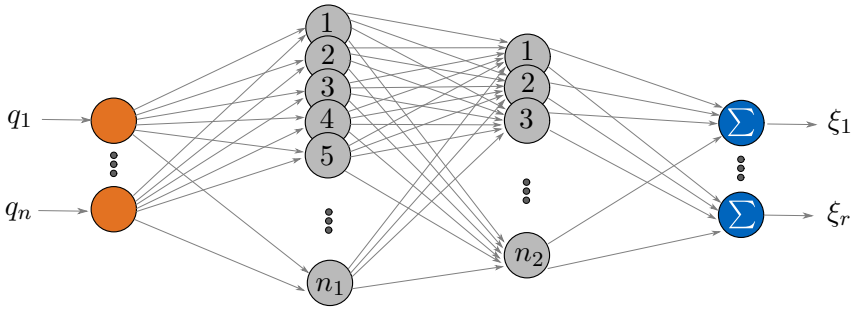


Figure 5.1.: Neural network for variational principle to find a smooth coordinate function φ. The input neurons take the joint angle and the output neurons correspond to the self-motion coordinates ξ_i.

First, the forward model of the neural network will be shown. Looking at the output with respect to input Jacobian of the neural network model, it can be observed that the model can also be optimized based on a scalar cost function that is dependent on the Jacobian of the network and not only the plain output. Subsequently, a cost function needed for the optimization of the neural network will be developed. The cost function is evaluated on a set of discrete samples, which are commonly known as training samples. Fortunately, here the training samples can be generated on the fly based on the robot model. This is beneficial, because classical problems like overfitting on training data and too little data do not occur here. The sampling strategy for training samples is shown later in this chapter. Finally, some implementation details and the training loop are presented.

5.1. Variational Principle via Neural Network

Any of the full rank solutions to the partial differential equation satisfies the need for a coordinate function $\varphi : \mathcal{Q} \to \mathcal{S}$. Basically, the idea is that a solution to the function can, at least approximately, be written as

$$\boldsymbol{\xi} = \varphi_\theta(\boldsymbol{q})\,, \tag{5.4}$$

where φ_θ is a nonlinear function with a lot of parameters that can be tuned in order to match the conditions on the solution. Given a parameter vector θ, it can be evaluated how well the function fits to the equations and conditions. In particular, the Jacobian $\boldsymbol{N}_\theta(\boldsymbol{q})$ of $\varphi_\theta(\boldsymbol{q})$ should satisfy (5.2) and also should be full rank. How well the parameters θ perform is evaluated based on a scalar cost function $L(\theta)$. Then, the parameters of the function are updated using gradient descent on the cost function.

Neural networks are nonlinear functions with a layered structure and many tunable parameters. They are in a sense very universal function approximators and the required model capacity can be adapted by adding/removing layers or neurons per layer. Here, the function φ_θ is a neural network with two hidden layers and nonlinear activation functions $\mathrm{act}_1, \mathrm{act}_2$ acting component-wisely (cmp. Figure 5.1 for a graphical view on this)

$$\boldsymbol{\xi} = \varphi_\theta(\boldsymbol{q}) = \boldsymbol{W}_{out}\left[\mathrm{act}_1\left(\boldsymbol{W}_2\left[\mathrm{act}_2\left(\boldsymbol{W}_1\boldsymbol{q} + \boldsymbol{b}_1\right)\right] + \boldsymbol{b}_2\right)\right] + \boldsymbol{b}_{out}\,, \tag{5.5}$$

where $\boldsymbol{W}_1$, $\boldsymbol{W}_2$ and $\boldsymbol{W}_{out}$ are weight matrices and $\boldsymbol{b}_1$, $\boldsymbol{b}_2$ and $\boldsymbol{b}_{out}$ are bias vectors. The parameter vector θ denotes a collection of all of the elements on the weight matrices $\boldsymbol{W}$ and the bias vectors $\boldsymbol{b}$. Let there be n_1 neurons and n_2 neurons in the first and second hidden layer, respectively. Then, the dimensions of the weight matrices and bias vectors are shown in Table 5.1.

Table 5.1.: Dimensions of the parameter matrices and vectors

	Linear Weight Matrix	**Bias Vector**
Hidden Layer 1	$\boldsymbol{W}_1 \in \mathbb{R}^{n_1 \times n}$	$\boldsymbol{b}_1 \in \mathbb{R}^{n_1}$
Hidden Layer 2	$\boldsymbol{W}_2 \in \mathbb{R}^{n_2 \times n_1}$	$\boldsymbol{b}_2 \in \mathbb{R}^{n_2}$
Output Layer	$\boldsymbol{W}_{out} \in \mathbb{R}^{r \times n_2}$	$\boldsymbol{b}_{out} \in \mathbb{R}^{r}$

Typical activation functions are the rectified linear unit (ReLU), the sigmoid function $\sigma(\cdot)$ or the hyperbolic tangent function. The function $\mathrm{ReLU}(x) = \max(0, x)$ would lead to a differentiable, but not continuously differentiable model. As the model is required to be continuously differentiable, the rectified linear unit function can not be used. In this thesis the hyperbolic tangent function tanh is chosen. It generally performs better than the logistic sigmoid [23, Sec. 6.3.2].

5.1.1. Network Jacobian

Differentiating φ_θ with respect to $\boldsymbol{q}$ leads to the Jacobian of the neural network

$$\boldsymbol{N}_\theta(\boldsymbol{q}) = \left[\frac{\partial(\varphi_\theta)_i}{\partial q_j}\right](\boldsymbol{q})\,. \tag{5.6}$$

Except for $\boldsymbol{b}_{out}$, the network Jacobian $\boldsymbol{N}_\theta(\boldsymbol{q})$ also depends on all the parameters θ. Moreover, when applying the chain rule on the forward pass model (5.5), an analytical expression of the Jacobian $\boldsymbol{N}_\theta(\boldsymbol{q})$ can be derived. This also enables differentiation of $\boldsymbol{N}_\theta(\boldsymbol{q})$ with respect to the

parameters θ. It can be concluded that the cost function $L(\theta)$ can also depend on the Jacobian $\boldsymbol{N}_\theta(\boldsymbol{q})$ and gradient descent for optimization of θ can be performed. Because $\boldsymbol{N}_\theta(\boldsymbol{q})$ is the Jacobian of the function $\varphi_\theta(\boldsymbol{q})$, changing the parameters θ changes the Jacobian as well as the function. By definition, $\varphi_\theta(\boldsymbol{q})$ is *always* a solution to the PDE given by $\boldsymbol{N}_\theta(\boldsymbol{q})$. Note that this approach can also be used for integration of vector fields by basically performing regression on given vector fields. For the regression, however, not the model output is fitted to the given vector field, but the input-output Jacobian of the model. Then, the actual output of the model approximates the integral of the vector field. This shows a motivating example for using a neural model, or more generally a parametrizable and optimizable function, for integration problems without analytic solution.

Note that is is essential to know that the distribution is integrable. Otherwise, the network would only provide the best approximation in a regression sense. Remember that integrability has been shown in Section 4.2.1.

5.1.2. Loss Function

For optimization of the model a scalar cost function is required. It will be evaluated on a set of training samples $\boldsymbol{q}_k$ for $k \in \{1, ..., K\}$. Let $\boldsymbol{A}(\boldsymbol{q})$ be the metric with respect to which we require the vector fields to be orthogonal. Further, a notation $\boldsymbol{v}_i^\top(\boldsymbol{q})$ for the rows of the task space Jacobian $\boldsymbol{J}(\boldsymbol{q})$ and for the rows $\boldsymbol{n}_i^\top(\boldsymbol{q}, \theta)$ of the self-motion Jacobian $\boldsymbol{N}_\theta(\boldsymbol{q})$ is introduced as

$$\boldsymbol{J}(\boldsymbol{q}) = \begin{bmatrix} \frac{\partial f_i}{\partial q_j} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_1^\top(\boldsymbol{q}) \\ \vdots \\ \boldsymbol{v}_m^\top(\boldsymbol{q}) \end{bmatrix} \tag{5.7}$$

$$\text{and } \boldsymbol{N}_\theta(\boldsymbol{q}) = \begin{bmatrix} \frac{\partial g_i}{\partial q_j} \end{bmatrix} = \begin{bmatrix} \boldsymbol{n}_1^\top(\boldsymbol{q}, \theta) \\ \vdots \\ \boldsymbol{n}_r^\top(\boldsymbol{q}, \theta) \end{bmatrix}. \tag{5.8}$$

Given a training sample $\boldsymbol{q}_k$, the metric $\boldsymbol{A}(\boldsymbol{q}_k)$, the Jacobian $\boldsymbol{J}(\boldsymbol{q}_k)$ and the predicted self-motion Jacobian $\boldsymbol{N}_\theta(\boldsymbol{q}_k)$ are computed. In order to remove clutter from the equations, the explicit dependency of $\boldsymbol{A}$, $\boldsymbol{v}_i^\top$ and $\boldsymbol{n}_i^\top$ on $\boldsymbol{q}$ and θ is omitted when clear out of context.

For the cost function one more definition from discrete mathematics is required.

Definition 22 (Iverson Bracket)**.** *The Iverson bracket* $[S]$ *is defined as [24]*

$$[S] = \begin{cases} 1, & \textit{if } S \textit{ is true,} \\ 0, & \textit{otherwise}. \end{cases} \tag{5.9}$$

Then, for one training sample the cost function is computed by summing all cosines of the angles between all rows of the task- and self-motion Jacobians and all cosines between mutual rows of the self-motion Jacobian. Let $\binom{n}{k}$ denote the binomial coefficient. Then the loss of one training sample $\boldsymbol{q}$ is computed according to

$$L(\boldsymbol{q}, \theta) = \underbrace{\frac{1}{2mr} \sum_{i=1}^{m} \sum_{j=1}^{r} \left(\frac{\boldsymbol{v}_i^\top \boldsymbol{A} \boldsymbol{n}_j}{|\boldsymbol{A}\boldsymbol{v}_i| \cdot |\boldsymbol{n}_j|} \right)^2}_{\text{task vs. self-motion}} + \underbrace{\frac{1}{2\binom{r}{2}} \sum_{i=1}^{r} \sum_{j=1}^{r} \left[\left(\frac{\boldsymbol{n}_i^\top \boldsymbol{A} \boldsymbol{n}_j}{|\boldsymbol{A}\boldsymbol{n}_i| \cdot |\boldsymbol{n}_j|} \right)^2 [i > j] \right]}_{\text{self-motion vs. self-motion}}, \tag{5.10}$$

where the Iverson bracket $[i > j]$ ensures that each combination is only counted once and that each row is not compared to itself. Generally, the neural network is a function approximator and

will therefore never be able to *exactly* converge to the desired function. However, as stated earlier, also any constant function satisfies the PDE. Neural networks can model constant functions exactly without remaining residuals. Consequently, the constant function $\varphi_\theta(\boldsymbol{q}) = const$ would be a global optimum to the cost function and a cost function taking the pure metric $L_{bad,ij}(\boldsymbol{q},\theta) = (\boldsymbol{v}_i^\top \boldsymbol{A}\boldsymbol{n}_j)^2$ would guide the parameters towards that undesirable optimum. In order to counteract this, the metrics between the vectors are normalized in the designed cost functions. Essentially, this can be interpreted as summing up the squared cosines of the angles between the vectors for $\boldsymbol{A}(\boldsymbol{q}) = \boldsymbol{I}$. For other metrics, one of the vectors is first rotated and stretched by the metric and then the cosine is computed.

The second term in $L(\boldsymbol{q},\theta)$ ensures that the Jacobian $\boldsymbol{N}_\theta(\boldsymbol{q})$ does not become rank deficient. Additionally, it even dynamically decouples the individual functions $\xi_i = \varphi_i(\boldsymbol{q})$ by using the same metric as for the decoupling between $\boldsymbol{J}$ and $\boldsymbol{N}$. Therefore, this loss function not only leads to decoupled task- and self-motion coordinates, but also the individual components of the self-motion coordinates are decoupled.

Finally, let there be K training samples available. The total cost of the parameters θ is

$$L(\theta) = \frac{1}{K}\sum_{k=1}^{K} L(\boldsymbol{q}_k,\theta)\,. \tag{5.11}$$

5.1.3. Generalization

Regularization refers to any modification of the training process that aims to lower the generalization error, but not the training error [23, Chap. 7]. In order to avoid overfitting and improve generalization of the model, regularization techniques are generally required.

Parameter norm penalties [23, Sec. 7.1] are very widely used techniques that basically penalize the magnitude of the parameters. For the PDE integration objective in this thesis this approach can not be used. Similar to unnormalized dot-products between the Jacobian, parameter norm penalties create gradients pushing the network to the undesired global optimum $\varphi(\boldsymbol{q}) = const$. Indirectly, parameter norm penalties penalize the length of the Jacobian vectors as well. Any penalization of the length of the Jacobian vectors must be avoided, else the network tends towards the global optimum of a constant function.

Fortunately, the robot model and the task space coordinate function are known and can be evaluated for arbitrary joint space configurations $\boldsymbol{q}$. In order to provide generalization of the network, the set of training samples will be exchanged frequently during the training process. During the training, new training samples can be generated randomly, which can potentially generate as many samples as desired. Using this technique, generalization of the model can be achieved. The training process is separated into multiple epochs. In each epoch the model is trained on a new set of training samples. Still, overfitting may occur in each epoch, which needs to be reverted in the beginning of the next epoch when the training samples have been replaced. Therefore, the amount of training steps per epoch needs to be tuned in order to prevent overfitting by the end of the epoch. This corresponds to an epoch-wise *early stopping* regularization [23, Sec. 7.8].

5.2. Training Sample Generation

Training samples can be generated randomly based on the robot model. These samples can then be used for both, training and evaluation of the model. Singular configurations should not

be part of the sample set in order to not disturb the training process. At singular configurations the task space Jacobian becomes row-rank deficient and the task space coordinates are no longer a good representation of the manipulator for control. The task space Jacobian $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{m \times n}$ is not square, so no straightforward computation of a determinant-based rank is possible. In order to evaluate the row-rank of the rectangular Jacobian matrix and to be robust to numerical instabilities, the singular value decomposition [87] of the Jacobian

$$\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\top} = \boldsymbol{J}(\boldsymbol{q}) \tag{5.12}$$

is used. Only the smallest singular value σ_{mn} is used to determine if the Jacobian is row-rank deficient. For efficiency, the matrices $\boldsymbol{U}$, $\boldsymbol{V}$ do not need to be computed. If the smallest singular value is smaller than a predefined threshold $\alpha^2_{thrshld}$, then the Jacobian will be considered rank-deficient.

Additionally, the model can be trained for only specific ranges in the task space. Computing the forward kinematics, the task space position of a joint space sample can be evaluated. For faster training and smaller model sizes, only samples corresponding to specific task space positions can be sampled.

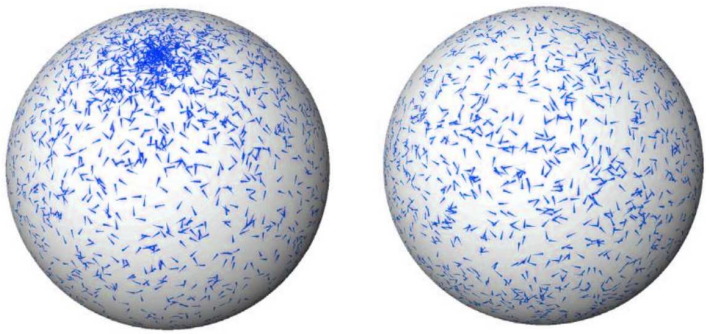
Sampling of training data is implemented using a rejection sampling scheme. Algorithm 1 summarizes the procedure used for sampling. The joint space vectors are uniformly sampled in a predefined region called *bounds*. Then, the Jacobian of the sampled joint configuration is evaluated and the singular values are computed. If the smallest singular value is large enough, the sampled joint configuration becomes a candidate for the sample set. Afterwards, the forward kinematics function is evaluated and the sample is kept when it satisfies the desired task space region. When no further task space constraints are desired, the check on the task space range always returns `true`. If either of the checks fails, the sample is rejected. The loop is implemented parallelized and the generated samples are written to a stream. As soon as n samples are collected from the stream, the sampling procedure finishes.

Algorithm 1 Rejection sampling for training data

function SAMPLECONFIGURATIONS(n)
  $Q \leftarrow \{\}$
  repeat ▷ Implemented as parallel loop
    sample $\boldsymbol{q} \sim \mathcal{U}(\mathit{bounds})$ ▷ Uniform distribution on volume in $\mathcal{Q}$
    $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^{\top} \leftarrow \mathrm{SVD}(\boldsymbol{J}(\boldsymbol{q}))$ ▷ Singular value decomposition
    if $\sqrt{\sigma_{mn}} \geqslant \alpha_{thrshld}$ **then** ▷ Check last singular value of $\boldsymbol{J}(\boldsymbol{q})$
      $\boldsymbol{x} = f(\boldsymbol{q})$ ▷ Compute forward kinematics
      if CHECKTASKSPACERANGE($\boldsymbol{x}$) **then**
        $Q \leftarrow Q \cup \{\boldsymbol{q}\}$ ▷ Keep $\boldsymbol{q}$
      end if
    end if
  until $|Q| =$ n
  return Q
end function

The approach with the rejection sampling based on a uniform distribution works fine for manipulators with toroidal $\mathbb{T}^n$ topology or hypercylinder-like topology $\mathbb{T}^r \times \mathbb{R}^p$. These topologies arise for manipulators with only 1-DoF joints. If, however, the manipulator also has spherical joints the uniform sampling of the joint parameters is no longer useful. On spherical topologies, uniform sampling of the joint parameters leads to denser distributions close to the poles.

Consider Figure 5.2 for a visualization. For better sampling in spherical topologies other distributions can be used. On $\mathbb{S}^3$ the uniform sampling on the surface can also be achieved when expressing the 3-sphere in terms of quaternions.



(a) Bad (b) Good

Figure 5.2.: Sampling on a sphere. Uniform sampling of the joint parameters does not lead to uniform distributions on the surface of the sphere. Figure adapted from [68].

Figure 5.3 shows example results from the rejection sampling scheme for the example robot with three rotational joints. The task space coordinate function is again chosen as the end effector position. The left column shows the selected samples and the right column rejected samples. Here, a value of $\alpha_{thrshld} = 0.2$ is chosen for detection of row-rank deficiency. The bounds for the joint values are chosen $q_i \in [-\pi, \pi]$ for each joint angle. While top row shows results without further task space constraints, the bottom row shows results for a task space range of $x \geqslant 0$ and $-0.5 \leqslant y \leqslant 0.5$. The value of the smallest singular value is indicated by the color code of the samples. Each of the plots additionally shows three example samples.

