TECHNISCHE UNIVERSITÄT WIEN

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Learning-based Motion Planning for a Timber Crane

DIPLOMARBEIT

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Associate Prof. Dr.-Ing. W. Kemmetmüller
Projektass. Dipl.-Ing. M. Ecker

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by

Marlene Birkelbach
Matriculation number 11704921

Vienna, in May 2024

Complex Dynamical Systems Group
A-1040 Wien, Gußhausstr. 27–29, Internet: https://www.acin.tuwien.ac.at

Preamble

First and foremost, I would like to express my deepest gratitude to my family. To my parents, whose support has made my studies possible, thank you. Your encouragement and sacrifices have been the foundation of my academic journey.

I extend my heartfelt thanks to my sister, Laura, for her emotional support throughout various life challenges, which she has provided consistently throughout my life. My brother Felix deserves special mention for the inspiring academic discussions and for always encouraging me in my studies, thank you.

I am also deeply thankful to all my fellow students, who have become good friends over the years. Thank you for accompanying me through the highs and lows, for the shared learning experiences and for the many celebrations. I will always cherish the memories of our time together.

Furthermore, I would like to express my sincere gratitude to my advisor, Marc Ecker, for his exceptional support and the time and effort he dedicated to guiding this thesis. I am also grateful to Bernhard Bischof for his valuable insights and suggestions. Finally, I extend my thanks to Professor Kemmetmüller for his helpful feedback and supervision of my work.

Vienna, in May 2024

Abstract

Motion planning is an essential part of robotics research, requiring algorithms that are computationally efficient and adaptable to varying environments. Using machine learning methods for motion planning can provide solutions to these challenges. In this thesis, imitation learning methods for motion planning are investigated, utilizing artificial neural networks to implement these techniques. The networks generate trajectories by imitating two algorithms: Via-point-based Stochastic Trajectory Optimization and the solution to an optimal control problem. These learning-based motion planning methods are applied to a timber crane across different environmental settings. The results demonstrate that the networks are generally able to learn the underlying task through behavioral cloning and adapt to varying obstacle heights. They also show a significant advantage regarding computational speed over the original algorithms. However, in the more complex scenario with two movable obstacles, further improvements are required.

Kurzzusammenfassung

Die Bewegungsplanung ist ein wichtiger Teil der Robotikforschung und erfordert Algorithmen, die recheneffizient sind und sich an unterschiedliche Umgebungen anpassen können. Die Verwendung von Machine Learning Ansätzen für die Bewegungsplanung kann Lösungen für diese Herausforderungen bieten. In dieser Arbeit werden Imitation Learning Methoden für die Bewegungsplanung untersucht, die mit künstlichen neuronalen Netzen implementiert werden. Die Netze generieren Trajektorien, indem sie zwei Algorithmen imitieren: Via-point-based Stochastic Trajectory Optimization und die Lösung eines Optimalsteuerungsproblems. Diese lernbasierten Bewegungsplanungsmethoden werden auf einen Holzkran in verschiedenen Umgebungsbedingungen angewandt. Die Ergebnisse zeigen, dass die Netzwerke im Allgemeinen in der Lage sind, die zugrundeliegende Aufgabe durch Imitieren des Verhaltens zu erlernen und sich an unterschiedliche Hindernishöhen anzupassen. Sie zeigen auch einen deutlichen Vorteil hinsichtlich der Rechengeschwindigkeit gegenüber den ursprünglichen Algorithmen. In dem komplexeren Szenario mit zwei beweglichen Hindernissen sind jedoch weitere Verbesserungen erforderlich.

Contents

List of Figures

1 Introduction

1.1 Motivation

Motion planning is a crucial part of research in the field of robotics [1, 2]. Its main objective is to find a trajectory from a starting configuration to a desired goal configuration without colliding with obstacles in the workspace or the robot itself while complying with system limitations. A popular approach to solve motion planning problems are gradient-based algorithms that solve an optimization problem for generating trajectories [3–5]. However, these approaches show their limitations when obstacles are present, as in most real-world settings. Including obstacles in the optimization problem makes it non-convex, causing gradient-based approaches to get stuck in local minima, often resulting in infeasible trajectories. Sampling-based approaches, on the other hand, are able to find a global optimum but can be computationally expensive and don't scale well with dimensions [6]. This calls for computationally efficient approaches that yield feasible trajectories, even in the presence of obstacles.

A promising domain to speed up motion planning and make it adaptable to obstacles and changes in the environment is to use machine learning methods for motion planning [7, 8]. The intersection of these two fields of research holds the potential to significantly increase computational efficiency and adaptability to different scenarios. Using machine learning provides robotic systems with the capability to learn from data and use experience from successfully solved motion tasks, thus offering a solution to the challenges posed by real-time online motion planning in varying environments [9].

Research in learning-based motion planning often centers on imitation learning methods, aiming to develop policies that replicate the behavior of motion planning algorithms while decreasing their computational complexity [10, 11]. Many of these approaches employ artificial neural networks to encode information from successfully solved planning problems. The expectation is that neural networks can learn a generalized understanding of feasible trajectories, enabling them to make meaningful predictions even in unseen scenarios [12]. Therefore, the intersection of imitation learning and neural networks presents a compelling field of research worthy of further exploration.

This thesis deals with the theoretical background of imitation learning methods for robot motion planning using artificial neural networks and their application on a timber crane.

1.2 Related Work

Imitation learning in the field of robotics emerged in the 1980s as a promising alternative to conventional manual programming of robots [13]. Among the pioneering works in

imitation learning is the renowned ALVINN project, which dates back to 1988 [14]. In this project, imitation learning was applied to train an artificial neural network, enabling it to translate input images into corresponding actions for the task of road following in autonomous driving scenarios. In the late 1990s, the task of trajectory planning and learning robot motion became a key domain of imitation learning [15]. Nevertheless, a lot of research projects predominantly concentrate on learning trajectories demonstrated by a human expert [16, 17]. This thesis, however, focuses on learning trajectories from motion planning algorithms, an area that has been researched in some recent works.

Qureshi et al. introduced the concept of Motion Planning Networks (MPNet), which integrates classical motion planning with learning-based methods [18, 19]. Their approach involves the utilization of two artificial neural networks to learn the heuristics employed by an RRT* planner for generating feasible trajectories. The encoder network (Enet) captures environment information and embeds it into a latent space. The planning network (Pnet) uses the environment encoding from Enet, along with the current and goal states of the robot, to predict the next state, leading the robot towards the goal region. MPNet recursively calls itself to generate the complete state trajectory. Both Enet and Pnet are designed as feedforward neural networks and are trained using a mean-squared-error loss function that compares predicted and target states.