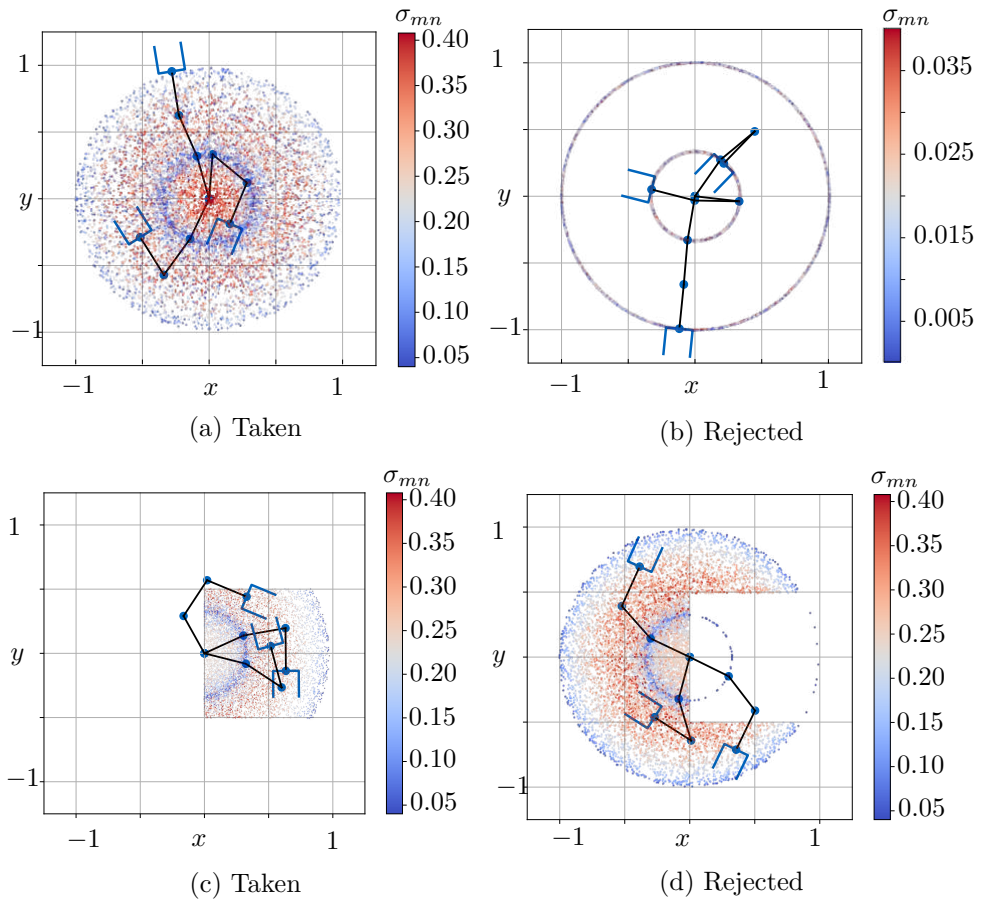
5.3. Implementation

TensorFlow [52] allows to automatically differentiate expressions symbolically and to generate fast code running on the GPU. The neural network model developed here is therefore implemented using TensorFlow. Automatic differentiation then allows to compute symbolic and fast expressions for the Jacobian $\boldsymbol{N}(\boldsymbol{q})$ and for the gradient of the loss function $\nabla_\theta L(\theta)$ used for optimization. Essentially, the overall loss function

$$L(\theta) = \frac{1}{2Kmr} \sum_{k=1}^{K} \sum_{i=1}^{m} \sum_{j=1}^{r} \left(\frac{\boldsymbol{v}_i^\top \boldsymbol{A} \boldsymbol{n}_j}{|\boldsymbol{A}\boldsymbol{v}_i| \cdot |\boldsymbol{n}_j|} \right)^2 + \frac{1}{2K \cdot \binom{r}{2}} \sum_{k=1}^{K} \sum_{i=1}^{r} \sum_{j=1}^{r} \left[\left(\frac{\boldsymbol{n}_i^\top \boldsymbol{A} \boldsymbol{n}_j}{|\boldsymbol{A}\boldsymbol{n}_i| \cdot |\boldsymbol{n}_j|} \right)^2 [i > j] \right],$$

can be efficiently implemented based on tensors. The elements summed up do not depend on each other and can be computed in parallel. The actual gradient descent of the parameters θ is implemented using the Adam [38] optimizer, which is also shipped with TensorFlow. Algorithm 2 summarizes the training process of the network.

At the beginning the parameters θ are randomly initialized. For the parameters of the weight matrices $\boldsymbol{W}$ Xavier initialization [22] is used and the initial biases $\boldsymbol{b}$ are set to zero. The training is separated into multiple n_{epochs} epochs. Each epoch starts by sampling new training data $\boldsymbol{q}_i$. The sampling algorithm is described in the subsequent section. Then, the task space Jacobians $\boldsymbol{J}(\boldsymbol{q}_i)$ and the metrics $\boldsymbol{A}(\boldsymbol{q}_i)$ are computed for each sample. By calling the Adam optimizer repeatedly for n_{steps} times, the parameters θ are updated. Finally, a new set of



(a) Taken (b) Rejected

(c) Taken (d) Rejected

Figure 5.3.: Rejection sampling results for an example manipulator. On the left the selected and on the right rejected samples are shown. While the bottom row shows results including additional task space constraints, the top row shows results without.

samples is generated for evaluation. The cost function is evaluated for those new samples. This can be used to detect if too many steps n_{steps} are used in the inner loop. Too many steps lead to overfitting on the training samples, which would result in a significantly higher evaluation- than training loss.

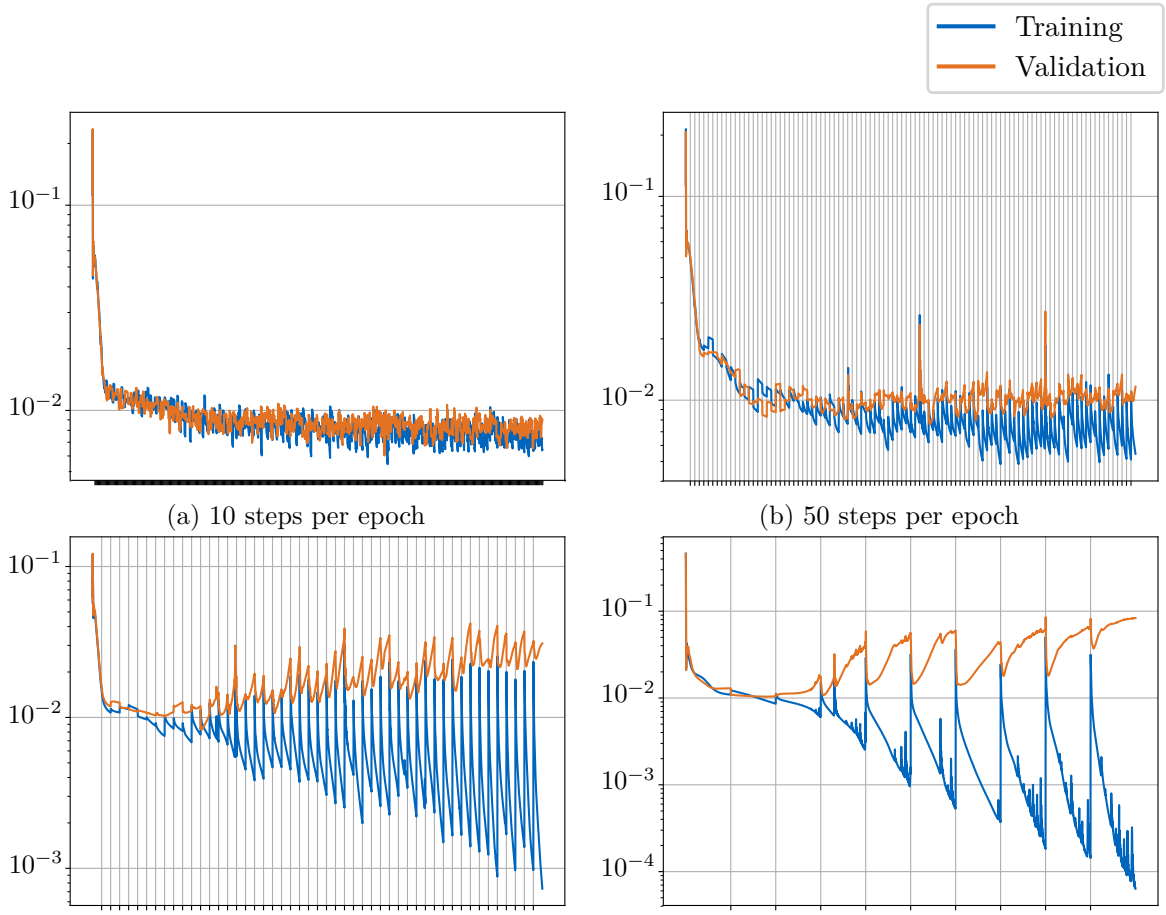
In order to find suitable hyperparameters for the training of the model, it is trained for different settings. The model was trained on the robot with four degrees of freedom and two task space coordinates. No further task space constraints were added in the joint configuration sampler, i.e. the rejection sampling only removed samples being too close to singularities. The number of neurons is set to $n_1 = 1024$ and $n_2 = 512$. The GPU memory allows to train on 10^4 samples per epoch without need to transfer training data between RAM and GPU memory. However, with that many training samples the effect of epoch-wise overfitting is small. In order to tune the hyperparameter for the steps per epoch, only 100 samples per epoch were generated in the training sample generation step. This only uses a small percentage of the available GPU memory. Using less samples amplifies the effect of epoch-wise overfitting. Then, the model was trained using four different settings for the steps per epoch. The total amount of steps $n_{total} = n_{epochs} \cdot n_{steps}$ was fixed to 5000. Among the different experiments the training was separated into different amounts of epochs.

Figure 5.4 shows the resulting training and validation loss curves. Note, that the x-axis of the

Algorithm 2 Training loop

function optimizeNetwork
  initialize parameters θ
  for epoch $\leftarrow 1, n_{\text{epochs}}$ **do**
    $Q \leftarrow$ sampleConfigurations(n) ▷ Perform rejection sampling
    compute $J = \{\boldsymbol{J}(\boldsymbol{q}_i) \,|\, \boldsymbol{q}_i \in Q\}$ ▷ Jacobian for each $\boldsymbol{q}_i \in Q$
    compute $A = \{\boldsymbol{A}(\boldsymbol{q}_i) \,|\, \boldsymbol{q}_i \in Q\}$ ▷ Corresponding metric for each $\boldsymbol{q}_i \in Q$
    for step $\leftarrow 1, n_{\text{steps}}$ **do**
      $\theta \leftarrow$ adamOptimizer(Q, J, A, θ) ▷ Gradient descent of $L(\theta)$ via Adam
    end for
    $Q_{val} \leftarrow$ sampleConfigurations(n) ▷ New samples for validation
    compute $J_{val} = \{\boldsymbol{J}(\boldsymbol{q}_i) \,|\, \boldsymbol{q}_i \in Q_{val}\}$ ▷ Validation Jacobians
    compute $A_{val} = \{\boldsymbol{A}(\boldsymbol{q}_i) \,|\, \boldsymbol{q}_i \in Q_{val}\}$ ▷ Validation metric
    evaluate $L(\theta \,|\, Q_{val}, J_{val}, A_{val})$
  end for
end function

graphs denotes steps, not physical time. In physical time, the first training (a) was slowest, while (d) was fastest. Once the training data have been transfered to the GPU memory, the steps are fast. However, sampling and copying the training data to the GPU takes time and more epochs required more sampling. For only few epochs and consequently more steps per epochs, the model tends to overfit on the training samples by the end of the epoch. This can be observed in (c) and (d). By the end of each epoch the validation loss is significantly higher than the training loss. As soon as a new epoch has started, the training loss jumps to a higher value and the validation loss quickly shrinks, i.e. the model is reverting the effect of overfitting from the previous epoch. Using more epochs fixes this issue. The results in (a) show basically no tendency of training and validation loss drifting apart. At 50 steps per epoch a good tradeoff between performance and training time is achieved. The total training time was 2.71min.



(a) 10 steps per epoch

(b) 50 steps per epoch

(c) 100 steps per epoch

(d) 500 steps per epoch

Figure 5.4.: Different amounts of steps per epoch result in different training performance. Each time a total amount of 5000 steps was used. Among the different trials the training steps were separated into more or less epochs. When there are too few epochs the model tends to overfit on training data.

6

Geometric and Closed-Loop Evaluation

The neural network based approach provides a framework for finding an approximate function φ of OSMCs for a given manipulator and task space coordinate function $\boldsymbol{x} = f(\boldsymbol{q})$. For the training procedure, only the Jacobian $\boldsymbol{J}(\boldsymbol{q})$ and the (inverse) mass matrix $\boldsymbol{M}^{-1}(\boldsymbol{q})$ is required. The training procedure was executed for different planar robots.

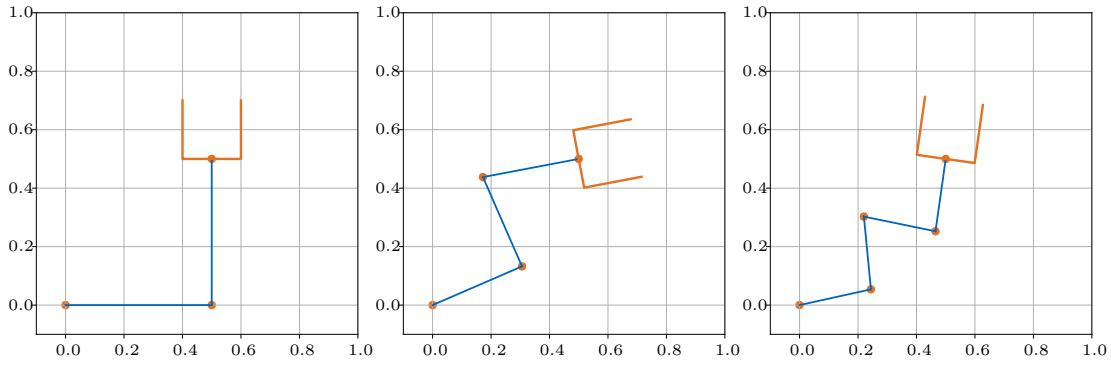
This chapter shows the results from the evaluation of the final functions $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$. Three different evaluation types are shown:

1. **Geometric:** First, some models are evaluated geometrically. When choosing the identity matrix as metric, the foliations are required to be orthogonal in a Euclidean sense. The effect of the theorems 2 to 5 will be directly observable when visualizing the results. It will be seen that the model can, as expected, not be globally valid. However, the local validity regions are large.
2. **Kinematic:** Secondly, the function φ is used in closed-loop on a first-order equation. In particular, the behavior is analyzed on a kinematic level only, i.e. the true physical dynamics of the manipulator is not considered. This allows to analyze the performance of the model without the presence of coupling terms like Coriolis/centrifugal forces and the term corresponding to the change of the Jacobian.
3. **Dynamical:** Finally, the true multi-body dynamics is used in closed-loop. This requires a model trained based on the inverse mass matrix as metric. This simulates how the controller would perform on a torque-controlled physical robot. Using the physical multi-body dynamics, the effects of the coupling terms can be observed.

Three different planar robots with different amounts of joints are used throughout this chapter. Figure 6.1 shows the manipulators. Chapter B in the appendix shows how the forward kinematics and the Jacobians are computed. Table 6.1 provides a guide to the remainder of this chapter.

Table 6.1.: Overview of the evaluation sections.

#	Type	DoF	Task Space	Metric	Network Settings	Section
1	Geometric	2	x	$\mathbf{I}$	$n_1 = 512$, $n_2 = 128$	6.1.1
2	Geometric	3	x, y	$\mathbf{I}$	$n_1 = 512$, $n_2 = 256$	6.1.2
3	Geometric	3	x	$\mathbf{I}$	$n_1 = 512$, $n_2 = 256$	6.1.3
4	Kinematic	4	x, y	$\mathbf{I}$	$n_1 = 1024$, $n_2 = 512$	6.2
5	Dynamical	3	x	$\boldsymbol{M}^{-1}(\boldsymbol{q})$	$n_1 = 512$, $n_2 = 256$	6.3



(a) $n = 2$ (b) $n = 3$ (c) $n = 4$

Figure 6.1.: The robots used for evaluation. The total link lengths always sum up to 1.

6.1. Geometric Evaluation

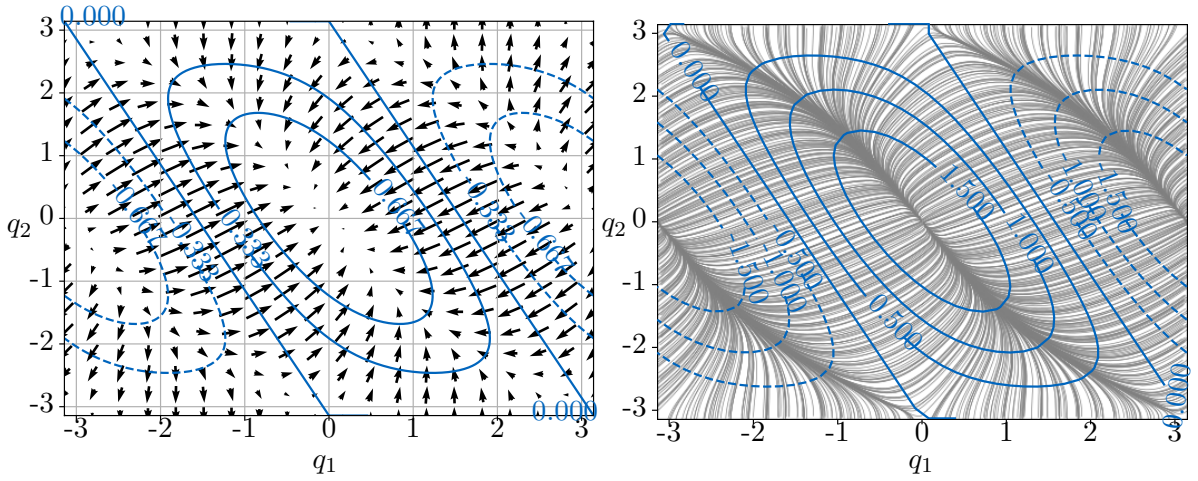
This section shows training results for a selection of manipulators and task space coordinates. Throughout these examples the model is trained using the identity matrix as metric for orthogonality, i.e. the model is not trained for dynamical consistency. This is beneficial for first analysis of the training, because the condition becomes geometrical orthogonality of vectors, lines and surfaces. These can be visualized in diagrams and interpreted intuitively.

6.1.1. 2 Degrees of Freedom, 1 Task Space Coordinate

First, a trained model for a manipulator with two degrees of freedom is analyzed. The task space coordinate is the x-component of the end-effector position. The link lengths are set to 0.5 units for each link. Figure 6.2(a) shows the forward kinematics function of the manipulator. The blue lines show isolines of the function, i.e. on every point on the isolines the forward kinematics $x = f(\boldsymbol{q})$ is constant. Additionally, the black arrows show the Jacobian $\boldsymbol{J}(\boldsymbol{q})$ on a regular grid. For this manipulator the Jacobian $\boldsymbol{J}(\boldsymbol{q})$ is a single row vector with two elements. The rows of the Jacobian correspond to the direction of greatest ascent in x. Consequently, the vectors are normal to the isolines.

For identity metric between the rows of $\boldsymbol{J}(\boldsymbol{q})$ and $\boldsymbol{N}(\boldsymbol{q})$, the vectors must be orthogonal in a Euclidean sense. Therefore, fitting line integrals to the vector field given by $\boldsymbol{J}(\boldsymbol{q})$ yields isolines of the desired OSMC. Figure 6.2(b) shows many of those line integrals for the manipulator with two degrees of freedom. These lines show manifolds, where the desired function φ is known to be constant. However, the actual value of the OSMC is not known from this analysis. Nevertheless, these lines can be used to validate the results of the neural network approach. The expected result is, that visualizing the isolines of a trained $\xi = \varphi_\theta(\boldsymbol{q})$ resembles the lines shown in Figure 6.2(b).

A neural network model has been trained on the entire joint space from $-\pi$ to π for both joints. No task space region restriction has been added, i.e. every sample, except for configurations close to singularities, is taken for the training. The model achieves a training loss of $5.4 \cdot 10^{-3}$ after 5000 training steps organized into 100 epochs. It has $n_1 = 512$ and $n_2 = 128$ neurons in the first and second hidden layer, respectively. For a first evaluation of the model, a set of 10000 joint configurations have been sampled. Figure 6.3(a) shows a histogram of the angle between the Jacobians $\boldsymbol{J}(\boldsymbol{q}) = \boldsymbol{v}^\top(\boldsymbol{q})$ and $\boldsymbol{N}(\boldsymbol{q}) = \boldsymbol{n}^\top(\boldsymbol{q})$ for the sampled test configurations. Among the test configurations are also samples close to singularities. The self-motion manifolds

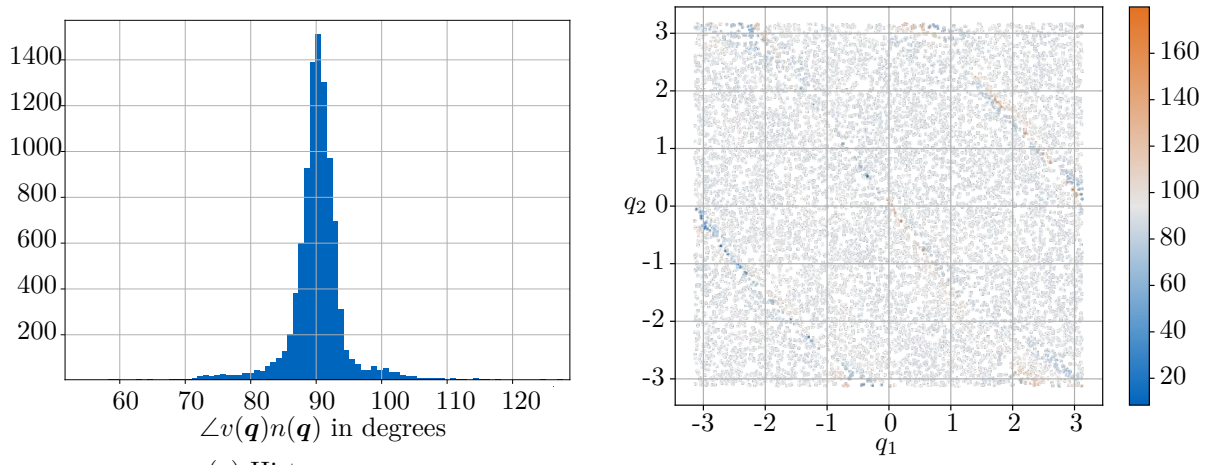


(a) Task space and Jacobians

(b) Line Integrals

Figure 6.2.: Simplest redundant manipulator with two joints.

shown as isolines in Figure 6.2 are homeomorphic to $\mathbb{S}^2$. Consequently, the function $\xi = \varphi(\boldsymbol{q})$ can not find a globally unique parametrization on the isolines (Theorem 4). Therefore, no perfect model can be expected as the model is trained on the entire joint space. The histogram shows that at most of the test configurations, the rows of the task- and self-motion Jacobian are orthogonal. However, there are also configurations where the orthogonality is off by some degrees. Figure 6.3(b) shows the location of the sampled test configuration. Additionally, the color of the scatter plot denotes the angle between the Jacobians at that configuration. The performance of the trained model is worse in regions which are singular or close to singular. For a color-coded map of the singular values of $\boldsymbol{J}(\boldsymbol{q})$ see Figure A.1(a).



(a) Histogram

(b) Scatter of Angles

Figure 6.3.: Angles between the Jacobians of the trained network of the example manipulator with two joints. On the left the histogram shows the angles distribution for randomly sampled test configurations. The peak is at 90°. On the right, the position of these samples in joint space is shown, while the color encodes the angle at that position. The distribution of the residual angles is position dependent.

Finally, Figure 6.4 shows the resulting isolines of the trained model combined with the isolines of the task space forward kinematics. For a larger version including labels on the isolines

consider Figure A.2 on page II. As expected the isolines of the trained OSMC function $\xi = \varphi(\boldsymbol{q})$ resemble the integrated line integrals of the task space Jacobian in the regions, where the model performs well. Because the single number ξ can never parameterize the self-motion manifolds with sphere topology, the model is not globally valid. Close to the singularities, the approximation of the integrated isolines becomes worse. Additionally, the regions where all the line integrals converge, are not properly imaged in the model. In these regions the ξ-value would need to vary very quickly, which the smooth neural network can not resemble properly. Note that the integrated isolines from Figure 6.2(b) do not provide any label, i.e. ξ-value, on the lines. The trained model, however, does. This is essential in order to be used for control approaches. The model can be used for the developed impedance control in the large white regions.

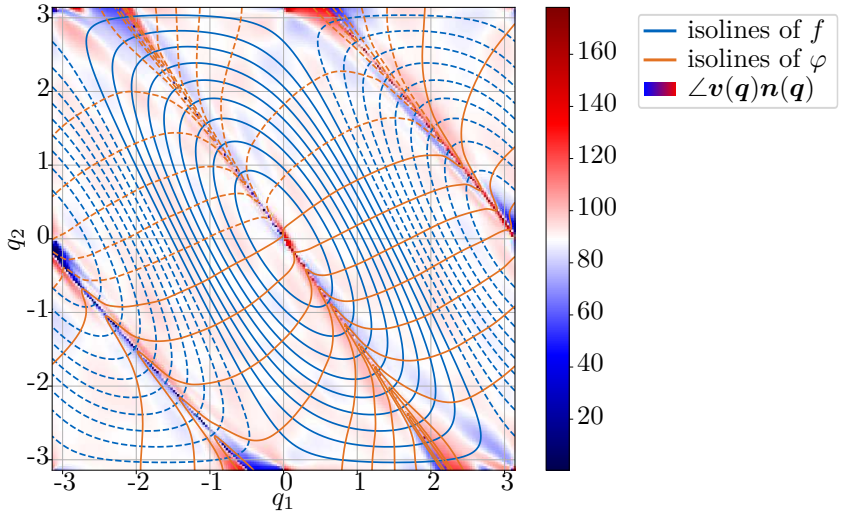


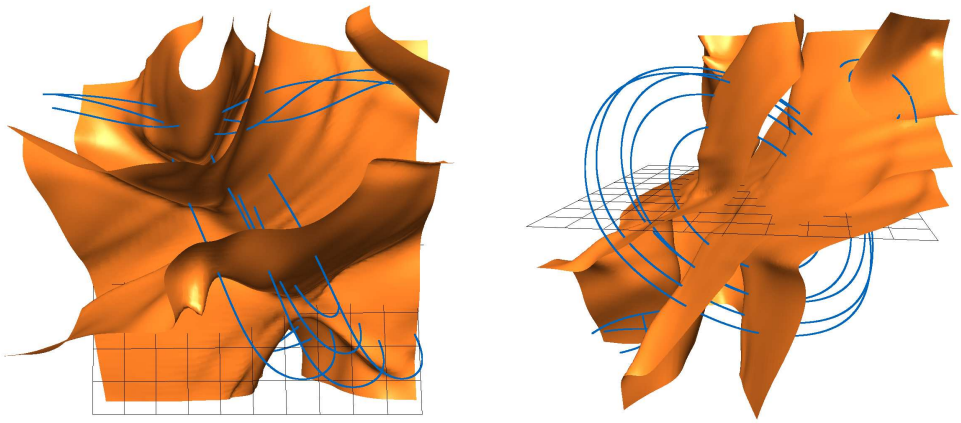
Figure 6.4.: Results of the training process on a manipulator with two degrees of freedom and task space coordinate. The blue lines show isolines of the task space forward kinematics and the orange lines show isolines of the resulting function φ of the training process. In the background, the color code shows the performance of the model in terms of the angle between the Jacobians.

6.1.2. 3 Degrees of Freedom, 2 Task Space Coordinates

For three degrees of freedom (Figure 6.1(b)) and choosing the end-effector position as task space coordinates, the self-motion manifolds are curves in three-dimensional space. The degree of redundancy is $r = 1$ and therefore there is one single OSMC $\xi_1 = \varphi(\boldsymbol{q})$. Basically, the leaves of the task space foliation show two different geometries. This can be seen in Figure 4.5. Further, [6] explains why these different types of geometry arise. For task space positions close to the origin, one of the joints can continuously rotate into the same direction. In contrast, this is no longer possible for larger task space values and all of the joints need to rotate back and forth on the self-motion manifold. Note that all of the self-motion manifolds are closed curves on the toroidal topology.

Figure 6.5 shows training results for this setting. The orange surface denotes iso-surfaces of the ξ_1-coordinate and the blue curves show some exemplary leaves of the task space foliation. It can be seen that the surfaces are normal to the self-motion manifolds shown in blue. This example also shows that the coordinates are not global. Each surface is intersected twice by each of the self-motion manifolds. This is a consequence of the Bursuk-Ulam theorem (Thm. 4):

there is no bijective mapping from $\mathbb{S}^n$ to $\mathbb{R}^n$.



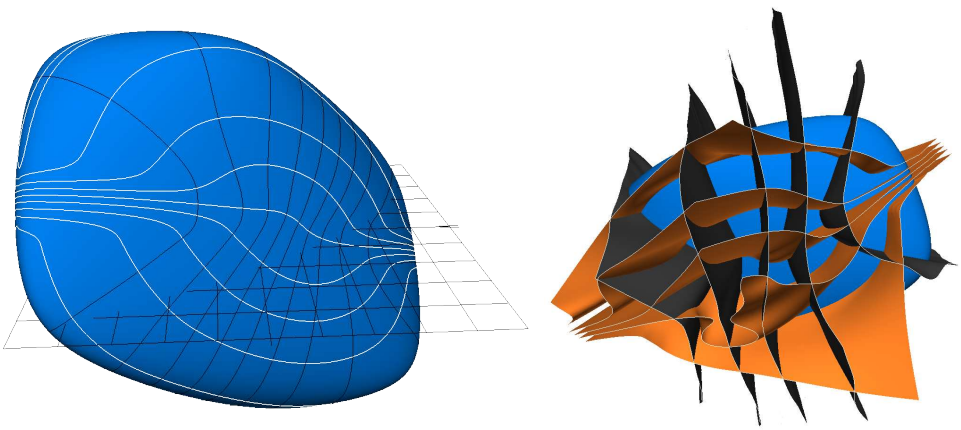
(a) View 1 (b) View 2

Figure 6.5.: Visualization of the training results for three degrees of freedom and two task space coordinates. The blue lines show some self-motion manifolds and the orange surfaces show isosurfaces of $\xi_1 = \varphi(\boldsymbol{q})$.

6.1.3. 3 Degrees of Freedom, 1 Task Space Coordinate

Removing the y-coordinate from the task space coordinate choice, leaves two-dimensional self-motion manifolds. These have already been shown in Figure 4.10. For $x > \frac{1}{3}$, the manifolds have a two-sphere topology. One of those is shown in Figure 6.6. On the left, the black and white lines show isolines of the ξ_1 and ξ_2-coordinate. As predicted by the hairy ball theorem, the Jacobian $\boldsymbol{N}(\boldsymbol{q})$ must vanish at least at one point for both rows. When considering the exact values of the coordinates, both coordinates show exactly one minimum and one maximum on the manifold. At the extrema, the Jacobian $\boldsymbol{N}(\boldsymbol{q})$ becomes singular. Figure A.3 in the appendix shows the two coordinates color-coded on the manifold.

Dupin's theorem is also observable in the visualization. The lines of intersection coincide with the lines of principal curvature. This can be observed in the outer regions of Figure 6.6(a). In Figure 6.6(b) the model is visualized by showing the isosurfaces.



(a) Isolines of φ (b) Foliation view

Figure 6.6.: Results for geometric analyisis for $n = 3$ and $m = 2$. 3D View here: http://thesis.aaarne.de/n3-m1-orthfol.stl

6.2. Kinematic Evaluation

For kinematic analysis the model is tested on first-order dynamics. The true second-order multi-body dynamics of the manipulator introduces couplings between the coordinates via velocity-dependent terms. In particular, the velocity $\dot{\boldsymbol{q}}$ couples into the task space dynamics via Coriolis and centrifugal forces as well as via the change of the Jacobian $\dot{\boldsymbol{J}}$. The acceleration $\ddot{\boldsymbol{x}}$ is governed by

$$\ddot{\boldsymbol{x}} = \boldsymbol{M}_x^{-1}\boldsymbol{f} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} - \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{C}\dot{\boldsymbol{q}}\,. \tag{6.1}$$

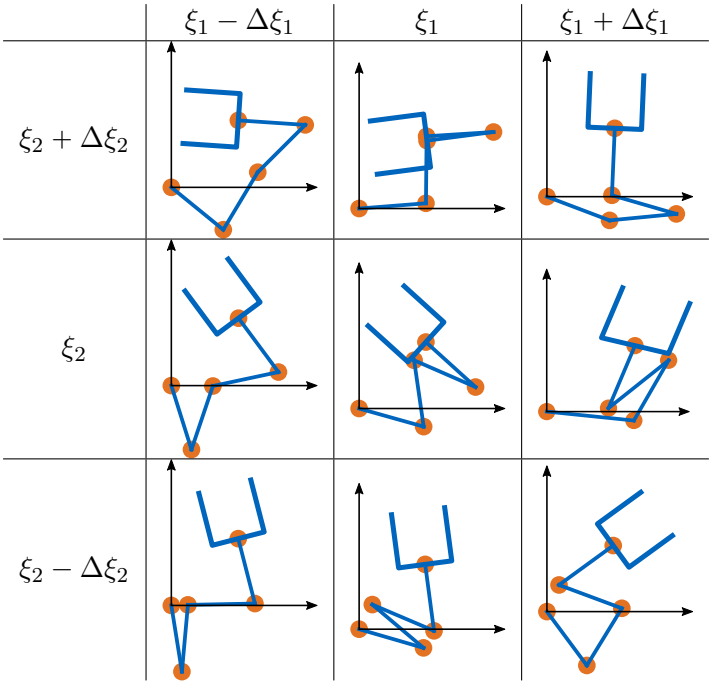
In order to analyze the model without these interfering effects, first-order dynamics is used in closed-loop. This corresponds to analysis on only a kinematic level. Particularly, it is assumed joint velocities can be commanded directly

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^\top \boldsymbol{K}_x(\boldsymbol{x}_d - f(\boldsymbol{q})) + \boldsymbol{N}^\top \boldsymbol{K}_\xi(\boldsymbol{\xi}_d - \varphi(\boldsymbol{q}))\,. \tag{6.2}$$

Essentially, the true second-order dynamics of the manipulator (3.16) is replaced by the pseudo-dynamics $\boldsymbol{\tau} = \dot{\boldsymbol{q}}$. This choice of dynamics equation is also used in numerical inverse kinematics, where it is known as the transposed Jacobian approach [81, Sec 3.7.2].

The neural network model $\boldsymbol{\xi} = \varphi_\theta(\boldsymbol{q})$ is trained for a manipulator with four degrees of freedom. The task space coordinates are the x- and y-coordinate of the position of the end-effector. Initially, the configuration of the manipulator is set to the configuration shown in Figure 6.1(c). This configuration $\boldsymbol{q}_0$ is determined by computing transpose Jacobian based numerical inverse kinematics for $\boldsymbol{x}_0 = [0.5, 0.5]^\top$. Then, the OSMCs $\boldsymbol{\xi}_0 = \varphi(\boldsymbol{q}_0)$ is evaluated. In the beginning of the simulation, the desired values are set to $\boldsymbol{\xi}_d = \boldsymbol{\xi}_0$ and $\boldsymbol{x}_d = \boldsymbol{x}_0$, respectively. Table 6.2 shows the resulting configurations upon variation of the self-motion coordinates. Afterwards, the desired values are updated by a predefined schedule. In particular, jumps in all directions of the task coordinates and OSMCs are commanded.

Table 6.2.: Configurations for the same task space position and different values for the self-motion coordinates. Columns: variation of ξ_1. Rows: variation of ξ_2.



The first-order dynamics (6.2) is simulated using an explicit fourth-order Runge-Kutta method [26]. The desired values $\boldsymbol{x}_d$ and $\boldsymbol{\xi}_d$ become functions of time and are determined by the experiment schedule. Explicit integration schemes are suitable for integration of non-stiff differential equations. The stiffness of an ordinary differential equation is the ratio of the fastest and the slowest time constant in the linearized system. This corresponds to the quotient of the largest and smallest magnitude of the eigenvalues [5]. In order to check if an explicit integration scheme is suitable for the differential equation, the numerical stiffness needs to be evaluated. Assuming stationary Jacobians, the linearized system can be written as

$$\Delta\dot{\boldsymbol{q}} = -\boldsymbol{J}^\top \boldsymbol{K}_x \boldsymbol{J} \Delta\boldsymbol{q} - \boldsymbol{N}^\top \boldsymbol{K}_\xi \boldsymbol{N} \Delta\boldsymbol{q} = \underbrace{-\boldsymbol{J}_a^\top \begin{bmatrix} \boldsymbol{K}_x & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{K}_\xi \end{bmatrix} \boldsymbol{J}_a}_{\tilde{\boldsymbol{A}}(\boldsymbol{q})} \Delta\boldsymbol{q}\,. \tag{6.3}$$

The block diagonal matrix of the gains is positive definite. Therefore, the quadratic form is strictly positive and the system matrix $-\tilde{\boldsymbol{A}}(\boldsymbol{q})$ is negative definite. Consequently, the dynamics are stable. Figure 6.7 shows a histogram of the numerical stiffness ratio for randomly sampled configurations $\boldsymbol{q}$. The real part of the individual eigenvalues is shown in Figure A.4. The differential equation is sufficiently numerically nonstiff and the explicit integration scheme can be used. The eigenvalues are sorted in descending order according to their magnitude.

Figure 6.8 shows the results. The top and mid diagram shows the coordinate functions $\boldsymbol{x} = f(\boldsymbol{q})$ and $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$, respectively. Additionally, the dashed lines in those diagrams show the desired values $\boldsymbol{x}_d$ and $\boldsymbol{\xi}_d$. On the bottom the joint angles are shown. In the first $30s$ jumps in the task space coordinate are commanded. The coordinate functions show typical first-order transients with different time constants. Note that the task space coordinates are not decoupled. Upon jumps in one coordinate, significant coupling into the other coordinate can be observed.

Subsequently, jumps in the self-motion coordinates are commanded. In contrast to the task coordinates, the OSMCs show good decoupling with respect to the task coordinates and to each other. Due to the decoupling there are only small disturbances in the task space coordinates. Additionally, the second term in the loss function (5.13) also achieves decoupling between mutual OSMCs. Finally, starting from $t = 60s$, the task space position is changed and again jumps in the self-motion coordinates are commanded. In other words, by changing the task space position of the manipulator, the system is moved onto another leaf of the task space foliation. Because of the foliation structure, the decoupling also works there. The joint angles on the bottom show that also the motions corresponding to jumps in the self-motion coordinate result in significant reconfiguration of the manipulator.

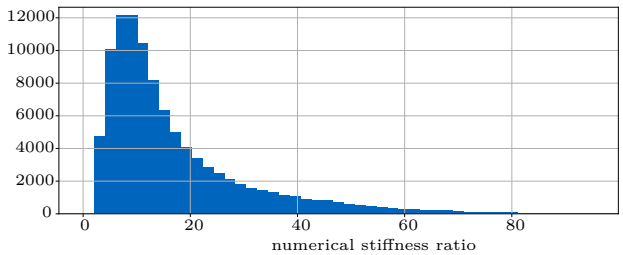
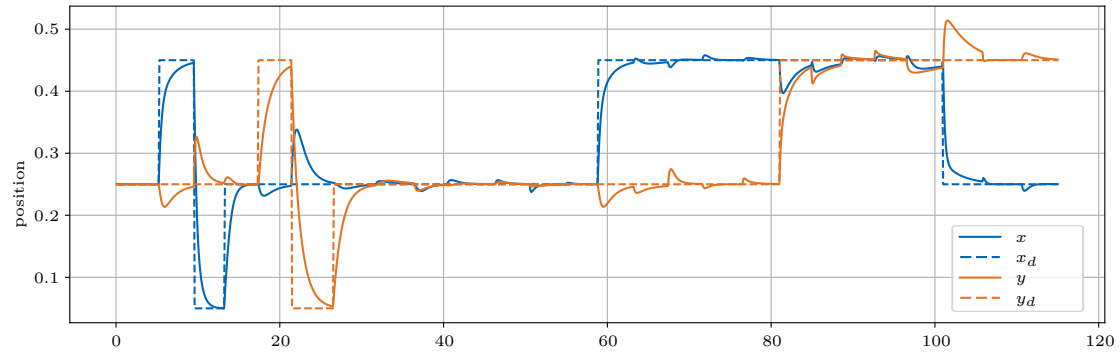
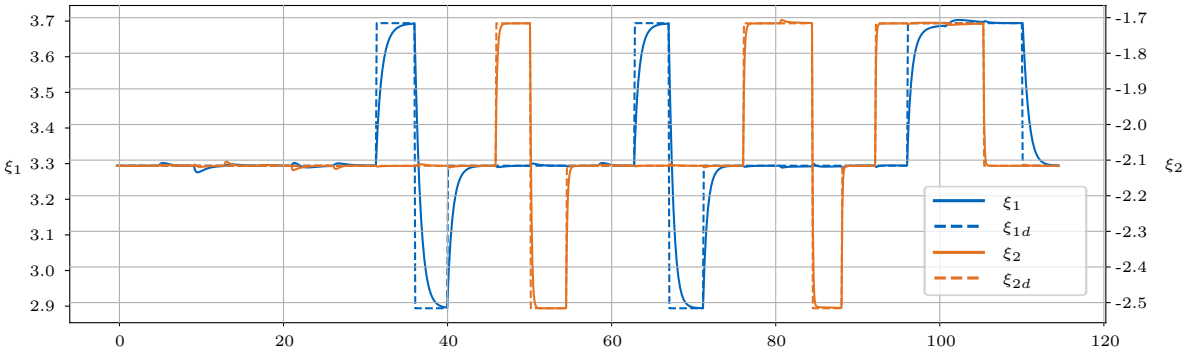


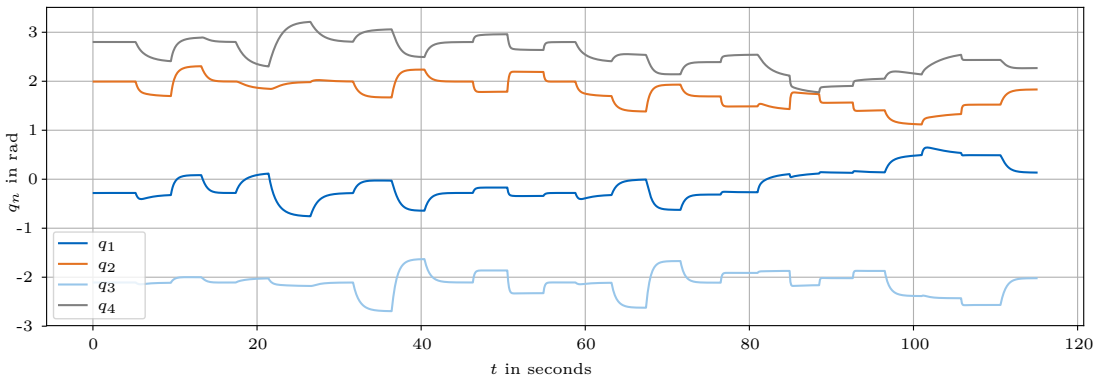
Figure 6.7.: Histogram of numerical stiffness ratio of the first order dynamics for the kinematic analysis.