Tenhumberg et al. use a neural network to encode successfully solved motion planning problems across diverse tasks and environments [12]. The neural network's role is to provide an educated initial guess for an optimization-based motion planner, thereby accelerating computation time while ensuring both feasibility and smoothness. The authors introduce the concept of Basis Point Set (BPS) to neural motion planning, offering a way to encode environment information such that it can be directly used as an input for the network. Through supervised training, the network learns to predict a feasible path based on the environment encoding, as well as the specified start and goal configurations. The study demonstrates that this learning-based planning approach also works for previously unseen environments.

Carius et al. present another imitation learning approach, MPC-Net, in their work [20]. This method emulates Model Predictive Control (MPC) algorithms by introducing a distinctive training strategy. Rather than minimizing a distance metric between predictions and labeled expert data, MPC-Net employs the control Hamiltonian as its loss function. This has the effect that the network is trained to directly solve the optimal control problem. Notably, the control Hamiltonian contains constraints, allowing for the consideration of obstacles in the environment during training. In [21], MPC-Net is extended for controlling multiple gaits of a walking robot with only a single trained policy by employing a mixture-of-experts network.

1.3 Contribution and Structure of this Thesis

This thesis deals with learning-based motion planning, employing behavioral cloning methods using artificial neural networks. These methods are applied to a timber crane to evaluate whether they allow for generalizing from diverse scenarios to apply motion planning to arbitrary start and goal configurations as well as arbitrary environment

geometries.

In Chapter 2, the two motion planning algorithms that are used as demonstrators for the neural networks are described. These are Via-point-based Stochastic Trajectory Optimization (VP-STO) and an optimal control formulation.

Chapter 3 covers the theory behind imitation learning concepts and artificial neural networks.

The application on the timber crane is discussed in Chapter 4. The system and training setup are described and subsequently, the results of the learning-based motion planning approaches are presented and evaluated.

2 Motion Planning

Motion planning is a fundamental part of robot control, dedicated to generating a trajectory that guides the robot from its starting configuration to a desired goal configuration. Additionally, collisions with obstacles or the robot itself have to be avoided while complying with system constraints. Gradient-based approaches are widely adopted techniques to address this challenge [3–5]. Another alternative for generating optimal trajectories are path integral methods, introducing stochastic elements to trajectory optimization [22].

In the context of this thesis, two different planners serve as expert demonstrators for training imitation learning models. The following sections focus on the approaches utilized in this thesis, providing a detailed description and comparison. In Section 2.1, motion planning with via-point based stochastic trajectory optimization is presented and an optimal control formulation of the planning problem, which is solved with gradient-based methods, is described in Section 2.2.

2.1 Via-point-based Stochastic Trajectory Optimization

Via-point-based Stochastic Trajectory Optimization (VP-STO) aims to generate smooth and time-optimal robot trajectories in joint space [23]. The core task is to find N via-points, denoted as $\mathbf{q}_{\text{via}}^{\text{T}} = \left[\mathbf{q}_1^{\text{T}}, \dots, \mathbf{q}_N^{\text{T}}\right] \in \mathbb{R}^{DN}$, which the trajectory has to pass through. Here, $\mathbf{q} \in \mathbb{R}^D$ denotes the robot joint configuration with D being the number of degrees of freedom. The via-points are obtained by employing Covariance Matrix Adaptation (CMA-ES), a stochastic black-box optimization algorithm, to minimize a specified cost function c of the form

$$\min_{\mathbf{q}_{\text{via}}} \int_0^1 c(\mathbf{q}(s), \dot{\mathbf{q}}(s), \ddot{\mathbf{q}}(s), T)\, ds, \tag{2.1}$$

with the robot configuration, velocity and acceleration $\mathbf{q}(s) \in \mathbb{R}^D, \dot{\mathbf{q}}(s) \in \mathbb{R}^D$ and $\ddot{\mathbf{q}}(s) \in \mathbb{R}^D$, respectively, and the total duration of the trajectory T. The phase variable $s \in [0, 1]$ maps to the time t with $s = \frac{t}{T}$.

The algorithm's first step is to sample M sets of via-points from a Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_{\text{via}}, \boldsymbol{\Sigma}_{\text{via}})$. In the first pass, this distribution is initialized with a straight-line guess between the defined start and goal configuration for the mean $\boldsymbol{\mu}_{\text{via}} \in \mathbb{R}^{DN}$ with high uncertainty, i. e. large diagonal values for $\boldsymbol{\Sigma}_{\text{via}} \in \mathbb{R}^{DN \times DN}$. Subsequently, the sampled sets of via-points are transformed into a population of M trajectories by incorporating cubic splines. Alongside the via-points and specified boundary conditions on the start and goal configuration and velocity, the computation of the trajectories only requires the total movement duration T. T can be obtained by finding the minimum positive movement duration such that the resulting trajectory velocity and acceleration stay within the defined limits. This is done by calculating $T_k(\mathbf{q}_{\text{via}})$ for K uniformly distributed evaluation

points in the phase state $[0, 1]$ and taking the most conservative value, i. e. the maximum value, to guarantee that the constraints are satisfied at each evaluation point k. This results in a set of M kinodynamically admissible trajectories with minimal movement duration. Finally, the mean $\boldsymbol{\mu}_{\text{via}}$ and the covariance matrix $\boldsymbol{\Sigma}_{\text{via}}$ are updated based on the performance of the obtained trajectories according to the specified cost function. By repeating these steps, the location of the via-points is optimized.

In this thesis, the cost function is defined as the total duration of the trajectory. Additionally, a high value is added to the cost if the trajectory collides with an obstacle or exceeds joint limits. Collision detection is implemented using the GJK algorithm, see [24, 25] for a detailed description. The boundary conditions and limitations on the joint velocity and acceleration are satisfied by design, as described above.

This approach for generating trajectories only considers the kinematic structure of the system. Dynamic effects caused by non-actuated joints, for example, are not taken into account.