(a) Task Space



(b) Self-Motion Space



(c) Joint Space

Figure 6.8.: Kinematic evaluation with a manipulator with four joints. The position of the end-effector is the task space coordinate. Top: the values of the task coordinates over time. The dashed value denote the desired value. Mid: the values of the self-motion coordinate function and corresponding desired values. Bottom: the joint angles. The diagrams show the decoupling between the coordinates. While the task coordinate show large mutual couplings, the OSMCs are well decoupled from the task coordinates and from each other. Animated manipulator of the experiment here: http://thesis.aaarne.de/kinematic.gif

6.3. Closed-Loop Dynamical Evaluation

For dynamic evaluation, a planar robot with three joints is used. A dynamic model (3.16) is derived for this manipulator. The mass is assumed to be equally distributed on the links, i.e. the center of mass is at the center of the links. Consequently, the moments of inertia are computed using $I_i = \frac{1}{3} m_i l_i^2$.

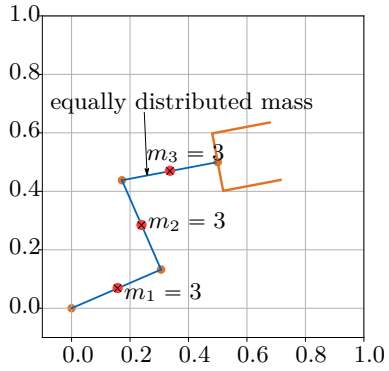


Figure 6.9.: Manipulator with dynamic parameters. The center of masses are assumed to be at the center of the links and the mass is equally distributed on the links.

The task space is chosen as the x-component of the end-effector. The dynamics equation (3.16) as well as the controller

$$\boldsymbol{\tau} = -\boldsymbol{J}_a^\top \left[\boldsymbol{K}_a \left(\begin{bmatrix} f(\boldsymbol{q}) \\ \varphi(\boldsymbol{q}) \end{bmatrix} - \begin{bmatrix} \boldsymbol{x}_d \\ \boldsymbol{\xi}_d \end{bmatrix} \right) - \boldsymbol{D}_a(\boldsymbol{q}) \boldsymbol{J}_a \dot{\boldsymbol{q}} \right] + g(\boldsymbol{q}) \tag{6.4}$$

are implemented in Simulink. The OSMCs $\varphi(\boldsymbol{q})$ as well as the last two rows are provided by the trained neural network. The network is reimplemented in Matlab and the optimized parameters from TensorFlow are imported. This time, the neural network model is trained using the inverse mass matrix as metric. For computation of the damping matrix $\boldsymbol{D}_a(\boldsymbol{q})$ the damping design equation (3.29) is used.

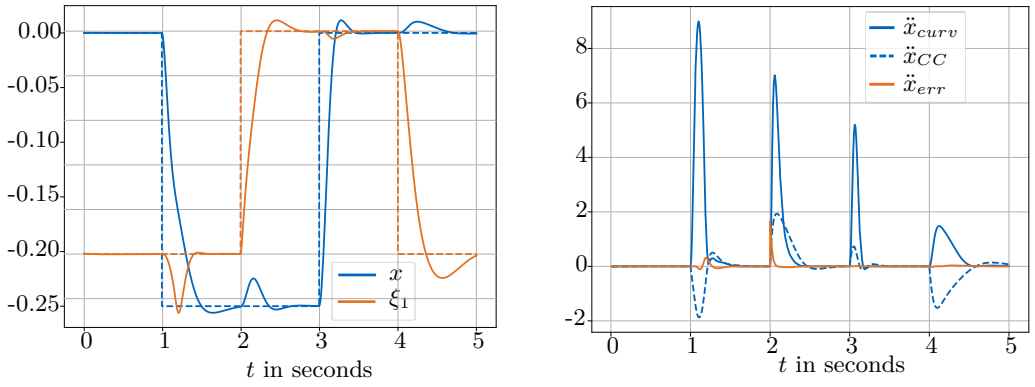
First, a set of two interleaved jumps on the x- and the ξ_1-coordinate is commanded. The damping ratio is set to $\zeta = 0.7$. Figure 6.10 shows the results.

On the left the values for the x and ξ_1 coordinate, as well as their desired values are shown. Generally, the task space acceleration depends on the following terms

$$\ddot{\boldsymbol{x}} = \boldsymbol{M}_x^{-1} \boldsymbol{f} + \underbrace{\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^\top \boldsymbol{\eta}}_{\ddot{\boldsymbol{x}}_{err}} + \underbrace{\dot{\boldsymbol{J}}\dot{\boldsymbol{q}}}_{\ddot{\boldsymbol{x}}_{curv}} - \underbrace{\boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{C}\dot{\boldsymbol{q}}}_{\ddot{\boldsymbol{x}}_{CC}} . \tag{6.5}$$

The diagram on the right shows different disturbing effects on the task space acceleration $\ddot{x}$. In particular, the following disturbing task space accelerations are shown:

- **Self-Motion Manifold Curvature:** The curvature of the self-motion manifolds generates a task space acceleration via the term $\ddot{x}_{curv} = \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}$. This disturbing acceleration is shown by the solid blue lines.
- **Coriolis and Centrifugal Forces:** The task space acceleration resulting from Coriolis and centrifugal forces $\ddot{x}_{CC} = \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{C}\dot{\boldsymbol{q}}$ are depicted by a dashed blue line.
- **Training Error:** Finally, the neural network only provides an approximation of $\varphi(\boldsymbol{q})$. The task space acceleration which are due to training errors $\ddot{x}_{err} = \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{N}^\top\boldsymbol{\eta}$ are shown in orange.

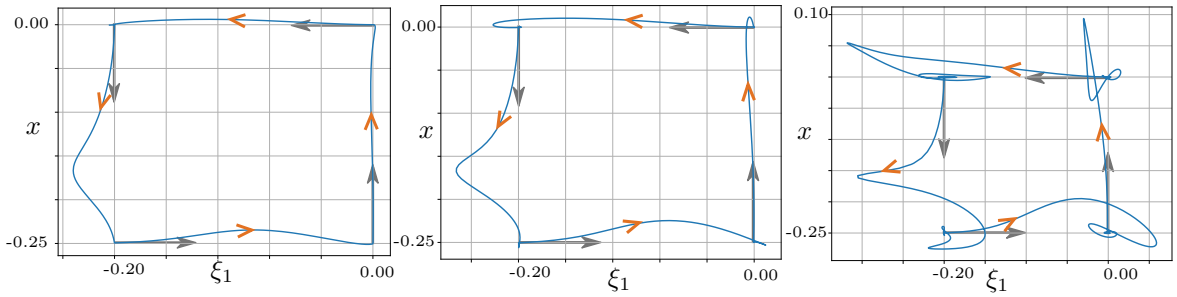


(a) Coordinates (b) Disturbing task space accelerations

Figure 6.10.: Closed-loop behavior of manipulator and augmented impedance controller for a damping ratio of $\zeta = 0.7$. On the left: actual and desired values of the coordinates. On the right: the effects on the task space acceleration.

The decoupling does not perform as well as in the kinematic analysis. However, when considering the different effects on the task space acceleration in diagram (b), the main contribution is the term $\dot{\boldsymbol{J}}\dot{\boldsymbol{q}}$. The accelerations due to training error are comparatively small. The velocity $\dot{\boldsymbol{q}}$ couples in to the task space dynamics via the change of the Jacobian $\dot{\boldsymbol{J}}$. Consequently, at low velocities the coordinates show decoupled behavior.

Figure 6.11 shows the trajectory of the manipulator for the interleaved jumps in the $x\xi_1$-plane. The diagram (b) shows the trajectory corresponding to the experiment for $\zeta = 0.7$. On the left and right experiments with the same stiffness parameters, but different damping ratios are shown. Overdamped systems have slower transients and velocities are lower. For a higher damping ratio of $\zeta = 1.0$ the decoupling performs better. In contrast, underdamped systems generate higher velocities and the disturbing accelerations based on the change of Jacobian and the Coriolis and centrifugal force generate large disturbances. This is shown on the right for a damping ratio of $\zeta = 0.3$. In each of the diagrams, the trajectories initially tend towards the same direction. This is shown by the gray vector in the diagrams. As soon as the manipulator accumulates velocity, the trajectories deviate from the straight path. The lower the damping ratio, the higher the joint velocity $\dot{\boldsymbol{q}}$ and the higher the disturbances.



(a) $\zeta = 1.0$ (b) $\zeta = 0.7$ (c) $\zeta = 0.3$

Figure 6.11.: Trajectory of the manipulator for interleaved jumps in the x- and ξ_1-coordinate. The trajectories are shown in the $x\xi_1$-plane. The trajectories result from closed-loop dynamics with the augmented impedance controller for different damping ratios.

So far, only the ξ_1-coordinate was used for the evaluation. The second OSMC ξ_2 was kept at 0 using the impedance controller. Figure 6.13 shows the results of the closed-loop simulation for a longer experiment using all the coordinates for a damping ratio $\zeta = 0.7$. While the topmost diagram shows the task space coordinate, the second diagram shows the self-motion coordinates. Diagram (c) shows the disturbing task space accelerations and (d) the joint angles.

In the first 14 seconds, the desired value for the task space coordinate is kept constant at $x = 0$. Simultaneously, jumps in the self-motion coordinates are commanded. The trajectory of the manipulator in joint space is visualized on the manifold in Figure 6.12.

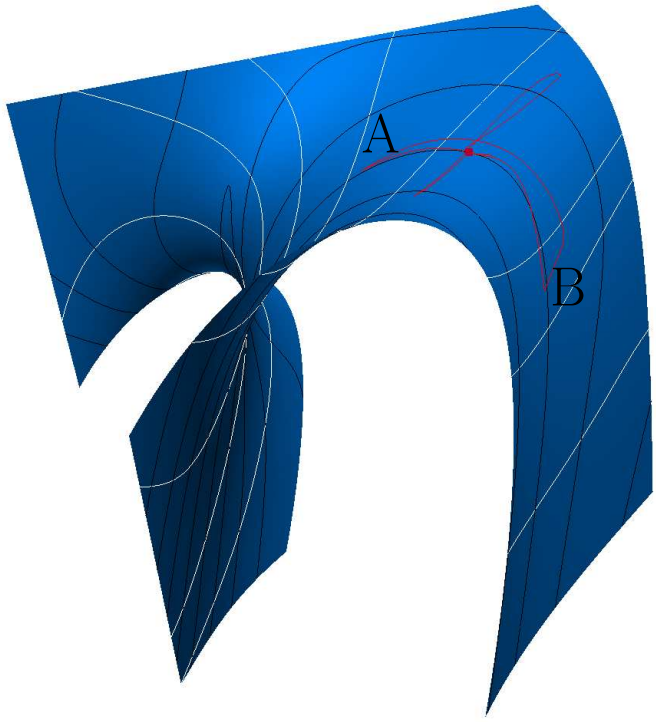


Figure 6.12.: Trajectory on manifold for $\zeta = 0.7$. The trajectory shows the first $14s$ of the simulation shown in Figure 6.13. Video the of the animated trajectory on different leaves of the foliation on the entire time series here: http://thesis.aaarne.de/traj-on-manifold.mp4

Especially in regions of high curvature of the self-motion manifolds the Jacobian changes quickly. Consider the commanded jump in ξ_1 at t_{AB} denoted in Figure 6.13. This corresponds to a jump from configurations A to B shown in Figure 6.12. For the jump from configuration A to configuration B, the trajectory passes through a region of high curvature. Therefore, the change of the Jacobian $\dot{\boldsymbol{J}}$ is comparatively large. The manipulator accumulates a velocity $\dot{\boldsymbol{q}}$ that is then coupled in to the task space dynamics. This leads to a delayed disturbance in the task space position x.

As this curvature-based disturbance is proportional to the joint velocity $\dot{\boldsymbol{q}}$, the effect is larger at higher velocities. Keeping the stiffness $\boldsymbol{K}_a$ constant and changing the damping ratio ζ, results in higher or lower velocities. Decreasing the damping ratio to $\zeta = 0.3$ increases the joint velocities and oscillations arise on the transients. At $\zeta = 0.3$ the disturbance on the jump from A to B is larger, whereas at $\zeta = 1.0$ it becomes smaller. This is due to higher joint velocities at $\zeta = 0.3$ and lower joint velocities for the overdamped behavior at $\zeta = 1.0$. For results of the experiments for different damping ratios, consider Figures A.5 and A.6 in the appendix.

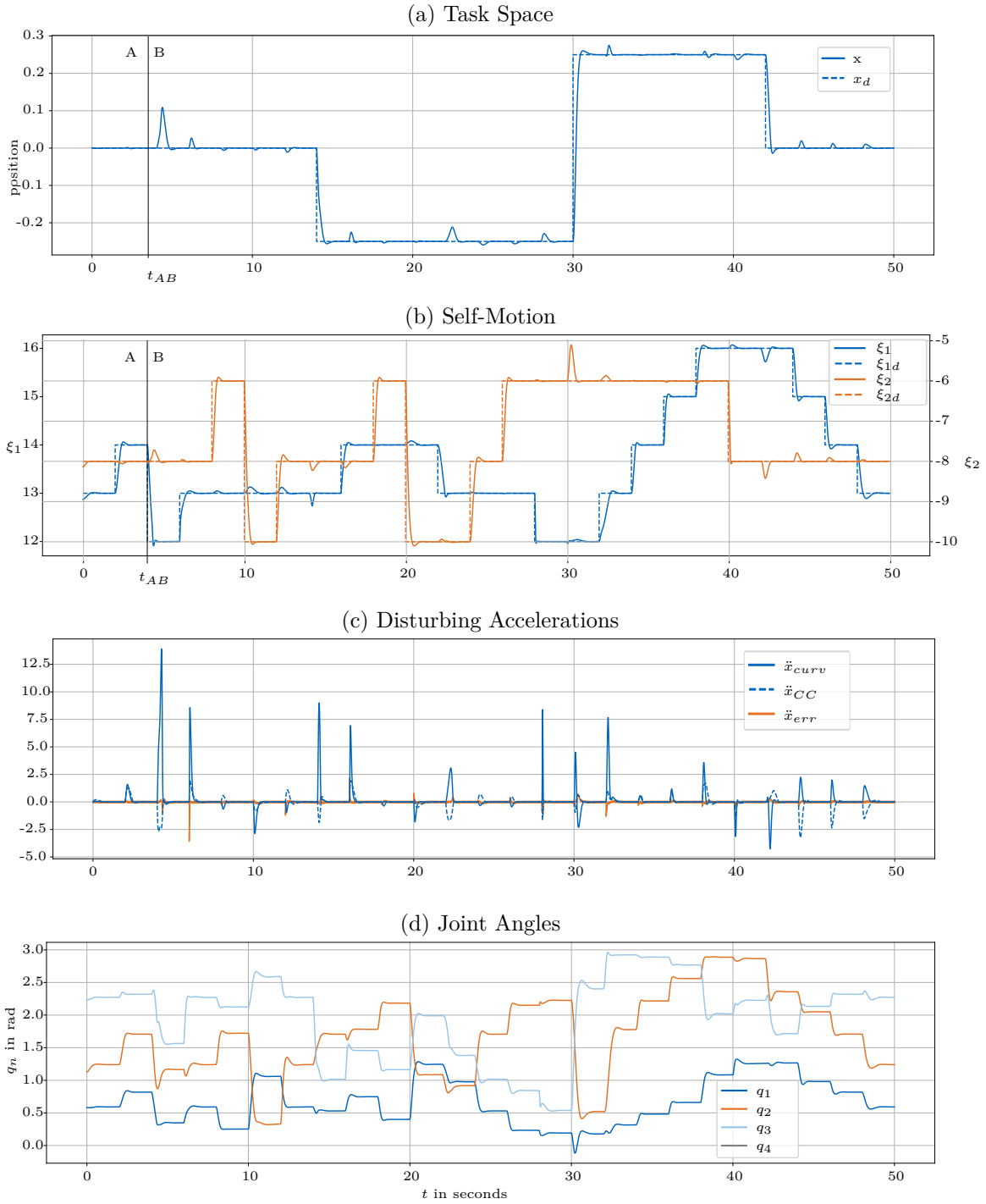


Figure 6.13.: Simulation results for closed-loop behavior of manipulator and the augmented control scheme using neural-network based coordinates. The results are for a damping ratio of $\zeta = 0.7$. Animated robot here: http://thesis.aaarne.de/anim-07.gif

6.3.1. Feed-Forward Controller

In this section a feed-forward controller is added to the control scheme in order to compensate for the disturbing accelerations resulting from the curvature of the self-motion manifold and

the Coriolis and centrifugal forces. Here, the bold $\boldsymbol{x}$ denotes augmented coordinates

$$\boldsymbol{x} = \begin{bmatrix} f(\boldsymbol{q}) \\ \varphi(\boldsymbol{q}) \end{bmatrix} = \begin{bmatrix} x \\ \xi_1 \\ \xi_2 \end{bmatrix} = f_a(\boldsymbol{q})\,. \tag{6.6}$$

Suppose a desired trajectory specified by $\boldsymbol{x}_d$, $\dot{\boldsymbol{x}}_d$ and $\ddot{\boldsymbol{x}}_d$ is provided. As before, the feedback controller is the impedance controller in augmented coordinates

$$\boldsymbol{f}_{FB} = \boldsymbol{K}_a \left[\boldsymbol{x}_d - f_a(\boldsymbol{q})\right] + \boldsymbol{D}_a(\boldsymbol{q}) \left[\dot{\boldsymbol{x}}_d - \boldsymbol{J}_a \dot{\boldsymbol{q}}\right]\,. \tag{6.7}$$

The augmented Jacobian $\boldsymbol{J}_a$ is a stack of $\boldsymbol{J}$ and $\boldsymbol{N}$

$$\boldsymbol{J}_a = \begin{bmatrix} \boldsymbol{J} \\ \boldsymbol{N} \end{bmatrix}, \tag{6.8}$$

which is always a square matrix. Together with the feed-forward controller, the closed-loop dynamics is

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{C}\dot{\boldsymbol{q}} + g(\boldsymbol{q}) = \boldsymbol{J}_a^\top \left[\boldsymbol{f}_{FF} + \boldsymbol{f}_{FB}\right] + g(\boldsymbol{q})\,. \tag{6.9}$$

In order to design the feed-forward controller $\boldsymbol{f}_{FF}$ perfect tracking is assumed. Therefore, consider the case for $\boldsymbol{f}_{FB} = 0$. Then, equation 6.9 is pre-multiplied by $\boldsymbol{J}_a \boldsymbol{M}^{-1}$ to obtain

$$\boldsymbol{J}_a \ddot{\boldsymbol{q}} + \boldsymbol{J}_a \boldsymbol{M}^{-1} \boldsymbol{C} \dot{\boldsymbol{q}} = \underbrace{\boldsymbol{J}_a \boldsymbol{M}^{-1} \boldsymbol{J}_a^\top}_{\boldsymbol{M}_a^{-1}} \boldsymbol{f}_{FF}\,. \tag{6.10}$$

Using,

$$\dot{\boldsymbol{x}} = \boldsymbol{J}_a \dot{\boldsymbol{q}} \quad \xrightarrow{\frac{d}{dt}} \quad \ddot{\boldsymbol{x}} = \dot{\boldsymbol{J}}_a \dot{\boldsymbol{q}} + \boldsymbol{J}_a \ddot{\boldsymbol{q}}$$

the feed-forward controller $\boldsymbol{f}_{FF}$ can be derived based on

$$\ddot{\boldsymbol{x}} - \dot{\boldsymbol{J}}_a \dot{\boldsymbol{q}} + \boldsymbol{J}\boldsymbol{M}^{-1}\boldsymbol{C}\dot{\boldsymbol{q}} = \boldsymbol{M}_a^{-1} \boldsymbol{f}_{FF}. \tag{6.11}$$

For perfect tracking the condition $\ddot{\boldsymbol{x}} \stackrel{!}{=} \ddot{\boldsymbol{x}}_d$ must be fulfilled. Therefore,

$$\boldsymbol{f}_{FF} = \boldsymbol{M}_a \left[\ddot{\boldsymbol{x}}_d - \dot{\boldsymbol{J}}_a \boldsymbol{J}_a^{-1} \dot{\boldsymbol{x}}_d + \boldsymbol{J}_a \boldsymbol{M}^{-1} \boldsymbol{C} \boldsymbol{J}_a^{-1} \dot{\boldsymbol{x}}_d\right]\,. \tag{6.12}$$

Combining the feedback and the feed-forward controller yields the overall controller

$$\boldsymbol{\tau} = \boldsymbol{J}_a^\top \left(\underbrace{\boldsymbol{M}_a \left[\ddot{\boldsymbol{x}}_d - \dot{\boldsymbol{J}}_a \boldsymbol{J}_a^{-1} \dot{\boldsymbol{x}}_d + \boldsymbol{J}_a \boldsymbol{M}^{-1} \boldsymbol{C} \boldsymbol{J}_a^{-1} \dot{\boldsymbol{x}}_d\right]}_{\text{feed-forward}} + \underbrace{\boldsymbol{K}_a \left[\boldsymbol{x}_d - f_a(\boldsymbol{q})\right] + \boldsymbol{D}_a(\boldsymbol{q}) \left[\dot{\boldsymbol{x}}_d - \boldsymbol{J}_a \dot{\boldsymbol{q}}\right]}_{\text{feedback}} \right) + g(\boldsymbol{q})\,.$$

For the application of the feed-forward control term, a desired trajectory in terms of position, velocity and acceleration is required. In order to generate a desired trajectory out of the discontinuous jumps, a second order prefilter is implemented. This prefilter is based on the transfer function

$$G(s) = \frac{X(s)}{\hat{X}(s)} = \frac{\omega_0^2}{s^2 + 2\zeta_p \omega_0 s + \omega_0^2}\,. \tag{6.13}$$

Using this transfer function, any desired value $\hat{\boldsymbol{x}}_d$ is transformed into a twice differentiable trajectory in terms of $\boldsymbol{x}_d$, $\dot{\boldsymbol{x}}_d$ and $\ddot{\boldsymbol{x}}_d$. Figure 6.14 shows the block diagram of the transfer function. In the integrator chain, the acceleration, velocity and position can be extracted. The

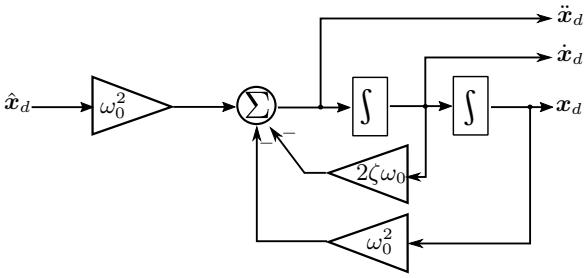


Figure 6.14.: Prefilter for the desired trajectory. This dynamics generates a desired trajectory based on a second order transfer function. Jumps on the input are transformed two a twice differentiable trajectory $\boldsymbol{x}_d(t)$.

prefilter acts component-wisely on the the input $\hat{\boldsymbol{x}}_d$. For the experiments the natural frequency is set to $\omega_0 = 2\pi$ and $\zeta_p = 0.7$.

Figure 6.15 shows a block diagram including the controllers and the prefilter. The input $\hat{\boldsymbol{x}}_d$ is transformed to a twice differentiable trajectory, which is fed to the controller blocks. Note, that the matrices $\boldsymbol{J}_a$, $\boldsymbol{C}$ and $\boldsymbol{M}$ used in the feed-forward controller depend on the joint configuration $\boldsymbol{q}$. This part of the signal flow is omitted in the block diagram.

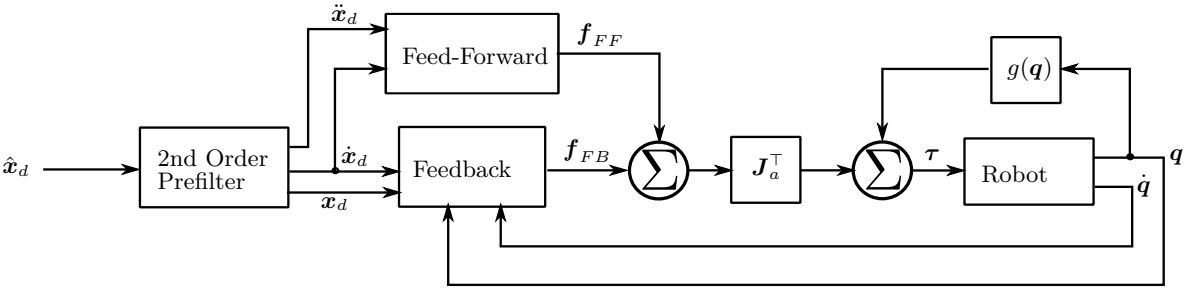
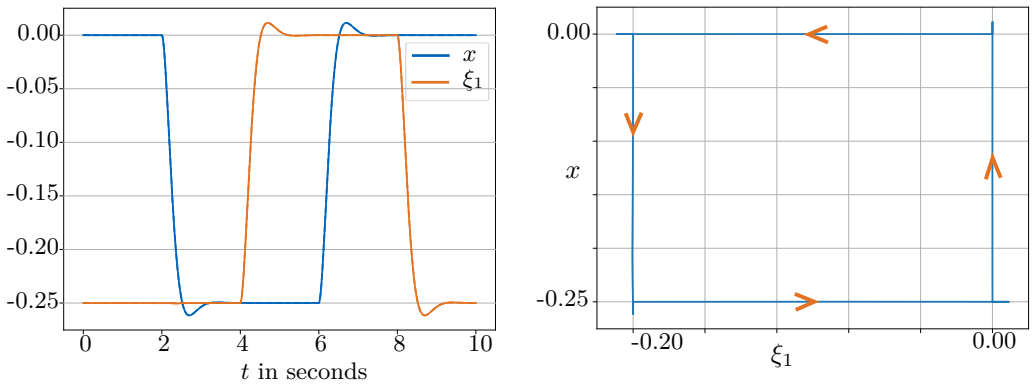


Figure 6.15.: Complete control structure consisting of feedback and feed-forward controller and the prefilter. The prefilter generates a desired trajectory out of an arbitrary input $\hat{\boldsymbol{x}}_d$. The forces generated by the controllers are transformed to joint torques via the transpose of the augmented Jacobian.

The experiment with the interleaved jumps in the x- and ξ_1-coordinate (Figure 6.10) is repeated using the new control scheme including the feed-forward controller. The desired value $\hat{\boldsymbol{x}}_d$ is set to the exact same values as in the previous experiment. Figure 6.16 shows the results of a dynamic simulation using the new control scheme. The feed-forward terms on the Coriolis and centrifugal forces as well as on the term related to the curvature of the self-motion manifolds, result in almost perfect decoupling between the coordinates. Also in the $x\xi_1$-plane (b) no disturbances are observable.

Finally, also the longer experiment schedule including all the available coordinates is repeated. Figure 6.17 shows the results and Figure A.7 shows the trajectory on the task space foliation. Also for the longer experiment, the coordinates show almost perfect decoupling. The major contribution to the disturbances without the feed-forward controller, is the acceleration corresponding to the self-motion manifold curvature. By predicting the curvature change related acceleration, the coordinates show decoupled behavior also at non-zero joint velocities.



(a) Coordinates (b) $x\xi_1$-plane

Figure 6.16.: Results of dynamic simulation of interleaved steps in the x- and ξ_1-coordinate. These results are obtained using the control scheme using the feed-forward and feedback controller. The coordinates show almost perfect decoupling.

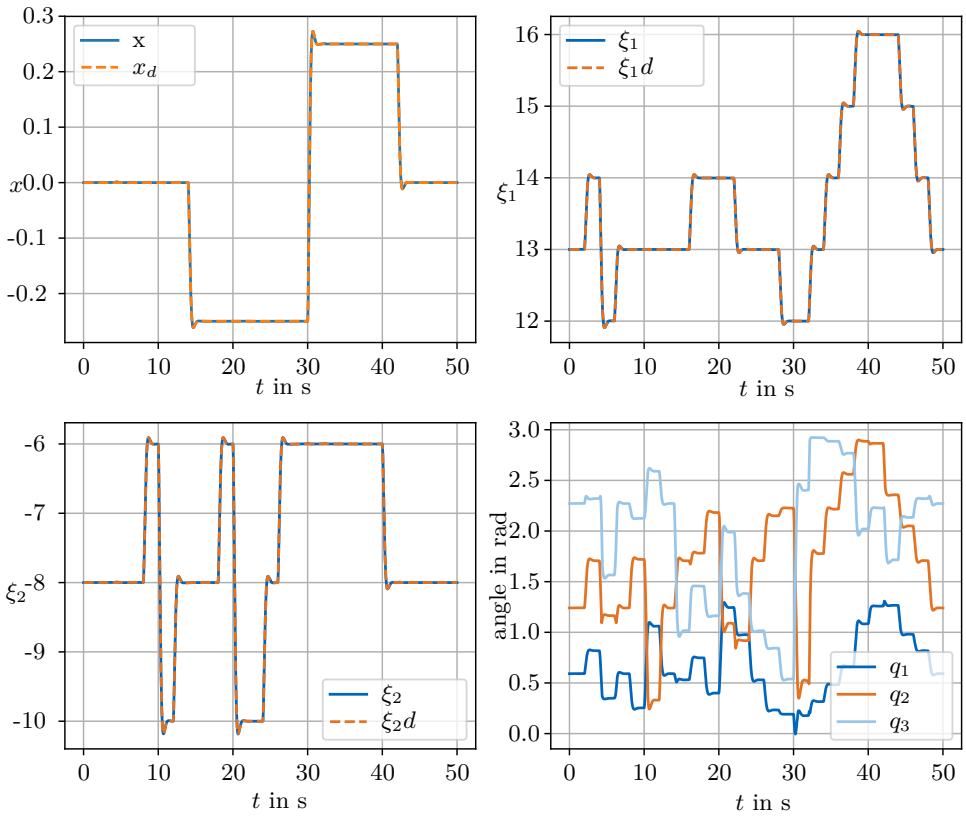


Figure 6.17.: Results for combined feedback and feed-forward control. The coordinates show almost perfect decoupling. Animated robot here: http://thesis.aaarne.de/anim-07-ff.gif

7 Discussion

In this chapter, the orthogonal foliation based control approach is compared to the gradient projection control approach of redundant manipulators. The approach requires optimization of a parametric function approximator in advance, while the state of the art approach requires no training at all. However, OSMCs also provide significant advantages. In contrast to the state of the art approach they provide a set of minimal coordinates. This allows to employ known techniques from task space manipulator control directly to self-motion control. Actually, the term self-motion control only partly describes the possibilities of such coordinates. Rather than only controlling the self-motion, it provides a set of decoupled coordinates that allow to superimpose controllers in both coordinates. This is captured by the term *orthogonal* in orthogonal self-motion coordinates (OSMCs).

The state of the art approach uses projections into the null space of the task space Jacobian

$$\boldsymbol{\tau} = \boldsymbol{J}^\top \boldsymbol{f} + \left[\boldsymbol{I} - \boldsymbol{J}^\top \boldsymbol{J}^{\#\top}\right] \boldsymbol{\tau}_0 \,. \tag{7.1}$$

This implicitly assumes a hierarchy of the control goals. A controller $\boldsymbol{\tau}_0 = \Phi(\boldsymbol{q}, \dot{\boldsymbol{q}})$ can not achieve arbitrary control goals, as parts of the torque are projected out. The new approach defines new coordinates $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$ that are minimal in a sense that they span each of the self-motion manifolds, but not more. A force $\boldsymbol{\eta}$ in those coordinates will not interfere with the task space controller. The decoupling works without a projector

$$\boldsymbol{\tau} = \boldsymbol{J}^\top \boldsymbol{f} + \boldsymbol{N}^\top \boldsymbol{\eta} \,. \tag{7.2}$$

In a sense, the method finds a new forward kinematics function $f_a(\boldsymbol{q})$, where the first m coordinates are the task space coordinates. This new forward kinematics function renders the redundant manipulator non-redundant under the new coordinates. Table 7.1 summarizes the arguments in the discussion, which are elaborated in the subsequent sections.

Table 7.1.: Comparison of state of the art and new approach

	Null Space Projectors (State of the art)	**Orthogonal Foliations** (New approach)
Decoupling based on	$\boldsymbol{\tau} = \boldsymbol{J}^\top \boldsymbol{f} + [\boldsymbol{I} - \boldsymbol{J}^\top \boldsymbol{J}^{\#\top}]\boldsymbol{\tau}_0$	$\boldsymbol{\tau} = \boldsymbol{J}^\top \boldsymbol{f} + \boldsymbol{N}^\top \boldsymbol{\eta}$
Provides coordinates	no	yes
Spring nature	projective	geodetic
Provides a potential	no	yes
Asymptotic Convergence Proof	complex	straightforward
Computational complexity	low	high
Requires Training/Optimization	no	yes

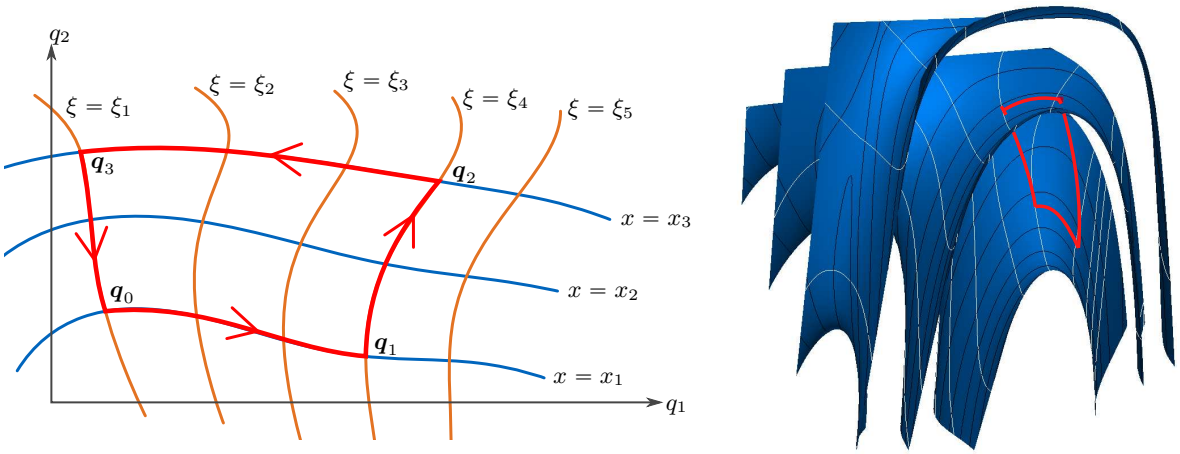
In the experiments with planar manipulators it was shown that the decoupling works equally well with both approaches. In closed-loop, the main contribution to disturbances were couplings of the joint velocity $\dot{\boldsymbol{q}}$ via the curvature of the self-motion manifolds and the Coriolis and centrifugal forces. Both approaches have this drawback. Using an additional feed-forward controller, the couplings almost entirely vanished.

The availability of coordinates provides some advantages compared to the state of the art approach. For instance, they can be used to perform linear motions in task space with additional reconfiguration of the robot arm using standard methods from instantaneous kinematics for trajectory generation. Also, for planning the existence of OSMCs promises advantages as planning algorithms can compute trajectories in task space and in an additional set of minimal coordinates. Usually, motion planning tasks are performed in joint space, where linear motions in task space are expressed by many via points. Given the augmented Jacobian, this can be computed without planners only using the Jacobian.

Moreover, the coordinates can be interpreted as potentials that allow to make statements on stability of the system. This is explained in section 7.1. Additionally, controllers in those coordinates can be interpreted as geodetic springs (section 7.2). It was concluded, that the coordinates, generally, can not be globally unique. Section 7.4 shows details on this and how to counteract the non-globality. Finally, section 7.4 provides a discussion on the interpretability of the coordinates.

7.1. Coordinates as Potential

The coordinates $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$ can also be interpreted as potential functions in joint space. Figure 7.1a shows this concept. The blue lines represent leaves of the task space foliation and the orange lines represent leaves of the OSMF. The red line shows a path in joint space, which would result from two interleaved jumps in the respective coordinates. After the transition $\boldsymbol{q}_0 \to \boldsymbol{q}_1 \to \boldsymbol{q}_2 \to \boldsymbol{q}_3 \to \boldsymbol{q}_0$ the start configuration is reached again. On the right (Figure 7.1b) this is shown for an actual simulation result corresponding to the experiment in Figure 6.16.



(a) 2D example

(b) Simulation result

Figure 7.1.: Coordinates can be interpreted as potential functions. On the left: schematic drawing. On the right: results from actual dynamic simulation.

Potentials allow to straightforwardly conclude statements on stability of the system. They enable the design of a straightforward Lyapunov function for the self-motion control part

and to apply Lyapunov-based stability theorems. For instance, in (6.3) it was trivial to show stability based on the quadratic form of a positive-definite matrix. The classical approach uses a pseudo-inverse and a projection in the controller, which makes conclusion of stability properties hard. For instance, [14] and [12] show analysis of stability using the classical approach. Using the new approach, the same techniques to show stability with respect to the task space dynamics can be applied to the new set of coordinates.

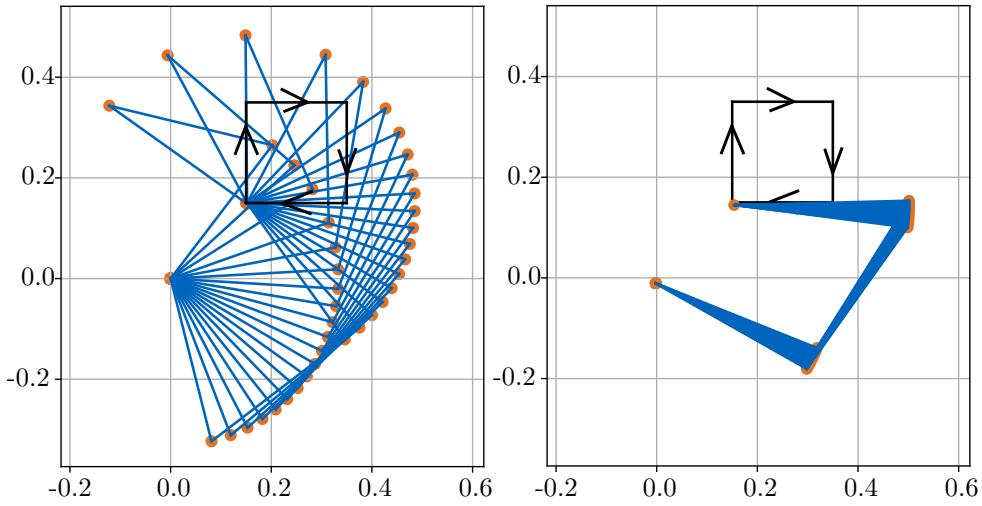
Additionally, as Klein and Huang pointed out, a controller based on the classical approach can drive the system to unpredictable configurations [39]. This is due to the non-integrability of the classical pseudo-inverse. The potential provided by the OSMC function $\varphi(\boldsymbol{q})$ does not have this issue. Figure 7.2 shows this experimentally. The idea for this experiment originates from [39]. A redundant manipulator traces out a square and the configurations are plotted after each full turn. On the left the Moore-Penrose pseudo-inverse is used

$$\Delta \boldsymbol{q} = \boldsymbol{J}^{+}(\boldsymbol{q}) \Delta \boldsymbol{x} \,. \tag{7.3}$$

The non-integrability leads to different configurations after each full turn. On the right, the inverse of the augmented Jacobian $\boldsymbol{J}_a^{-1}(\boldsymbol{q})$ is used and a fixed value is used for the $\xi = \xi_0 = \text{const}$ coordinate:

$$\Delta \boldsymbol{q} = \boldsymbol{J}_a^{-1} \begin{bmatrix} \Delta \boldsymbol{x} \\ 0 \end{bmatrix} . \tag{7.4}$$

After each full cycle the manipulator returns to the same configuration. In both cases a correctional velocity is added in order to keep the manipulator exactly on the square (cmp. (20) in [39]). Note that on the right open-loop control is applied to the ξ-coordinate. With an additional closed-loop correction on ξ no configuration change would be observable.