2.2 Optimal Control Formulation

A discrete-time optimal control problem can be formulated as

$$\begin{aligned} \min_{\mathbf{u}_0,\dots,\mathbf{u}_{N-1}} \quad & J(\mathbf{u}_0,\dots,\mathbf{u}_{N-1}) \\ \text{s.t. } & \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k,\mathbf{u}_k)\,, \qquad k = 1,\dots,N-1, \\ & \mathbf{g}_k(\mathbf{x}_k,\mathbf{u}_k) = \mathbf{0} \\ & \mathbf{h}_k(\mathbf{x}_k,\mathbf{u}_k) \leq \mathbf{0}, \end{aligned} \tag{2.2}$$

with the system's state $\mathbf{x}_k \in \mathbb{R}^n$, the control input $\mathbf{u}_k \in \mathbb{R}^m$, the continuously differentiable system dynamics $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ and the continuously differentiable equality and inequality contraints $\mathbf{g}_k : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^p$ and $\mathbf{h}_k : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^q$, respectively. The equality constraints include path constraints, e. g. to enforce that the system follows a specific trajectory, while the inequality constraints are used to comply with actuator limitations and avoid obstacles, for example. The starting value $\mathbf{x}_0$ is fixed. The cost function

$$J(\mathbf{u}_0,\dots,\mathbf{u}_{N-1}) = \ell_N(\mathbf{x}_N) + \sum_{k=0}^{N-1} \ell_k(\mathbf{x}_k,\mathbf{u}_k) \tag{2.3}$$

consists of the intermediate costs $\ell_k : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}$ and the terminal cost $\ell_N : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}$. Both the intermediate cost and the terminal cost are twice continuously differentiable and defined as

$$\ell_k(\mathbf{x}_k,\mathbf{u}_k) = \frac{1}{2}(\mathbf{e}_{\text{u},k}^{\text{T}}\mathbf{L}_{\text{uu},k}\mathbf{e}_{\text{u},k} + 2\mathbf{e}_{\text{x},k}^{\text{T}}\mathbf{L}_{\text{xu},k}\mathbf{e}_{\text{u},k} + \mathbf{e}_{\text{x},k}^{\text{T}}\mathbf{L}_{\text{xx},k}\mathbf{e}_{\text{x},k}) \tag{2.4}$$

and

$$\ell_N(\mathbf{x}_N) = \frac{1}{2}\mathbf{e}_{\text{x},N}^{\text{T}}\mathbf{L}_{\text{xx},N}\mathbf{e}_{\text{x},N}, \tag{2.5}$$

with the state and control errors $\mathbf{e}_{\text{x},k} = \mathbf{x}_k - \mathbf{x}_{\text{d},k}$ and $\mathbf{e}_{\text{u},k} = \mathbf{u}_k - \mathbf{u}_{\text{d},k}$, respectively, where $\mathbf{x}_{\text{d},k}$ and $\mathbf{u}_{\text{d},k}$ denote the desired state and desired control input. The latter can be set to

zero to minimize control effort and prevent uncontrolled behavior in the non-actuated system. $\mathbf{L}_{\mathrm{uu},k}, \mathbf{L}_{\mathrm{xu},k}$ and $\mathbf{L}_{\mathrm{xx},k}$ are weighting matrices.

The cost function can be extended to account for the constraints in the optimization problem (2.2). For this, the augmented Lagrangian is formed as

$$\begin{aligned}\mathcal{L}_{\mathrm{A}} = \ell_N(\mathbf{x}_N) + \left(\boldsymbol{\lambda}_N + \frac{1}{2}\mathbf{c}_N(\mathbf{x}_N)\mathbf{I}_{\mu,N}\right)^{\mathrm{T}} \mathbf{c}_N(\mathbf{x}_N) \\ + \sum_{k=0}^{N-1}\left[\ell_k(\mathbf{x}_k, \mathbf{u}_k) + \left(\boldsymbol{\lambda}_k + \frac{1}{2}\mathbf{c}_k(\mathbf{x}_k, \mathbf{u}_k)\mathbf{I}_{\mu,k}\right)^{\mathrm{T}} \mathbf{c}_k(\mathbf{x}_k, \mathbf{u}_k)\right],\end{aligned} \tag{2.6}$$

with the Lagrange multipliers $\boldsymbol{\lambda}_k \in \mathbb{R}^{p+q}$ and the stacked constraints $\mathbf{c}_k^{\mathrm{T}} = \left[\mathbf{g}_k^{\mathrm{T}}, \mathbf{h}_k^{\mathrm{T}}\right] \in \mathbb{R}^{p+q}$. The penalty matrix $\mathbf{I}_{\mu,k} \in \mathbb{R}^{(p+q)\times(p+q)}$ is a diagonal matrix, each diagonal value corresponding to a constraint in $\mathbf{c}_k$. The Lagrange multipliers and the penalty matrix's values are iteratively increased if a constraint is violated or close to being violated, hence increasing the overall cost. For a detailed description of these term's calculation, refer to [26].

This optimal control problem can be solved with gradient-based approaches, aiming to minimize the cost function (2.6). For this purpose, the iterative Linear Quadratic Regulator (iLQR) is used in this thesis. In contrast to VP-STO, the system dynamics are taken into account in this approach.

3 Imitation Learning

Imitation learning is a domain of machine learning, in which an agent learns to mimick an expert's behavior. The expert can be a human or, as in this thesis, an optimizing algorithm, generating optimal actions for the agent to imitate. The machine learning model that is the subject of the training is referred to as agent, e. g. an artificial neural network. It observes the expert demonstrations and learns a policy to reproduce the demonstrated behavior without explicit knowledge of the underlying optimized function employed by the algorithm [15].

A policy π can be represented in different ways. One option is to define a trajectory-level policy

$$\pi : \mathbf{s} \mapsto \boldsymbol{\tau}, \tag{3.1}$$

mapping a context $\mathbf{s}$ to a trajectory $\boldsymbol{\tau}$ [27]. The context contains information required for the task, e. g. desired start and end position or information about the system's environment. The trajectory consists of a sequence of K states and/or control inputs

$$\boldsymbol{\tau} = [\mathbf{x}_0, \mathbf{u}_0, \dots, \mathbf{x}_K, \mathbf{u}_K]. \tag{3.2}$$

Alternatively, the policy can be formulated as

$$\pi : \mathbf{x}_k, \mathbf{s} \mapsto \mathbf{u}_k, \tag{3.3}$$

mapping the system's current state $\mathbf{x}_k$ and contextual information $\mathbf{s}$ to an action, which, in this case, is the control input $\mathbf{u}_k$, for every step k. This is referred to as action-state space representation [28].