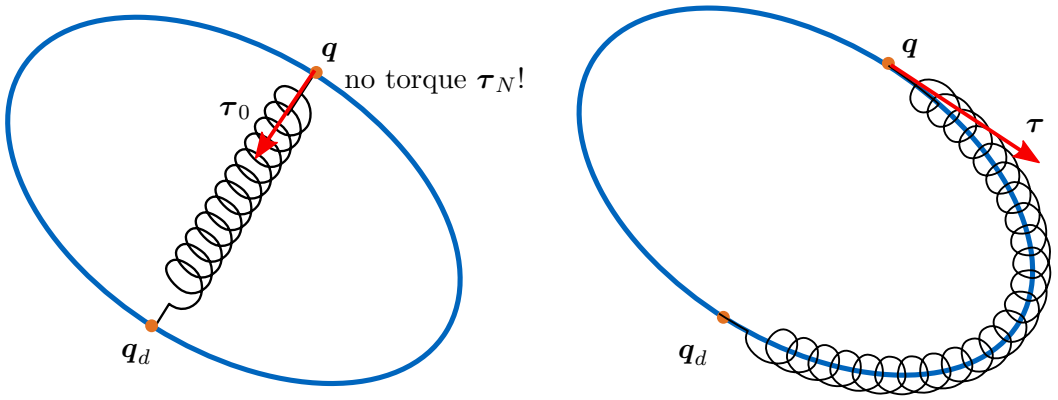


(a) Moore-Penrose pseudo-inverse (b) Full Jacobian inverse $\boldsymbol{J}_a^{-1}$

Figure 7.2.: Tracing out a square with a redundant manipulator. The robot configuration after each full cycle is shown in the diagrams for ten cycles. On the left: the non-integrability of the pseudo-inverse leads to different configurations. On the right: the robot stays at constant configurations after each cycle.

7.2. Geodetic Springs

The classical approach uses a controller in joint space $\boldsymbol{\tau}_0 = \Phi(\boldsymbol{q}, \dot{\boldsymbol{q}})$, which is projected into the null space of the task space Jacobian. Due to the projective nature of this mapping, it might result in a vanishing torque after the projection $\boldsymbol{\tau}_N = \left[\boldsymbol{I} - \boldsymbol{J}^\top \boldsymbol{J}^{\#\top}\right]\boldsymbol{\tau}_0$. This corresponds to a linear spring in joint space, which is projected into the tangent space of the self-motion manifolds. Dependent on the specific geometry of the task space foliation, this can lead to instable or stable equilibria. Especially for concave self-motion manifolds this may lead to strongly attractive local minima. Additionally, this may lead to strongly tensioned springs, which have no effect in one configuration, but that suddenly discharge upon small variations of the configuration. In contrast, a controller generating a generalized force $\boldsymbol{\eta}$, corresponds to a geodetic spring on the self-motion manifolds. Figure 7.3 shows this graphically.



(a) Classical approach

(b) Geodetic spring

Figure 7.3.: Joint space spring versus geodetic spring. On the left: the classical approach with the null space projection stays in a stable equilibrium and no joint torque $\boldsymbol{\tau}_N$ is generated. The spring remains stretched. On the right, the geodetic spring generates a joint torque and will relax.

7.3. Locality

Generally, it is not possible to find a minimal set of OSMCs that globally and uniquely define the configuration of the robotic manipulator. In particular, the spherical topology of typical self-motion manifolds makes it impossible to bijectively map Euclidean spaces to it. Additionally, even-dimensional spheres do not admit nowhere vanishing vector fields.

Non-redundant manipulators have as many task space coordinates as degrees of freedom in joint space. However, also for non-redundant manipulators the forward kinematics function in generally not unique. Typically, robot arms show different solutions like elbow-up and elbow-down configurations. Also the forward kinematics of a non-redundant manipulator do not provide a globally unique and bijective mapping.

Essentially, the augmented coordinate function $f_a(\boldsymbol{q})$ renders a redundant manipulator non-redundant. Consequently, the new coordinate function has the same limitations as well-known forward kinematic functions of non-redundant manipulators. In the results chapter, it could be observed that the validity regions are large. When staying in the valid region, the locality of the coordinate function is no drawback. However, when large motions are required another

approach needs to be used.

If globality is required, an *atlas* of coordinate functions can be used. An atlas is a collection of local coordinates that smoothly overlap at the boundaries. The entirety of maps of the atlas allows to reach every point on the manifold such that the local map is full rank. At the boundaries the coordinates can be switched from one set of coordinates to the next one. Figure 7.4 shows two coordinate functions φ_1 and φ_2 schematically. They provide a mapping from the manifold depicted in blue, to an Euclidean space. The coordinate functions are valid in the regions A and B, respectively. In the orange regions both coordinates are valid and they can be switched.

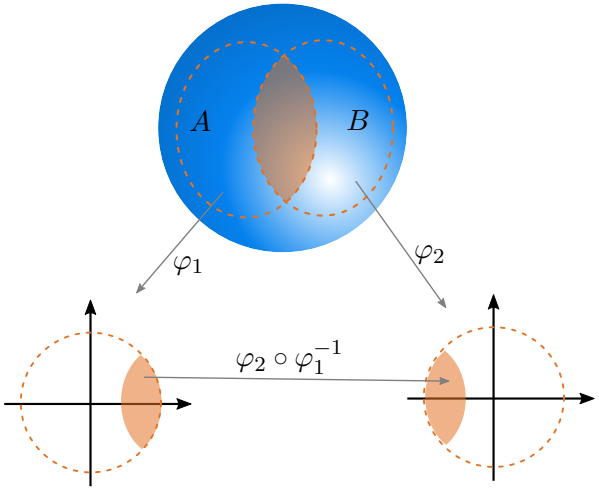


Figure 7.4.: Coordinate switching. When no global coordinates are possible, then coordinate switching can be applied. Coordinate functions are then valid only in regions on the manifold. At the intersection of valid regions, the coordinates can be switched. Figure adapted from [42, Fig. 1.3].

Then, this is no longer a global foliation of the manifold. When the coordinate functions are only valid in regions, this corresponds to a *lamination* [46] of the manifold. A set of foliations, which are valid in subsets of a manifold are called a lamination. It might or might no be possible to merge all the foliations of the subsets to a foliation of the entire manifold [18].

7.4. Interpretability

The neural network model provides coordinates, whose interpretation is not intuitively clear. The base problem is an integration problem, so a constant offset $\boldsymbol{\xi}_0$ can be added to the coordinates without altering the validity. Additionally, the underlying PDE is underdetermined and consequently the neural network has a lot of freedom to find a specific instantiation of the solution $\boldsymbol{\xi} = \varphi(\boldsymbol{q})$. The PDE neither provides information about the scaling of the coordinates, nor on the offset or the assignment. Effectively, the neural network simply finds any of the theoretically infinitely many solutions. The solution obtained strongly depends on the initial training parameters, which are sampled in the beginning of the training procedure. Enforcing additional boundary constraints on the solution leads to a contradiction with the initial training parameters and the network needs more time to train and is more likely to get stuck in local minima. Therefore, the best results are obtained when no boundary conditions at all are added to the training loss. In order to enforce boundary constraints, the initial parameters of the

network need to be consistent with the boundary conditions to avoid convergence to local minima.

For the experiments, the network was trained without any boundary conditions. After the training procedure the offset and the scaling of the coordinates were experimentally investigated by moving the robot around and checking the values of the coordinate function φ.

When specifying stiffnesses $\boldsymbol{K}_\xi$ it is not intuitively clear, how they feel on the physical robot as they do not provide any unit. Another trained model for the same manipulator may need another stiffness matrix in order to achieve the same dynamical properties.

Given a trained coordinate function φ, an arbitrary full-rank diagonal scaling matrix and an arbitrary offset $\boldsymbol{\xi}_0$ can be applied to the coordinate without changing the requirements on the orthogonality to the task space Jacobian. For example, consider the results for the model used in Figure 6.13. The two coordinates ξ_1 and ξ_2 show different offsets. In the experiments the scaling of the coordinates was always similar. The architecture of the neural network and the initialization scheme for the weights does not lead to a significant difference in the scaling of the coordinates. However, the offset was different for each trained model. Additionally, the model sometimes flipped the coordinates ξ_1 and ξ_2 in the experiments. The direction of the row vectors in the neural network Jacobian $\boldsymbol{N}(\boldsymbol{q})$ is basically fixed by the lines of principal curvature (Thm. 2). On the other hand, the order of the rows and the scaling of the vectors is free.

Other state of the art techniques for redundancy resolution use more interpretable representations. For example, the additional coordinate function for the self-motion coordinates may simply be a minimal subset of the joint angles. Consider a robot with seven degrees of freedom operating in three-dimensional task space. This requires six degrees of freedom, which leads to a degree of redundancy of one. Choosing, for instance, $\xi = \varphi(\boldsymbol{q}) = \boldsymbol{q}_1$ also locally leads to a coordinate function that can be used for redundancy resolution. This choice of coordinate function clearly is interpretable as joint angles are used directly. Another method [78] for a seven-dimensional uses the angle between the plane spanned by the elbow joint and the table plane as an additional coordinate. This type of coordinate is implemented on the light weight robot KUKA iiwa and described in [76]. Also this approach provides direct interpretability of the coordinate function. However, both examples do not provide dynamical decoupling of the task space dynamics. They require using the method based on the Jacobian null space projector. Therefore, these methods do not provide geodetic springs or potentials.

It can be concluded, that the approach chosen here currently does not lead to intuitively interpretable coordinates for the self-motion part of the dynamics. Improvement of the interpretability is a question for further research.

7.5. Neural Network Issues

The neural network approach provides only an approximative solution for the desired OSMCs. In the results it was shown that the training error is small compared to the disturbing effects resulting from the terms proportional to the velocity $\dot{\boldsymbol{q}}$. For larger robot models with many degrees of freedom the number of neurons required to accurately model the desired function might become intractable and larger training errors are to be expected. The classical approach does not need any trained model, so it has no problem with larger robot models.

For models with larger training errors, a combined approach is conceivable. The resulting torque from the controller in the new coordinates can be explicitly projected into the Jacobian

null space using

$$\boldsymbol{\tau} = \left[\boldsymbol{I} - \boldsymbol{J}^\top \boldsymbol{J}^{\#\top}\right] \boldsymbol{N}^\top \boldsymbol{\eta}\,. \tag{7.5}$$

This definitely results in a torque not interfering with the task space controller. Still, the neural network model would provide approximative geodetic springs, so local minima problems can be avoided. If the training error is still reasonably low, the model also provides an approximation of the potential.

From a computation perspective, the neural network also has the disadvantage of larger computational effort. As the neural network and its input-output Jacobian need to be evaluated in the control loop, it might limit the control frequency. However, the implementation using TensorFlow is fast. TensorFlow generates analytical expressions for the Jacobian, which are evaluated on the GPU in microseconds. Also the forward pass of the network is highly parallelizable and GPUs are efficient in computing it. Certainly, this leads to additional requirements on the control computer. Evaluation of the neural network and its Jacobian on a CPU is comparatively slow. Consequently, especially for models with a large amount of neurons, a GPU is required on the control computer.

When comparing the regression approach via the neural network to alternative approaches, it shows some advantages. An alternative would be a grid-based technique. Based on the conditions on the Jacobian the coordinate can be grown from a starting configuration on a grid in joint space. Then, interpolation techniques could be used in order to interpolate off-grid points. Clearly, the grid-based approach would highly suffer from the curse of dimensionality. The amount of grid points required per grid explodes with the number of joints due to the exponential growth. For function approximation using parametric functions the issue of the curse of dimensionality is less severe. Still, the amount of required training samples would probably increase exponentially, but the size of the final model can be limited by fixing the number of parameters of the model. In fact, the training results have shown that the number of neurons required does not grow exponentially with the number of joints.

8 Conclusion

8.1. Summary

Throughout the thesis, a concept for a special task space augmentation method for redundancy resolution was developed. The basic idea was to find a minimal function providing coordinates for the remaining degrees of freedom for redundant manipulators. In contrast to existing task space augmentation schemes, the coordinates are designed such that they are dynamically decoupled from the task space dynamics.

First, a foliation view on the classical task space forward kinematics was given. The entirety of self-motion manifolds can be regarded as a foliation of the joint space manifold. This provides a very geometrical understanding of the given problem. It follows that the rows of the task space Jacobian must be orthogonal to the rows of the Jacobian of the desired OSMCs. For dynamical consistency, the required metric for this orthogonality is the inverse mass matrix. Fundamental theorems from differential geometry show that the desired function can not be globally singularity-free and unique. However, the OSMC function exists locally. It was also shown how the OSMCs can be used for a simple impedance control scheme. Because of the dynamical decoupling, controllers in both coordinates can be additively superimposed without mutual disturbances.

The requirements on the Jacobian of the desired function can be interpreted as an underdetermined system of partial differential equations. In order to obtain a solution for this, a variational principle is applied. The target function is written as a two layer neural network and the network is optimized using gradient descent. The cost function is based on the derived requirements on the Jacobian.

Multiple trained models were evaluated kinematically and dynamically. Kinematic evaluation showed the decoupling of the coordinates on first-order dynamics. Then, for dynamic evaluation a manipulator is simulated in closed-loop with an impedance controller using the neural-network based coordinate function. On the actual multi-body dynamics, slight couplings occur. However, the major contribution to the couplings are the curvature of the self-motion manifold and Coriolis and centrifugal forces. An additional feed-forward controller leads to almost perfect decoupling of the coordinates. The method currently provides no straightforward interpretability of the self-motion coordinates. Additionally, the neural-network based method is approximative.

In contrast to simple Jacobian-based techniques and gradient projection methods, the approach developed in this thesis provides coordinates that can be interpreted as potential. Because no projection is required the springs corresponding to the controller can be considered geodetic springs instead of projected linear springs.

8.2. Future Work

The present thesis can be regarded as a proof of concept for the orthogonal foliation based self-motion coordinates. All results have only been shown on simulated planar manipulators with comparatively few degrees of freedom. The consequent next steps are training of models and showing experiments for non-planar robots and with more degrees of freedom. Additionally, the models have not been evaluated on an actual physical robotic system so far. In simulation everything is known, the dynamics model of an actual physical robot is almost certainly not exact. Furthermore, simulation can be arbitrarily slow and time can be stretched when more computation time is needed. On hardware, timing is strict and real-time applications are required. Therefore, the performance needs to be evaluated on actual hardware in the future.

The neural network is one approach to find an approximate solution to the underdetermined system of partial differential equations under constraints. Neural networks are a strong tool in the computer scientist's toolbox that allow to quickly try out things in an approximate way. However, other techniques to find a solution to the constrained and undertermined system of partial differential equations may exist. Further research on alternative solutions will be conducted in the future.

Currently, the network is trained using a least-squares cost function on the residual angles between the rows of the Jacobian. The network generally can not find a globally valid function. However, it is trying to do its best under the given cost function. Least squares leads to a non-sparse distribution of the residuals. The effect of a L2-norm in the loss function is known to lead to results that rather distribute the error throughout the space instead of having small regions with large errors. Therefore, another loss function that leads to sparser distributions of the error might result in a better overall performance. Also this needs further experiments and research. For the given least-squares cost function, the performance becomes better when no training samples are generated in the regions, where the network simply can not approximate the function. Else, the training would try to distribute the error out of those regions and also the performance elsewhere becomes worse. In consequence, a good region selection for the sampling of training samples is essential in order to achieve good performance. For low-dimensional problems the regions can be graphically estimated when the foliations are visualized. This is no longer possible for more than three dimensions. An algorithm to automatically find those regions is also future work. Also the total number of existing regions is not always clear. In order to transition from the valid regions of one model to the next, also the coordinate switching scheme shortly introduced in the discussion needs to be implemented.

The non-interpretability of the self-motion coordinates is currently a drawback of the proposed methods. Additionally, multiple training runs almost certainly lead to different coordinates. Further research is required in order to increase the interpretability of the coordinates and the repeatability of trainings.

In the present thesis, the only model evaluated was a traditional neural network based on the multilayer perceptron (MLP) scheme. The literature on machine learning techniques for learning of inverse dynamics and kinematics shows that different model choices perform better for those use cases. Especially locally weighted projection regression (LWPR) and Gaussian processes perform better than standard MLP models. Here, no inverse model is learned so the results can not directly be applied to the given problem. However, further investigation is required in order to find out if other models perform better for the given problem of finding dynamically decoupled self-motion coordinates.

Bibliography

[1] Alin Albu-Schäffer, Christian Ott, Udo Frese, and Gerd Hirzinger. "Cartesian impedance control of redundant robots: recent results with the DLR-light-weight-arms". In: *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*. Vol. 3. Sept. 2003, pp. 3704–3709.

[2] Alin Albu-Schäffer, Christian Ott, and Gerd Hirzinger. "A Unified Passivity-based Control Framework for Position, Torque and Impedance Control of Flexible Joint Robots". In: *The International Journal of Robotics Research* 26.1 (2007), pp. 23–39.

[3] Alexander. I. Bobenko, Daniel Matthes, and Yuri B. Suris. "Discrete and smooth orthogonal systems: C^∞-approximation". In: *International Mathematics Research Notices* 2003.45 (Jan. 2003), pp. 2415–2459.

[4] Christoph Borst, Thomas Wimbock, Florian Schmidt, Matthias Fuchs, Bernhard Brunner, Franziska Zacharias, Paolo Robuffo Giordano, Rainer Konietschke, Wolfgang Sepp, Stefan Fuchs, Christian Rink, Alin Albu-Schaffer, and Gerd Hirzinger. "Rollin' Justin - Mobile platform with variable base". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2009, pp. 1597–1598.

[5] Richard L. Burden and J. Douglas Faires. *Numerical Analysis*. Cengage Learning, 2010. ISBN: 9780538733519.

[6] Joel W. Burdick. "On the inverse kinematics of redundant manipulators: characterization of the self-motion manifolds". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE Computer Society, 1989, pp. 264–270.

[7] Sylvain Calinon, Florent Guenter, and Aude Billard. "On Learning, Representing, and Generalizing a Task in a Humanoid Robot". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37.2 (2007), pp. 286–298.

[8] John J. Craig. *Introduction to Robotics: Mechanics and Control*. 2nd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201095289.

[9] Aaron D'Souza, Sethu Vijayakumar, and Stefan Schaal. "Learning inverse kinematics". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 1. 2001, pp. 298–303.

[10] Percy Dahm and Frank Joublin. *Closed form solution for the inverse kinematics of a redundant robot arm*. Institut für Neuroinformatik, Ruhr-Universität. Bochum. Aug. 1997.

[11] Alexander Dietrich. *Whole-Body Impedance Control of Wheeled Humanoid Robots*. Vol. 116. Springer Tracts in Advanced Robotics. Springer International Publishing, 2016.

[12] Alexander Dietrich, Kristin Bussmann, Florian Petit, Paul Kotyczka, Christian Ott, Boris Lohmann, and Alin Albu-Schäffer. "Whole-body Impedance Control of Wheeled Mobile Manipulators: Stability Analysis and Experiments on the Humanoid Robot Rollin' Justin". In: *Autonomous Robots* (2015).

[13] Alexander Dietrich, Christian Ott, and Alin Albu-Schäffer. "An overview of null space projections for redundant, torque-controlled robots". In: *The International Journal of Robotics Research* 34.11 (2015), pp. 1385–1400.

[14] Alexander Dietrich, Christian Ott, and Alin Albu-Schäffer. "Multi-Objective Compliance Control of Redundant Manipulators: Hierarchy, Control, and Stability". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 3043–3050.

[15] Alexander Dietrich, Thomas Wimbock, Alin Albu-Schaffer, and Gerd Hirzinger. "Integration of Reactive, Torque-Based Self-Collision Avoidance Into a Task Hierarchy". In: *IEEE Transactions on Robotics* 28.6 (Dec. 2012), pp. 1278–1293.

[16] Keith L. Doty, Claudio Melchiorri, and Claudio Bonivento. "A Theory of Generalized Inverses Applied to Robotics". In: *The International Journal of Robotics Research* 12.1 (1993), pp. 1–19.

[17] Olav Egeland. "Task-space tracking with redundant manipulators". In: *IEEE Journal on Robotics and Automation* 3.5 (Oct. 1987), pp. 471–475.

[18] Encyclopedia of Mathematics. *Lamination*. 2011. URL: `http://www.encyclopediaofmath.org/index.php?title=Lamination&oldid=24489`.

[19] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN: 0387743146.

[20] Fanny Ficuciello, Luigi Villani, and Bruno Siciliano. "Variable Impedance Control of Redundant Manipulators for Intuitive Human–Robot Physical Interaction". In: *IEEE Transactions on Robotics* 31.4 (Aug. 2015), pp. 850–863.

[21] Theodore Frankel. *The Geometry of Physics: An Introduction*. 3rd ed. Cambridge University Press, 2011. ISBN: 0521539277.

[22] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN: 0262337371.

[24] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 2nd. USA: Addison-Wesley Longman Publishing Co., Inc., 1994. ISBN: 0201558025.

[25] Ulrich Hagn et al. "DLR MiroSurge - A Versatile System for Research in Endoscopic Telesurgery". In: *International Journal of Computer Assisted Radiology and Surgery* Volume (2009), pp. 183–193.

[26] Ernst Hairer, Syvert P. Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2008. ISBN: 9783540566700.

[27] Gerd Hirzinger, Alin Albu-Schäffer, Matthias Hähnle, I. Schaefer, and Norbert Sporer. "On a new generation of torque controlled light-weight robots". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Vol. 4. 2001, pp. 3356–3363.

[28] Neville Hogan. "Impedance control: An approach to manipulation: Part I-Theory; Part II-Implementation; Part III-Applications". In: *Journal of Dynamic Systems, Measurement and Control* 107.1 (1985), pp. 1–24.

[29] John Hollerbach and Ki C. Suh. "Redundancy resolution of manipulators through torque optimization". In: *IEEE Journal on Robotics and Automation* 3.4 (Aug. 1987), pp. 308–316.

[30] Steven Hurder. “Lectures on Foliation Dynamics”. In: *Foliations: Dynamics, Geometry and Topology*. Ed. by Jesús Álvarez López and Marcel Nicolau. Basel: Springer Basel, 2014, pp. 87–149. ISBN: 978-3-0348-0871-2.

[31] Jessica Hutzl, David Oertel, and Heinz Wörn. “Knowledge-based direction prediction to optimize the null-space parameter of a redundant robot in a telemanipulation scenario”. In: *IEEE International Symposium on Robotic and Sensors Environments (ROSE) Proceedings*. Oct. 2014, pp. 25–30.

[32] Jürgen Jost. *Riemannian Geometry and Geometric Analysis*. Universitext. Springer International Publishing, 2017. ISBN: 9783319618609.

[33] Oussama Khatib. “A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation”. In: *IEEE Journal on Robotics and Automation* 3.1 (1987), pp. 43–53.

[34] Oussama Khatib. “Inertial Properties in Robotic Manipulation: An Object-Level Framework”. In: *The International Journal of Robotics Research* 14.1 (1995), pp. 19–36.

[35] Oussama Khatib. *Lecture Notes on Advanced Robotic Manipulation*. Stanford University. 2005.

[36] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Vol. 2. Mar. 1985, pp. 500–505.

[37] Oussama Khatib. “Reduced Effective Inertia in Macro-/Mini-Manipulator Systems”. In: *The Fifth International Symposium on Robotics Research*. Cambridge, MA, USA: MIT Press, 1990, pp. 279–284.

[38] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *Proceedings of the International Conference for Learning Representations (ICLR)*. 2015.

[39] Charles A. Klein and Ching-Hsiang Huang. “Review of pseudoinverse control for use with kinematically redundant manipulators”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.2 (Mar. 1983), pp. 245–250.

[40] Paul Kotyczka. *Nonlinear Control*. Lecture Notes, Technical University of Munich, Department of Mechanical Engineering, Chair of Automatic Control. 2019.

[41] Steven M. LaValle. *Planning Algorithms*. USA: Cambridge University Press, 2006. ISBN: 0521862051.

[42] John M. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer, 2003. ISBN: 9780387954486.

[43] Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. “Efficient implementation of marching cubes cases with topological guarantees”. In: *Journal of Graphics Tools* 8.2 (Dec. 2003), pp. 1–15.

[44] Lennart Ljung. *System Identification: Theory for the User*. Pearson Education, 1998. ISBN: 9780132440530.

[45] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. 1st. New York, NY, USA: Cambridge University Press, 2017. ISBN: 1107156300.

[46] Mikhail Lyubich and John Willard Milnor. *Laminations and Foliations in Dynamics, Geometry and Topology: Proceedings of the Conference on Laminations and Foliations in Dynamics, Geometry and Topology*. Contemporary mathematics - American Mathematical Society. American Mathematical Society, 2001.

[47] Anthony A. Maciejewski and Charles A. Klein. “Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments”. In: *The International Journal of Robotics Research* 4.3 (1985), pp. 109–117.

[48] Nico Mansfeld, Fabian Beck, Alexander Dietrich, and Sami Haddadin. “Interactive Null Space Control for Intuitively Interpretable Reconfiguration of Redundant Manipulators”. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*. 2017.

[49] Nico Mansfeld, Badis Djellab, Jaime Raldua Veuthey, Fabian Beck, and Sami Haddadin. “Improving the Performance of Biomechanically Safe Velocity Control for Redundant Robots through Reflected Mass Minimization”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2017.

[50] Nico Mansfeld, Youssef Michel, Tobias Bruckmann, and Sami Haddadin. “Improving the Performance of Auxiliary Null Space Tasks via Time Scaling-Based Relaxation of the Primary Task”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. May 2019, pp. 9342–9348.

[51] Jerrold E. Marsden and Anthony Tromba. *Vector Calculus*. Bd. 1. W.H. Freeman, 1996. ISBN: 9780716724322.

[52] Martín Abadi et al. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems”. In: *Proceedings of the USENIX Symposium on Operating Systems Desgin and Implementation (OSDI)*. Vol. 12. Savannah, GA, USA, Nov. 2016, pp. 265–283.

[53] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1994. ISBN: 0849379814.

[54] Ferdinando A. Mussa-Ivaldi and Neville Hogan. “Integrable Solutions of Kinematic Redundancy via Impedance Control”. In: *The International Journal of Robotics Research* 10.5 (1991), pp. 481–491.

[55] Duy Nguyen-Tuong, Jan Peters, Matthias Seeger, and Bernhard Schölkopf. “Learning Inverse Dynamics: A Comparison”. In: *Advances in Computational Intelligence and Learning: Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*. Jan. 2008, pp. 13–18.

[56] Duy Nguyen-Tuong, Matthias Seeger, and Jan Peters. “Local Gaussian Process Regression for Real Time Online Model Learning and Control”. In: *Advances in neural information processing systems: Proceedings of the Annual Conference on Neural Information Processing Systems*. Vol. 22. Max-Planck-Gesellschaft. Red Hook, NY, USA: Curran, June 2009, pp. 1193–1200.

[57] Luong An Nguyen, Rajnikant V. Patel, and Kash Khorasani. “Neural network architectures for the forward kinematics problem in robotics”. In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. Vol. 3. 1990, pp. 393–399.

[58] Henk Nijmeijer and Arjan van der Schaft. *Nonlinear Dynamical Control Systems*. Berlin, Heidelberg: Springer, 1990. ISBN: 0-387-97234-X.

[59] Nicolai Ommer, Alexander Stumpf, and Oskar von Stryk. “Real-Time Online Adaptive Feedforward Velocity Control for Unmanned Ground Vehicles”. In: *Proceedings of the RoboCup 2017 Conference: Robot World Cup XXI*. Ed. by Hidehisa Akiyama, Oliver Obst, Claude Sammut, and Flavio Tonidandel. Cham: Springer International Publishing, 2018, pp. 3–16.

[60] Romeo Ortega and Eloísa García-Canseco. “Interconnection and Damping Assignment Passivity-Based Control: A Survey”. In: *European Journal of Control* 10.5 (2004), pp. 432–450.

[61] Romeo Ortega, Arjan van der Schaft, Bernhard Maschke, and Gerardo Escobar. “Interconnection and damping assignment passivity-based control of port-controlled Hamiltonian systems”. In: *Automatica* 38.4 (2002), pp. 585–596.

[62] Samuel Otten. *An Introduction to Distributions and Foliations*. Michigan State University. 2008.

[63] Roger Penrose. “A generalized inverse for matrices”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 51.3 (1955), pp. 406–413.

[64] Jan Peters and Stefan Schaal. “Learning to Control in Operational Space”. In: *The International Journal of Robotics Research* 27.2 (2008), pp. 197–212.

[65] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN: 026218253X.

[66] Paul Renteln. *Manifolds, Tensors and Forms: An Introduction for Mathematicians and Physicists*. Cambridge University Press, 2014. ISBN: 9781107042193.

[67] David S. Richeson. *Euler’s Gem: The Polyhedron Formula and the Birth of Topology*. Princeton Science Library. Princeton University Press, 2019. ISBN: 9780691191997.

[68] Markus Rickert. *Sampling-Based Motion Planning*. Lecture in Robot Motion Planning at Techincal University of Munich, Department of Informatics, Chair of Robotics, Artificial Intelligence an Real-Time Systems. May 2019.

[69] Roberto Rossi, Matteo Parigi Polverini, Andrea Maria Zanchettin, and Paolo Rocco. “A pre-collision control strategy for human-robot interaction based on dissipated energy in potential inelastic impacts”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2015, pp. 26–31.

[70] Joseph J. Rotman. *An Introduction to Algebraic Topology*. 0072-5285. Springer, New York, NY, 1988. ISBN: 978-1-4612-8930-2.

[71] Camille Salaün, Vincent Padois, and Olivier Sigaud. “Learning Forward Models for the Operational Space Control of Redundant Robots”. In: *From Motor Learning to Interaction Learning in Robots*. Ed. by Olivier Sigaud and Jan Peters. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 169–192.

[72] Stefan Schaal, Christopher Atkeson, and Sethu Vijayakumar. “Scalable Techniques from Nonparametric Statistics for Real Time Robot Learning”. In: *Applied Intelligence* 17 (July 2002), pp. 49–60.

[73] Lorenzo Sciavicco and Bruno Siciliano. “Solving the Inverse Kinematic Problem for Robotic Manipulators”. In: *Proceedings of the Sixth CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*. Ed. by A. Morecki, G. Bianchi, and K. Kędzior. Boston, MA: Springer US, 1987, pp. 107–114.

[74] Lorenzo Sciavicco, Bruno Siciliano, and Pasquale Chiacchio. “On the Use of Redundancy in Robot Kinematic Control”. In: *Proceedings of the American Control Conference*. June 1988, pp. 1370–1375.

[75] Jonathan M. Selig. *Geometric Fundamentals of Robotics*. Monographs in Computer Science. Springer, 1996. ISBN: 978-1-4757-2486-8.

[76] Shashank Sharma and Christian Scheurer. "Generalized Unified Closed Form Inverse Kinematics for Mobile Manipulators With Reusable Redundancy Parameters". In: *Proceedings of the ASME 2017 Mechanisms and Robotics Conference*. Vol. Volume 5B: 41st Mechanisms and Robotics Conference. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Aug. 2017.

[77] Masayuki Shimizu. "Singularity Avoidance with Preservation of Manipulation Performance in Impedance Control for Human-Robot Collaboration". In: *Proceedings of the 58th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*. Sept. 2019, pp. 1604–1610.

[78] Masayuki Shimizu, Hiromu Kakuya, Woo-Keun Yoon, Kosei Kitagaki, and Kazuhiro Kosuge. "Analytical Inverse Kinematic Computation for 7-DOF Redundant Manipulators With Joint Limits and Its Application to Redundancy Resolution". In: *IEEE Transactions on Robotics* 24.5 (Oct. 2008), pp. 1131–1142.

[79] Aaron P. Shon, Keith Grochow, and Rajesh P. N. Rao. "Robotic imitation from human motion capture using Gaussian processes". In: *Proceedings of the 5th IEEE-RAS International Conference on Humanoid Robots*. 2005, pp. 129–134.

[80] Bruno Siciliano. "Kinematic Control of Redundant Robot Manipulators: A Tutorial". In: *Journal of Intelligent and Robotic Systems* 3 (Sept. 1990), pp. 201–212.

[81] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. 1st. Springer Publishing Company, Incorporated, London, 2009. ISBN: 1846286417.

[82] Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice Hall, 1991. ISBN: 9780130408907.

[83] Jorge Sotomayor and Ronaldo Garcia. "Lines of Curvature on Surfaces, Historical Comments and Recent Developments". In: *The São Paulo Journal of Mathematical Sciences* 2.1 (June 2008), pp. 99–139.

[84] Freek Stulp and Olivier Sigaud. "Many regression algorithms, one unified model – A review". In: *Neural Networks* 69 (2015), pp. 60–79.

[85] Hang Su, Shuai Li, Jagadesh Manivannan, Luca Bascetta, Giancarlo Ferrigno, and Elena De Momi. "Manipulability Optimization Control of a Serial Redundant Robot for Robot-assisted Minimally Invasive Surgery". In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. May 2019, pp. 1323–1328.

[86] Ganghua Sun and Brian Scassellati. "Reaching through learned forward model". In: *Proceedings of the 4th IEEE-RAS International Conference on Humanoid Robots*. Vol. 1. 2004, pp. 93–112.

[87] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1997. ISBN: 9780898719574.

[88] Sethu Vijayakumar and Stefan Schaal. "Locally Weighted Projection Regression: Incremental Real Time Learning in High Dimensional Space". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 1079–1086.

[89] Ian D. Walker. "The use of kinematic redundancy in reducing impact and contact effects in manipulation". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Vol. 1. May 1990, pp. 434–439.

[90] Charles W. Wampler. “The Inverse Function Approach to Kinematic Control of Redundant Manipulators”. In: *Proceedings of the American Control Conference*. June 1988, pp. 1364–1369.

[91] Daniel E. Whitney. “Resolved Motion Rate Control of Manipulators and Human Prostheses”. In: *IEEE Transactions on Man-Machine Systems* 10.2 (June 1969), pp. 47–53.

[92] Tsuneo Yoshikawa. “Manipulability of Robotic Mechanisms”. In: *The International Journal of Robotics Research* 4.2 (1985), pp. 3–9.

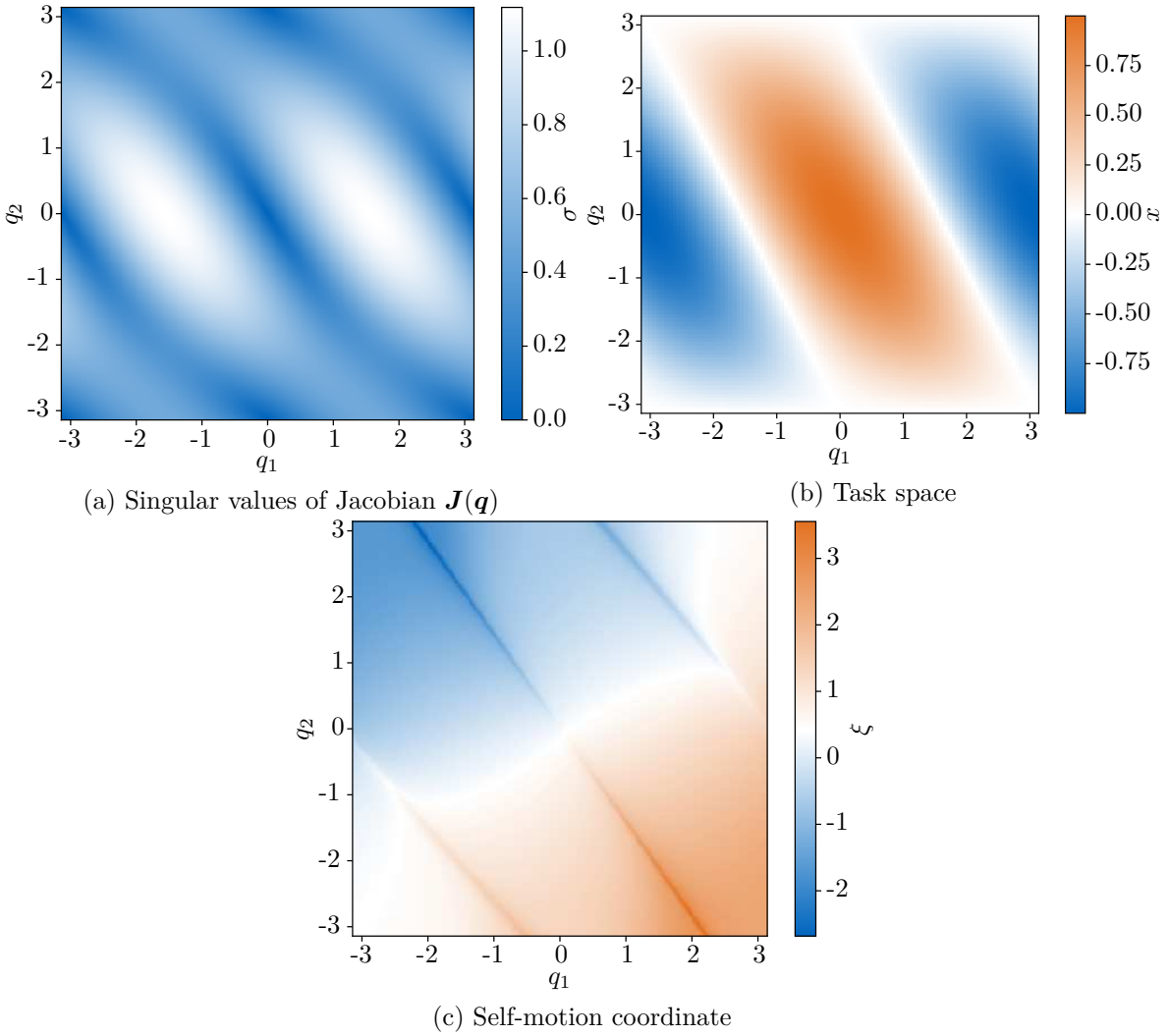
List of Figures

List of Acronyms

API	Application Programming Interface
CPU	Central Processing Unit
DLR	German Aerospace Center (Ger. Deutsches Zentrum für Luft- und Raumfahrt e.V.)
DoF	Degree of Freedom
GMM	Gaussian Mixture Model
GP	Gaussian Process
GPU	Graphical Processing Unit
IDA-PBC	Interconnection and Damping Assignment Passivity-Based Control
LBR	Light Weight Robot (Ger. Leichtbauroboter)
LWPR	Locally Weighted Projection Regression
MLP	Multilayer Perceptron
OSMC	Orthogonal Self-motion Coordinate
OSMF	Orthogonal Self-motion Foliation
PDE	Partial Differential Equation
RAM	Random Access Memory
SVD	Singular Value Decomposition

A

Enlarged and Additional Figures



(a) Singular values of Jacobian $\boldsymbol{J}(\boldsymbol{q})$

(b) Task space

(c) Self-motion coordinate

Figure A.1.: Additional figures for section 6.1.1.

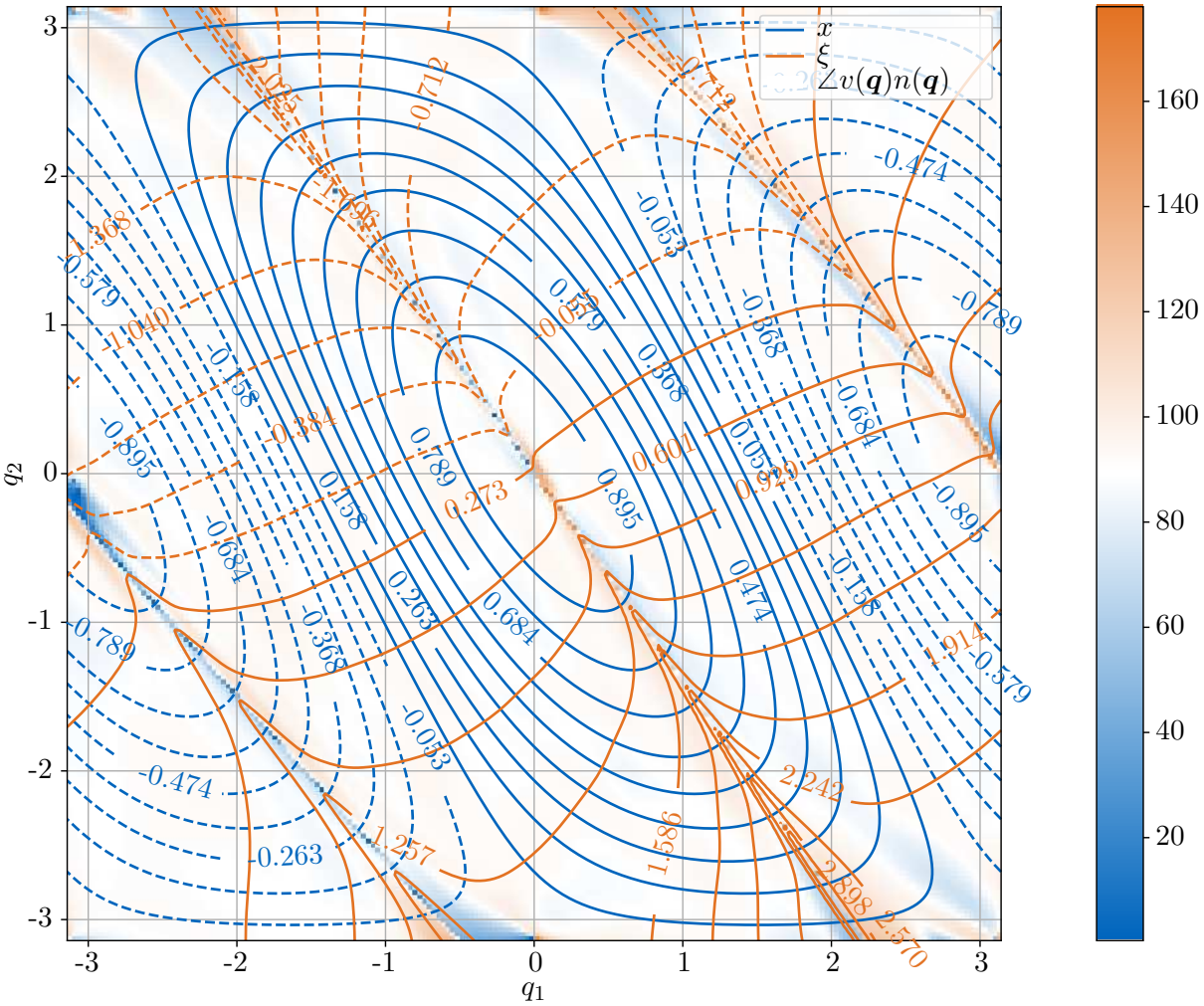
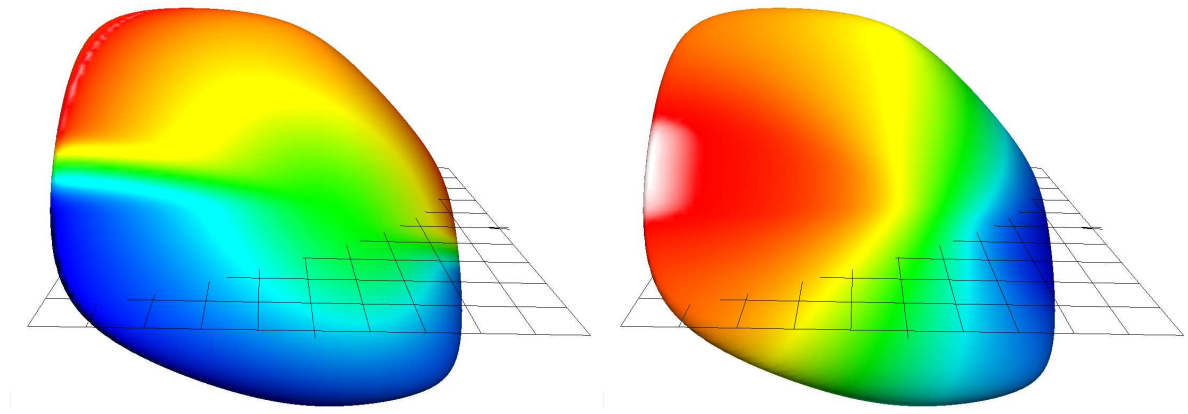


Figure A.2.: Enlarged version of Figure 6.4 with additional labels on the isolines.