Another distinction between policies can be made with regard to the feedback of the state. Feedback policies take the changes of the environment and the system into account, which were caused by the previous action, when determining the next action [16]. With feedback-free policies, on the other hand, the action is based solely on the initial input [29].

The expert's policy π^{E} results from the optimization of a cost or reward function, which represents the quality of a given action or trajectory. This function is generally unknown to the agent but it is used to generate the expert demonstrations for the training data $\mathcal{D}$. This dataset comprises pairs of inputs and outputs, denoted as $(\boldsymbol{\xi}_i, \mathbf{y}_i)$, each representing one of the n samples in the training data. The input $\boldsymbol{\xi}_i$ includes the current state of the system and the context, the output $\mathbf{y}_i$ can take the form of a trajectory or an action, depending on the chosen policy representation. The agent leverages these expert demonstrations to estimate a policy π^{A} that replicates the expert's behavior [30, 31].

There are two major approaches to how this policy is obtained, behavioral cloning (BC) and inverse reinforcement learning (IRL). BC methods involve learning a direct mapping

from the input to the optimal action, while in IRL, the policy is obtained by recovering and optimizing a reward function from the expert's demonstrations. In this thesis, only BC methods are considered. They are described in the following section.

3.1 Behavioral Cloning

The aim of behavioral cloning (BC) methods is to learn a direct mapping from the context to the optimal trajectory as in (3.1) or from the current state and the context to the optimal action as in (3.3) without recovering the underlying reward function. These methods represent an efficient strategy for emulating the expert's behavior as they can be formulated as a supervised learning problem and solved by minimizing the difference between the expert's demonstrations and the learned policy with respect to a specified loss function [15].

3.2 Artificial Neural Networks

Artificial neural networks (ANNs) are computational models that are often used to represent the agent's policy. They are inspired by the neuronal structure of the animals' and humans' nervous system.

3.2.1 The Neuron

Neurons are the fundamental cells of our central nervous system. They process and forward impulses, which are transferred to other neurons by so called synapses, the connections between neurons. ANNs consist of artificial neurons connected by artificial synapses, similar to the organization in biological neural networks [32].

The model for an artificial neuron consists of multiple input signals represented by the vector $\boldsymbol{\xi} = [\xi_1, ..., \xi_n]^{\mathrm{T}} \in \mathbb{R}^n$, the synapses represented by weights $\mathbf{w} = [w_1, ..., w_n]^{\mathrm{T}} \in \mathbb{R}^n$, a bias $b \in \mathbb{R}$, an activation function $g : \mathbb{R} \mapsto \mathbb{R}$ and the output signal $y \in \mathbb{R}$. The input signals represent the values coming from the application for which the network is used. Each signal ξ_i is multiplied with the respective weight w_i according to its relevance for the neuron. The weighted inputs are summed up and the bias is added as an additional parameter. The activation function's role is to limit the neuron output to a reasonable range and can be used to add non-linear behavior to the model. The output signal is computed as

$$y = g(\mathbf{w}^{\mathrm{T}}\boldsymbol{\xi} + b), \tag{3.4}$$

which in turn can be used as input signal for another neuron.

3.2.2 Multilayer Perceptron

Multiple neurons can be combined to a network according to a specific architecture describing how the different neurons are arranged in relation to each other. In this thesis, a multilayer feedforward architecture, commonly known as multilayer perceptron (MLP), is employed. It consists of an input layer, responsible for receiving the input signals

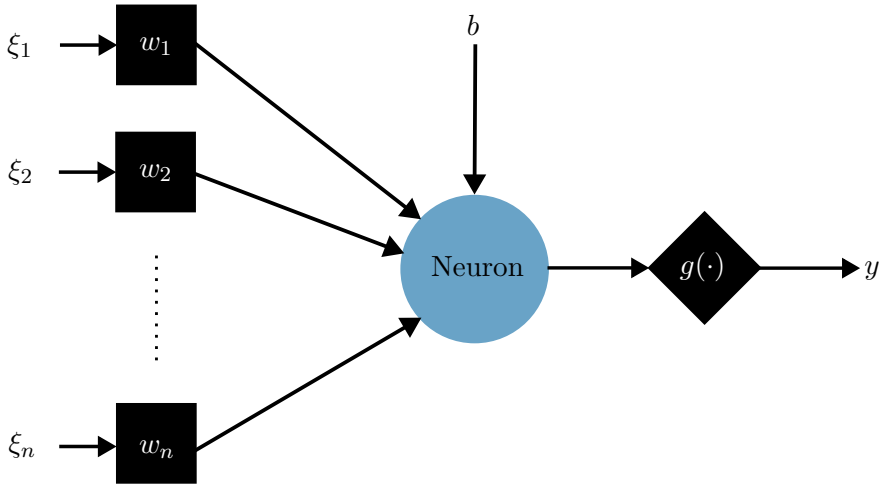


Figure 3.1: Structure of an artificial neuron

according to the specific application of the network. This is followed by one or more so called hidden layers, each comprising multiple neurons, constructed as explained before. The output signals of the first hidden layer serve as the input signals to the second hidden layer and so on. The hidden layers are responsible for extracting information and patterns from the input data, building a representation of the application's environment. The last part is the output layer, also composed of neurons, representing the network's final output resulting from the processed data within the ANN. The number of hidden layers and neurons in each layer depend on the complexity of the application as well as the quality and quantity of the data available [33, 34].

The term "feedforward" describes how information is processed in the network. The input signals are propagated layer by layer until they reach the output layer, allowing the flow of information in only one direction during execution. Layer l, with $l = 1, \ldots, L$, is constructed of n_l neurons, represented by each neuron's weights $\mathbf{w}_1, \ldots, \mathbf{w}_{n_l}$ and their biases $b_1, \ldots, b_{n_l}$, which can be combined to the weight matrix $\mathbf{W}^{(l)} = [\mathbf{w}_1, \ldots, \mathbf{w}_{n_l}]^{\mathrm{T}}$ and the bias vector $\mathbf{b}^{(l)} = [b_1, \ldots, b_{n_l}]^{\mathrm{T}}$. The weight matrix's elements $W_{ji}^{(l)}$ describe the connection between the jth neuron of layer l and the ith neuron of layer $l-1$. Each layer l is also equipped with an activation function $\mathbf{g}^{(l)} : \mathbb{R}^{n_l} \mapsto \mathbb{R}^{n_l}$, where $\mathbf{g}(\mathbf{y}) = [g(y_1), \ldots, g(y_{n_l})]^{\mathrm{T}}$. Consequently, the output of the first layer $\mathbf{z}^{(1)}$ can be computed as

$$\mathbf{z}^{(1)} = \mathbf{g}^{(1)}(\mathbf{W}^{(1)}\boldsymbol{\xi} + \mathbf{b}^{(1)}) \tag{3.5}$$

and the outputs of the following layers are obtained one after another by evaluating

$$\mathbf{z}^{(l)} = \mathbf{g}^{(l)}(\mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}), \qquad l = 2, ..., L, \tag{3.6}$$

with $\mathbf{z}^{(L)} = \mathbf{y}$ being the output of the network.

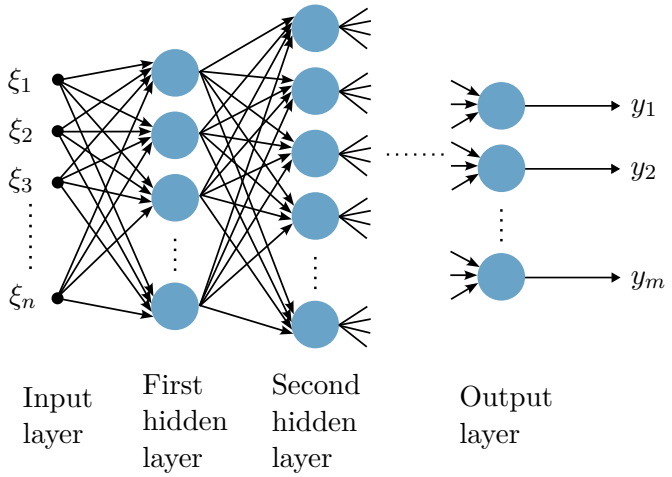


Figure 3.2: Structure of the Multilayer Perceptron

Training

To ensure that the ANN provides the correct output for each sample of the input data, the network has to be trained, requiring the adaptation of all its parameters, i. e. the weights and biases associated with each neuron. This training procedure requires multiple input samples alongside their corresponding desired output signals, constituting the training data. The training algorithm for MLPs is known as backpropagation, a two-part process [35].

In the first stage, a single input data sample is introduced into the network and propagated through its layers as described before. The network generates an output in accordance with the current weights and biases, which remain unchanged during this training phase. Subsequently, in the second stage, the network's output is compared against the desired output, and the objective is to adjust the parameters systematically to minimize the output error. This adjustment is achieved by an optimizer. The optimizer computes the gradient of the error with respect to the parameters layer by layer, but now starting from the last layer and propagating backwards, hence the name backpropagation. With the parameters for layer l being represented by $\mathbf{\Theta}^{(l)}$ and the scalar output error value E, the gradients are computed as

$$\Delta \mathbf{E}^{(l)} = \frac{\partial E}{\partial \mathbf{\Theta}^{(l)}} \tag{3.7}$$

and the parameters are adjusted according to the delta rule

$$\mathbf{\Theta}^{(l)} \leftarrow \mathbf{\Theta}^{(l)} - \eta \, \Delta \mathbf{E}^{(l)}, \qquad l = L, ..., 1 \tag{3.8}$$

with the learning rate η. This is done for each layer until all parameters are updated. Repeating this procedure for every sample of the training dataset is called stochastic gradient descent (SGD).

Another popular optimizer used for training artificial neural networks is Adam, which is short for Adaptive Moment Estimation [36]. Adam adjusts the learning rate for each

parameter in the model based on estimates of the first and second moments of the gradients, which are computed as

$$\mathbf{m}^{(l)} \leftarrow \beta_1 \mathbf{m}^{(l)} + (1-\beta_1)\Delta\mathbf{E}^{(l)} \tag{3.9a}$$

$$\mathbf{v}^{(l)} \leftarrow \beta_2 \mathbf{v}^{(l)} + (1-\beta_2)(\Delta\mathbf{E}^{(l)})^2, \tag{3.9b}$$

where $\mathbf{m}^{(l)}$ and $\mathbf{v}^{(l)}$ are the estimates of the first and second order moments, respectively, and β_1 and β_2 are decay rates. $(\Delta\mathbf{E}^{(l)})^2$ indicates the elementwise square. These moments are bias-corrected, yielding

$$\hat{\mathbf{m}}^{(l)} \leftarrow \frac{\mathbf{m}^{(l)}}{1-\beta_1^t} \tag{3.10a}$$

$$\hat{\mathbf{v}}^{(l)} \leftarrow \frac{\mathbf{v}^{(l)}}{1-\beta_2^t}, \tag{3.10b}$$

where t indicates the number of steps already taken, i. e. the training sample's number. Finally, the parameters are updated according to

$$\mathbf{\Theta}^{(l)} \leftarrow \mathbf{\Theta}^{(l)} - \eta \, \frac{\hat{\mathbf{m}}^{(l)}}{\hat{\mathbf{v}}^{(l)}}, \tag{3.11}$$

where the division of the two vectors indicates the elementwise division. In case of high uncertainty, the second order moment will be high, leading to a smaller step size in parameter space, which is a form of automatic annealing. Another property of this optimizer is that the parameter updates are invariant to rescaling of the gradient, allowing for more efficient convergence.

A complete iteration over all samples in the training dataset is referred to as an epoch. The algorithm can be applied for more than one epoch, randomly shuffling the samples during each pass. Instead of considering each individual sample to update the parameters, the gradient can also be computed based on a mini-batch, consisting of a predetermined number of randomly selected samples from the training dataset. In this case, the average gradient for the entire batch is used to update the ANN's parameters [37].

Training performance strongly depends on the choice of hyperparameters such as the number and size of hidden layers, batch size and number of epochs.

4 Application on a Timber Crane

In this chapter, learning-based motion planning approaches are applied to a timber crane. The crane model as described in [38] is presented in Section 4.1. In Section 4.2 the findings of the simulations are presented and discussed. All experiments are performed on an Intel Core i7-7600U CPU @ 2.8 GHz.

4.1 Crane Model

The timber crane presented in Figure 4.1 has eight degrees of freedom (DoFs). However, the jaw angle q_8 of the gripper is specified as a constant value to reduce the model's complexity for planning. Consequently, the model used for the following simulations only has seven degrees of freedom, whereby $\mathbf{q}_a = [q_1, q_2, q_3, q_4, q_7]^{\mathrm{T}}$ are actuated joints and $\mathbf{q}_{na} = [q_5, q_6]^{\mathrm{T}}$ are non-actuated. As shown in Figure 4.1, all DoFs are revolute joints, except for one prismatic joint q_4.