(a) ξ_1 (b) ξ_2

Figure A.3.: Color coded values of the self-motion coordinates on one self-motion manifold. Green values denote values close to zero. More negative values are indicated by bluer colors, black being most negative. More positive values are encoded in red, white being the most positive value.

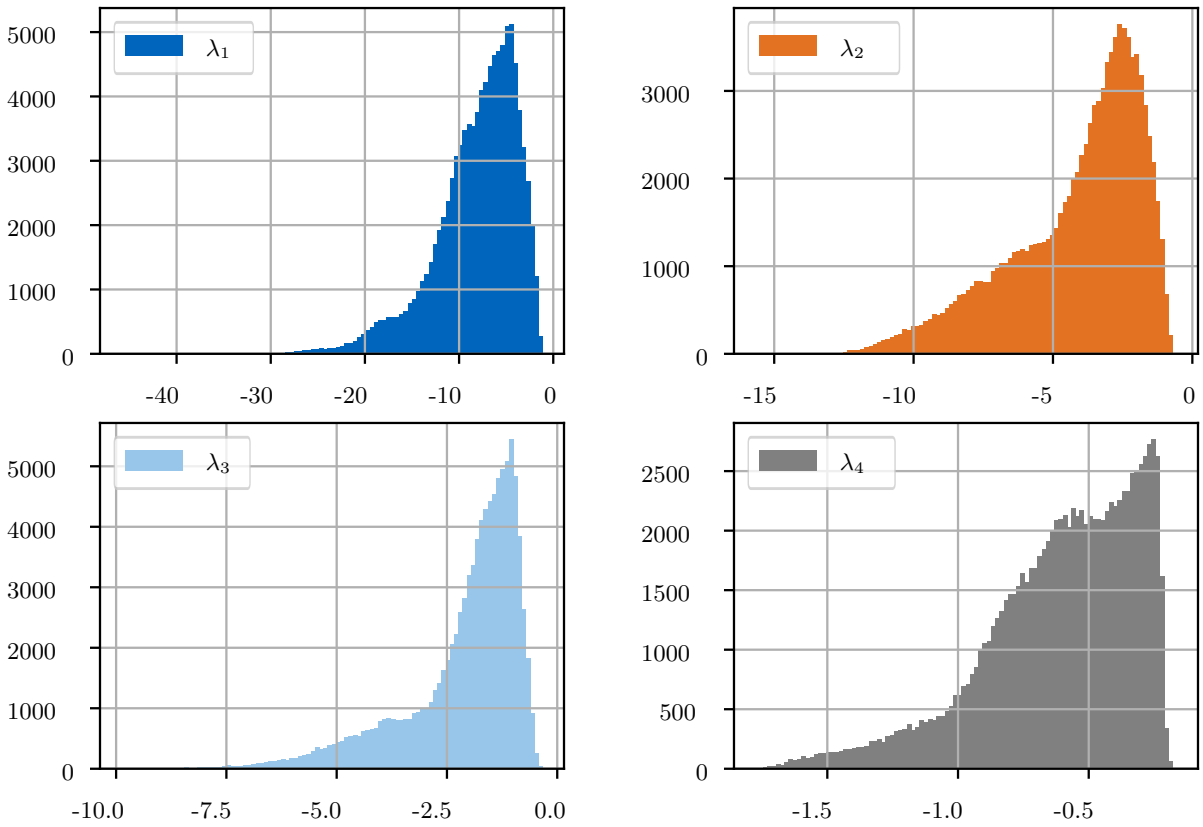


Figure A.4.: Histogram of the eigenvalues of the linearized system matrix for transposed Jacobian inverse kinematics.

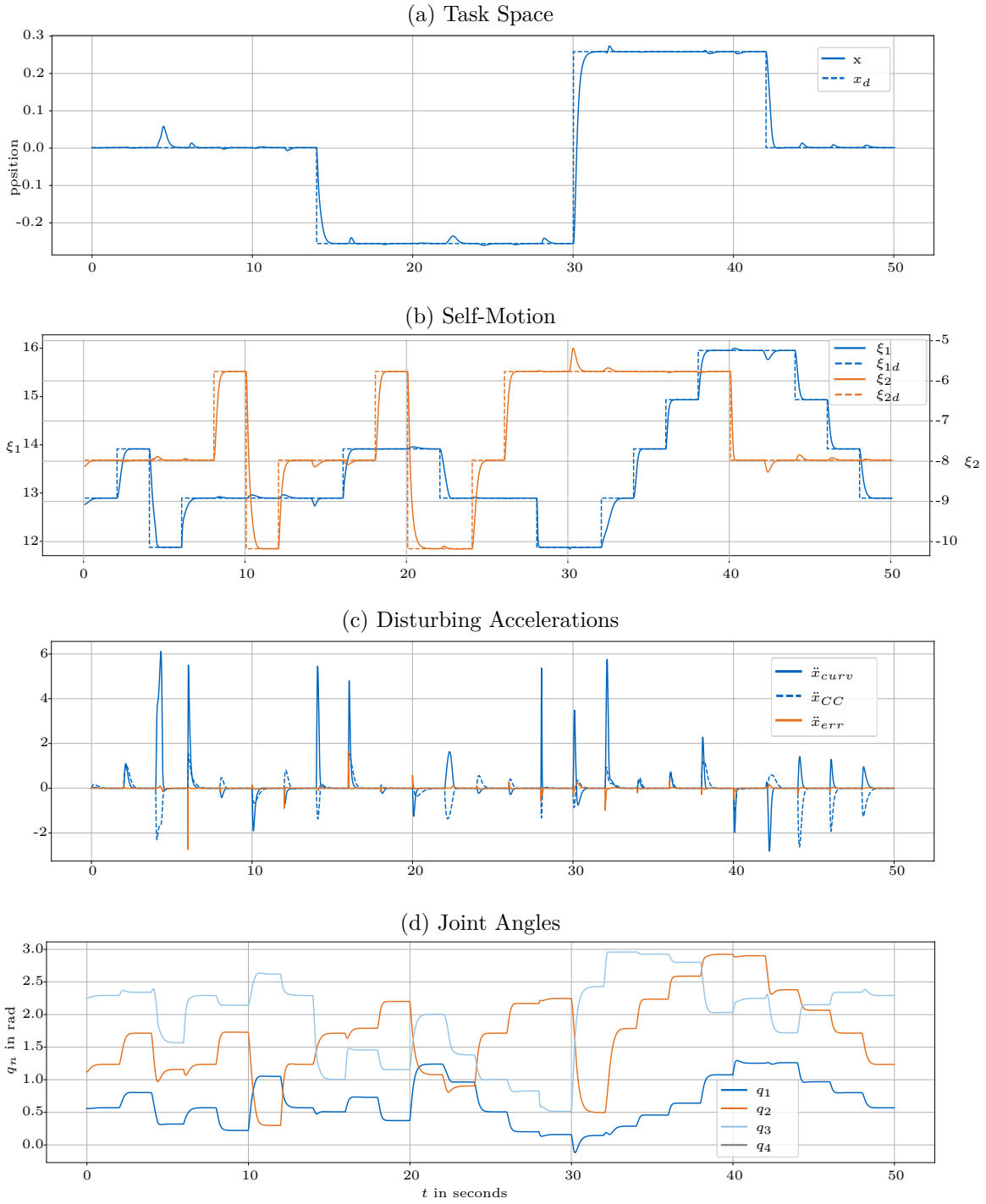


Figure A.5.: Simulation results for closed-loop behavior of manipulator and the augmented control scheme using neural-network based coordinates. The results are for a damping ratio of $\zeta = 1.0$. Animated robot here: http://thesis.aaarne.de/anim-10.gif

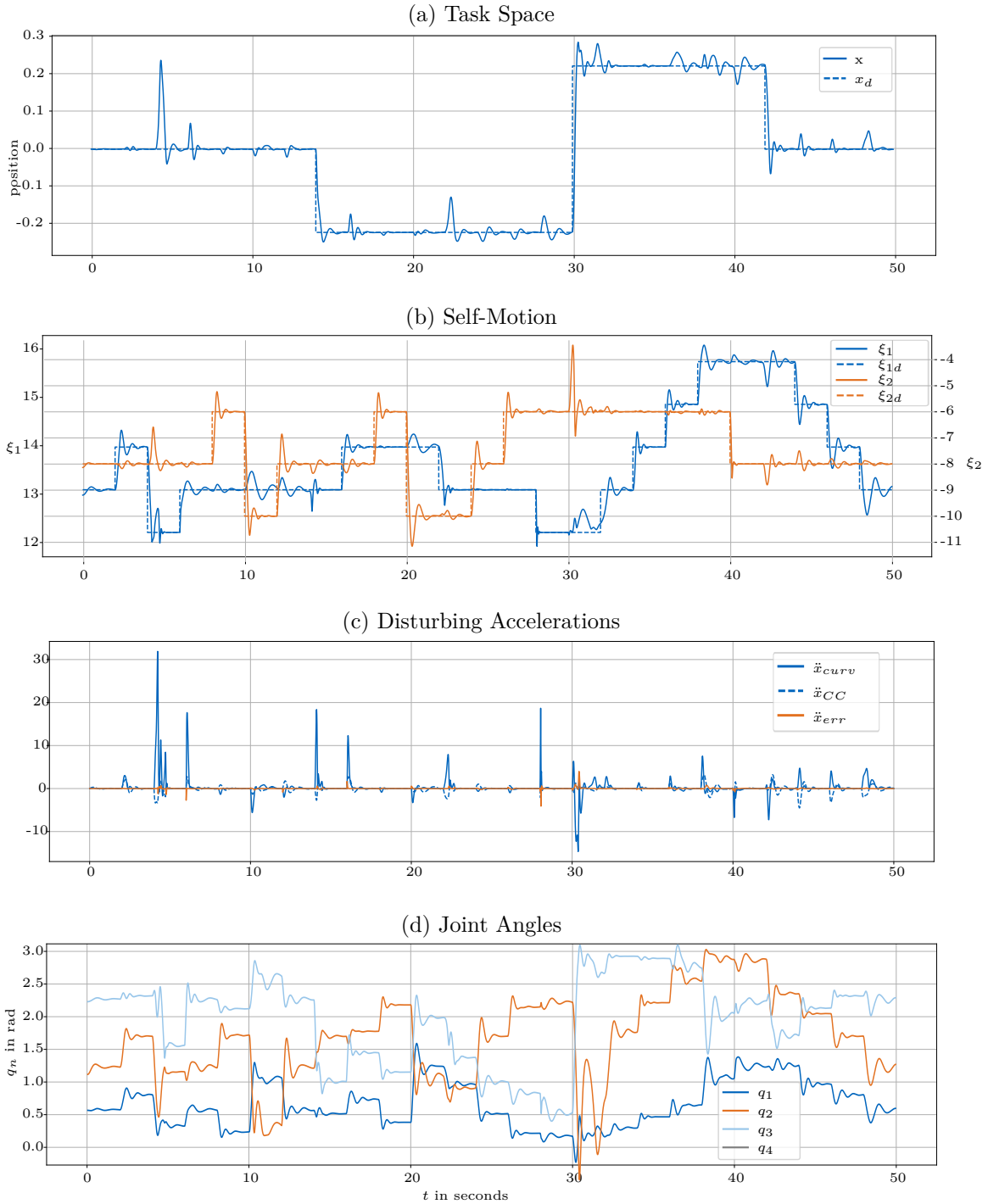


Figure A.6.: Simulation results for closed-loop behavior of manipulator and the augmented control scheme using neural-network based coordinates. The results are for a damping ratio of $\zeta = 0.3$. Animated robot here: http://thesis.aaarne.de/anim-03.gif

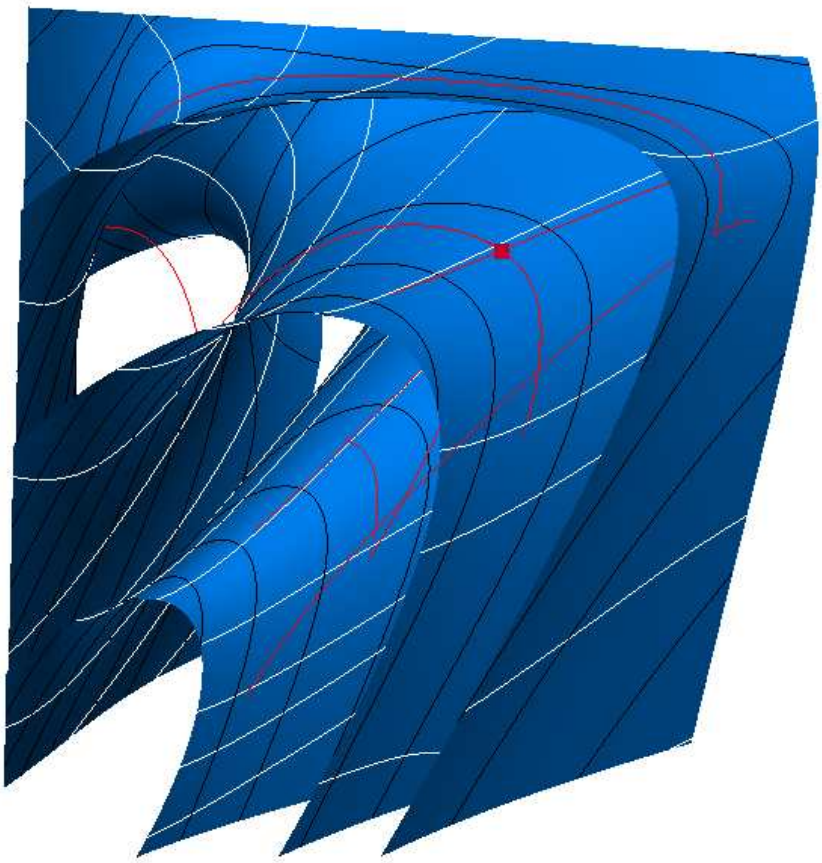


Figure A.7.: Visualization of the joint trajectory on three leaves of the task space foliation.

B

Planar Cartesian Forward Kinematics

This chapter briefly describes how the forward kinematics function and the Jacobian of the planar robots is computed. The links of the robot are rigid bodies. Therefore, turning the angles of the manipulator corresponds to a rigid body transformation. Consider the drawing in Figure B.1. Each of the links i has a frame $\{F_i\}$ attached at the end. The base frame $\{F_0\}$ is fixed. In homogeneous coordinates, the transformation from frame $\{F_{i-1}\}$ to $\{F_i\}$ is given by the transformation matrix

$$\boldsymbol{T}_{i-1,i} = \begin{bmatrix} \cos q_i & -\sin q_i & l_i \cos q_i \\ \sin q_i & \cos q_i & l_i \sin q_i \\ 0 & 0 & 1 \end{bmatrix} . \tag{B.1}$$

Then, the final transformation between $\{F_0\}$ and the final links $\{F_n\}$ can be computing by chaining the transformation matrices

$$\boldsymbol{T}_{0,n} = \prod_{i=1}^{n} \boldsymbol{T}_{i-1,i} . \tag{B.2}$$

Given the overall transformation matrix $\boldsymbol{T}(\boldsymbol{q}) = \boldsymbol{T}_{0,n}$ the forward kinematics function is computed using

$$\boldsymbol{x} = f(\boldsymbol{q}) = \begin{bmatrix} x \\ y \\ \varphi \end{bmatrix} = \begin{bmatrix} T_{1,3} \\ T_{2,3} \\ \operatorname{atan2}(T_{2,1}, T_{1,1}) \end{bmatrix} . \tag{B.3}$$

An analytic expression for the Jacobian $\boldsymbol{J}(\boldsymbol{q}) = \left[\frac{\partial f_i}{\partial q_i}\right]$ is obtained by symbolically differentiating the elements of $f(\boldsymbol{q})$. Listing B.1 shows the implementation of the forward kinematics and the symbolic differentiation in Python. This code was used to generate the expressions for the Jacobian for the training and evaluation procedures within the thesis.

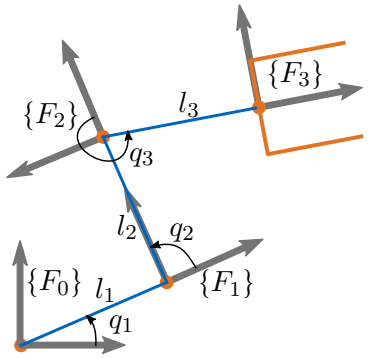


Figure B.1.: Planar example robot with frames attached to the links.

Listing B.1: Code generator for planar forward kinematics and Jacobians

```
from sympy import *
from functools import reduce
from argparse import ArgumentParser


def create_cartesian_forward_kinematics_expr(dof):
    """Create symbolical expressions for the forward kinematics and
       Jacobian of a n-dim planar robot"""

    def generate_trafo(q, l):
        """Create a planar homogeneous transformation matrix for one joint"""
        return Matrix([
            [cos(q), -sin(q), l*cos(q)],
            [sin(q), cos(q), l*sin(q)],
            [0 , 0, 1]
        ])

    q = [Symbol(f'q[{i}]') for i in range(dof)]
    l = [Symbol(f'l[{i}]') for i in range(dof)]

    # Chain transformations to compute the end-effector pose:
    trafos = (generate_trafo(qi, li) for qi, li in zip(q, l))
    tcp = reduce(lambda x, y: x @ y, trafos)

    # Forward kinematics f(q) = [x, y, phi]:
    fkin = Matrix([
        tcp[0, 2],
        tcp[1, 2],
        sum(q)
    ])

    # Differentiate fkin wrt. qi:
    J = Matrix([diff(fkin, qi).T for qi in q]).T

    return fkin, J


if __name__ == '__main__':
    parser = ArgumentParser(description="Planar Robot Code Generator")
    parser.add_argument("dof", type=int, help="Degrees of Freedom")
    args = parser.parse_args()

    fkin, J = create_cartesian_forward_kinematics_expr(args.dof)

    def sympy_to_numpycode(expr):
        """Convert sympy matrix to executable python expression with numpy arrays"""
        return str(expr).replace("Matrix", "np.array")

    print("Forward Kinematics:")
    print(sympy_to_numpycode(fkin))

    print("Jacobian:")
    print(sympy_to_numpycode(J))
```