4.1.1 Kinematics

To describe the crane's kinematics, transformation matrices of the form

$$\mathbf{H}_i^j = \begin{bmatrix} \mathbf{R}_i^j & \mathbf{d}_i^j \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \in \mathcal{SE}(3) \tag{4.1}$$

are used, with the rotation matrix $\mathbf{R}_i^j \in \mathcal{SO}(3)$ and the translation vector $\mathbf{d}_i^j \in \mathbb{R}^3$. They describe the transformation from a coordinate frame $\mathcal{F}_j$ to a coordinate frame $\mathcal{F}_i$, corresponding to the joints j and i, respectively. The frames are defined using the Denavit-Hartenberg convention, for a detailed explanation see [39]. Hence, the transformation matrices from a coordinate frame $\mathcal{F}_j$ to its predecessor $\mathcal{F}_{j-1}$ can be obtained through the four Denavit-Hartenberg parameters θ_j, d_j, a_j and α_j according to

$$\mathbf{H}_{j-1}^j = \mathbf{H}_{Rz}(\theta_j)\mathbf{H}_{Tz}(d_j)\mathbf{H}_{Tx}(a_j)\mathbf{H}_{Rx}(\alpha_j), \tag{4.2}$$

with $\mathbf{H}_{Ri}(\phi)$ describing a rotation of ϕ around the i-axis and $\mathbf{H}_{Ti}(s)$ describing a translation of s in the direction of the i-axis. The transformation from coordinate frame $\mathcal{F}_{11}$ poses an exception as it is defined to frame $\mathcal{F}_8$ instead of $\mathcal{F}_{10}$, with the parameters $\theta_{11}, d_{11}, a_{11}$ and α_{11}. Consequently, the transformation matrix $\mathbf{H}_i^j, 0 \leq i < j$ is obtained by computing

$$\mathbf{H}_i^j = \begin{cases} \Pi_{k=i+1}^{j} \mathbf{H}_{k-1}^k & \text{for } j \leq 10, \\ \left(\Pi_{k=i+1}^{8} \mathbf{H}_{k-1}^k\right) \mathbf{H}_8^{11} \mathbf{H}_{11}^j & \text{for } 11 \leq j \leq 12. \end{cases} \tag{4.3}$$

The Denavit-Hartenberg parameters for the timber crane are presented in Table 4.1.

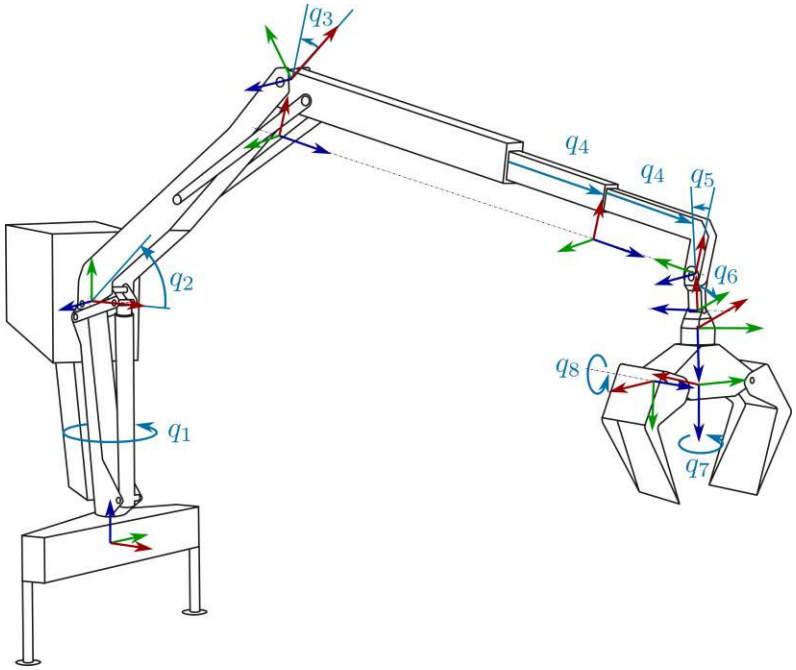


Figure 4.1: Kinematic chain of the timber crane

Table 4.1: Denavit-Hartenberg parameters for the timber crane [38].

j	θ_j in rad	d_j in m	a_j in m	α_j in rad
1	q_1	2.425	0.180	$\frac{\pi}{2}$
2	q_2	0	3.493	0
3	q_3	0	-0.393	$\frac{\pi}{2}$
4	0	q_4	0	0
5	0	q_4	0	$-\frac{\pi}{2}$
6	q_5	0	-0.213	$-\frac{\pi}{2}$
7	q_6	0	0	$-\frac{\pi}{2}$
8	q_7	0.578	0	0
9	$-\frac{\pi}{2}$	0	0.340	$\frac{\pi}{2}$
10	$\frac{\pi}{2}$	0	0.857	0
11	$\frac{\pi}{2}$	0	0.325	$\frac{\pi}{2}$
12	$\frac{\pi}{2}$	0	0.857	0

4.1.2 Dynamics

The equations of motion for the timber crane can be described as

$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) + \mathbf{F}\dot{\mathbf{q}} = \boldsymbol{\tau}, \tag{4.4}$$

with the joint coordinates $\mathbf{q} \in \mathbb{R}^D$, the positive definite mass matrix $\mathbf{D}(\mathbf{q}) \in \mathbb{R}^{D\times D}$, the Coriolis matrix $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}}) \in \mathbb{R}^{D\times D}$, the gravitational vector $\mathbf{g}(\mathbf{q}) \in \mathbb{R}^D$, the diagonal matrix $\mathbf{F} = \operatorname{diag}(\boldsymbol{\mu}_v) \in \mathbb{R}^{D\times D}$ with the viscous friction coefficients $\boldsymbol{\mu}_v \in \mathbb{R}^D$ and the generalized joint forces $\boldsymbol{\tau} \in \mathbb{R}^D$ [40].

With the joint coordinates comprising the actuated joints $\mathbf{q}_a \in \mathbb{R}^{D_a}$ as well as the non-actuated joints $\mathbf{q}_{na} \in \mathbb{R}^{D_{na}}$, i. e.

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_a \\ \mathbf{q}_{na} \end{bmatrix}, \tag{4.5}$$

the equations (4.4) can be partitioned into

$$\underbrace{\begin{bmatrix} \mathbf{D}_a(\mathbf{q}) & \mathbf{D}_{ana}(\mathbf{q}) \\ \mathbf{D}_{naa}(\mathbf{q}) & \mathbf{D}_{na}(\mathbf{q}) \end{bmatrix}}_{\mathbf{D}(\mathbf{q})} \begin{bmatrix} \ddot{\mathbf{q}}_a \\ \ddot{\mathbf{q}}_{na} \end{bmatrix} + \left(\underbrace{\begin{bmatrix} \mathbf{C}_a(\mathbf{q},\dot{\mathbf{q}}) \\ \mathbf{C}_{na}(\mathbf{q},\dot{\mathbf{q}}) \end{bmatrix}}_{\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})} + \underbrace{\begin{bmatrix} \mathbf{F}_a \\ \mathbf{F}_{na} \end{bmatrix}}_{\mathbf{F}} \right) \dot{\mathbf{q}} + \underbrace{\begin{bmatrix} \mathbf{g}_a(\mathbf{q}) \\ \mathbf{g}_{na}(\mathbf{q}) \end{bmatrix}}_{\mathbf{g}(\mathbf{q})} = \underbrace{\begin{bmatrix} \boldsymbol{\tau}_a \\ \mathbf{0} \end{bmatrix}}_{\boldsymbol{\tau}}, \tag{4.6}$$

as the generalized joint forces of the non-actuated joints are zero.

For the discrete time state-space formulation required for (2.2), the state is defined as

$$\mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{bmatrix} = \begin{bmatrix} \mathbf{q}_a \\ \mathbf{q}_{na} \\ \dot{\mathbf{q}}_a \\ \dot{\mathbf{q}}_{na} \end{bmatrix}. \tag{4.7}$$

The acceleration of the actuated joint coordinates serves as the control input

$$\mathbf{u} = \ddot{\mathbf{q}}_a, \tag{4.8}$$

which is possible due to the presence of a subordinate velocity controller. The expression for the non-actuated joints' acceleration can be derived from the second line in (4.6):

$$\ddot{\mathbf{q}}_{na} = -\mathbf{D}_{na}(\mathbf{q})^{-1} \left(\mathbf{D}_{naa}(\mathbf{q})\ddot{\mathbf{q}}_a + \left(\mathbf{C}_{na}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{F}_{na} \right) \dot{\mathbf{q}} + \mathbf{g}_{na}(\mathbf{q}) \right). \tag{4.9}$$

Assuming $\mathbf{u}$ as well as $\ddot{\mathbf{q}}_{na}$ to be constant during the time step k yields the discrete time state equations for $\mathbf{x}_{k+1}$ as defined in (4.7)

$$\mathbf{q}_{a,k+1} = \mathbf{q}_{a,k} + \dot{\mathbf{q}}_{a,k} T_s + \frac{1}{2}\mathbf{u}_k T_s^2 \tag{4.10a}$$

$$\mathbf{q}_{na,k+1} \approx \mathbf{q}_{na,k} + \dot{\mathbf{q}}_{na,k} T_s + \frac{1}{2}\ddot{\mathbf{q}}_{na,k} T_s^2 \tag{4.10b}$$

$$\dot{\mathbf{q}}_{a,k+1} = \dot{\mathbf{q}}_{a,k} + \mathbf{u}_k T_s \tag{4.10c}$$

$$\dot{\mathbf{q}}_{na,k+1} \approx \dot{\mathbf{q}}_{na,k} + \ddot{\mathbf{q}}_{na,k} T_s \tag{4.10d}$$

with the sampling time $T_s > 0$ and $\mathbf{q}_{a,k} = \mathbf{q}_a(kT_s)$, $\dot{\mathbf{q}}_{a,k} = \dot{\mathbf{q}}_a(kT_s)$, $\mathbf{u}_k = \mathbf{u}(kT_s)$, $\mathbf{q}_{na,k} = \mathbf{q}_{na}(kT_s)$, $\dot{\mathbf{q}}_{na,k} = \dot{\mathbf{q}}_{na}(kT_s)$ and $\ddot{\mathbf{q}}_{na,k} = \ddot{\mathbf{q}}_{na}(kT_s)$.

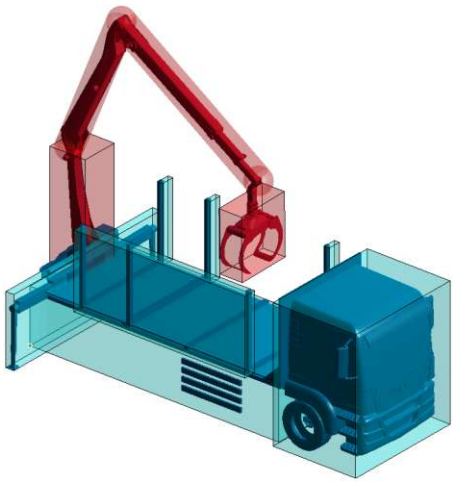


Figure 4.2: Collision model for the timber crane

4.1.3 Inequality Constraints

In the following simulations, the timber crane is mounted on a truck. For the purpose of collision avoidance, this truck is represented by static oriented bounding boxes, as illustrated in blue in Figure 4.2. The crane's slewing column and gripper are also modeled using bounding boxes, while the remainder of the crane is represented by two cylinders. The crane is highlighted in red. There is one collision constraint for each combination of crane link and truck object. As the crane's slewing column is only relevant for collision checking if the grab holds a log, it is sufficient to consider the three other links of the crane in the log-free case. For 3 crane links and the 7 truck objects presented in Figure 4.2, this makes 21 collision constraints. The minimal distances between the crane and the collision objects, i. e. the truck, are calculated using the GJK algorithm, see [24, 25]. The negative minimal distances between each crane link and truck object are stacked into the collision inequality vector

$$\begin{bmatrix} h_1(\mathbf{q}) \\ \vdots \\ h_{21}(\mathbf{q}) \end{bmatrix} = \mathbf{h}_{\text{collision}}(\mathbf{q}) \leq \mathbf{0}, \tag{4.11}$$

i. e. a positive value for h_i indicates a collision.

Self-collision is avoided through limits on the joint angles. Alongside limitations on the achievable joint velocity, they are implemented in the state inequality vector

$$\begin{bmatrix} \mathbf{x} - \overline{\mathbf{x}} \\ \underline{\mathbf{x}} - \mathbf{x} \end{bmatrix} = \mathbf{h}_{\text{state}}(\mathbf{x}) \leq \mathbf{0}, \tag{4.12}$$

where $\overline{\mathbf{x}}$ and $\underline{\mathbf{x}}$ are the upper and lower state limit, respectively.

Similarly, the control inequality vector is defined, imposing limitations on the control input as

$$\begin{bmatrix} \mathbf{u} - \overline{\mathbf{u}} \\ \underline{\mathbf{u}} - \mathbf{u} \end{bmatrix} = \mathbf{h}_{\text{control}}(\mathbf{u}) \leq \mathbf{0}, \tag{4.13}$$

with the upper control limit $\overline{\mathbf{u}}$ and the lower control limit $\underline{\mathbf{u}}$.

Together, this yields the overall inequality vector

$$\begin{bmatrix} \mathbf{h}_{\text{collision}}(\mathbf{q}) \\ \mathbf{h}_{\text{state}}(\mathbf{x}) \\ \mathbf{h}_{\text{control}}(\mathbf{u}) \end{bmatrix} = \mathbf{h}(\mathbf{x}, \mathbf{u}) \leq \mathbf{0}, \tag{4.14}$$

a positive value indicating the violation of an inequality constraint.

4.2 Learning-based Motion Planning

In this section, learning-based motion planning approaches are applied to the timber crane under different environmental conditions. The first task, covered in Section 4.2.1, is to solve the motion planning problem without any additional obstacles besides the truck the crane is mounted on. In Section 4.2.2, a wall with known variable height is added next to the truck. The third variant involves two additional obstacles representing trees, which can be placed arbitrarily in a specified area beside the truck. It is presented in Section 4.2.3.

The first considered learning-based approach involves imitating VP-STO, which is described in Section 2.1, through the application of behavioral cloning techniques on an artificial neural network. The training data is generated by computing via points with the VP-STO algorithm, using the parameter values specified in Table 4.2, which were chosen based on empirical performance. The input to the network consists of potential environmental information as well as the initial and desired end configuration of the five actuated joints. To reduce the amount of required data for training, only one starting configuration is considered while the end configuration varies. The network outputs the via points, which are compared to those computed by the VP-STO algorithm, utilizing a mean squared error. This network represents a trajectory-level policy as defined in (3.1), mapping contextual information to a trajectory consisting of a sequence of states. In Section 4.2.1, the context comprises of five elements each for the start and goal configuration of the actuated joints. In Section 4.2.2, the obstacle height is added to the context for environmental information and in Section 4.2.3, four elements are added to represent the position of each of the two trees in x and y direction.

Table 4.2: Parameter values for VP-STO

Parameter	Value
D	5
N	10
M	50
K	100

The second method is the imitation of the optimal control problem's solution as described in Section 2.2, with the system dynamics (4.10), the constraints (4.14) and the parameters presented in Table 4.3. The parameters where chosen based on the observed performance of different parameter combinations. The training data for this approach is generated

by solving the optimal control problem utilizing an iterative Linear Quadratic Regulator (iLQR). The iLQR is used in an MPC framework, i. e. the iLQR computes the full optimal control sequence for the specified horizon from the current to the goal state but only the first control input is applied to the system, yielding the next state. The iLQR algorithm is run again to compute the full optimal control sequence from the new state. This is done until the obtained next state is close enough to the goal state, yielding trajectories of approximately the specified horizon's length, which are 80 steps in this case. The network used here takes the current state $\mathbf{x}_k$, consisting of joint configuration and joint velocity for all seven joints, as well as the goal configuration $\mathbf{q}_{\text{goal}}$ for all joints alongside potential environmental information as an input. In contrast to the first approach, the neural network does not output the whole trajectory, but only the control input $\mathbf{u}_k$ required to lead the system one step closer to the goal state. Hence, the final trajectory is obtained by applying the control input to the system, using the resulting next state as new input to the net and repeating this process until the goal state is reached, see Figure 4.3 for an explanatory illustration. For training, the outputted control input is compared to the one computed by the iLQR for that step in a mean squares sense. In this case, the network represents an action-state space policy according to (3.3), mapping the current state and contextual information to the control input. In Section 4.2.1, the context comprises only of the goal configuration of all seven joints. For the case with one additional obstacle in Section 4.2.2, the context is extended by one element representing the obstacle height and in Section 4.2.3, four elements are added to the context to represent the x and y coordinates of each of the two trees. As before, the initial configuration is the same for all training and test samples, only the end configuration varies.

Table 4.3: Parameter values for the optimal control problem. $\mathbf{I}$ denotes the identity matrix and $\mathbf{0}$ the zero matrix.

Parameter	Value
$\mathbf{L}_{\text{uu},k}$	$0.1 \cdot \mathbf{I}_{5\times5}$
$\mathbf{L}_{\xi\text{u},k}$	$\mathbf{0}_{14\times5}$
$\mathbf{L}_{\xi\xi,k}$	$\text{diag}(l_1,\dots,l_{14})$, $l_i = \begin{cases} 0.1 & \text{for } i = 1,\dots,7 \\ 10^{-5} & \text{for } i = 8,\dots,11,14 \\ 10^{-2} & \text{for } i = 12,13 \end{cases}$
$\mathbf{L}_{\xi\xi,N}$	$10 \cdot \mathbf{I}_{14\times14}$

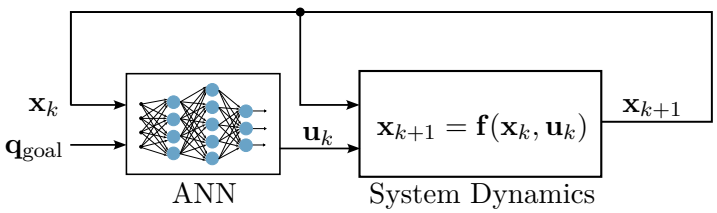


Figure 4.3: Principle of the optimal control imitation method

4.2.1 No Additional Obstacles

In the first scenario, there are no additional obstacles in the crane's workspace. Hence, there is no need for environmental information for the net's input as the truck's presence is learned implicitly.

18 000 trajectories computed utilizing the VP-STO algorithm and the iLQR algorithm for the optimal control approach serve as samples for the training process, of which 90% are used for the actual training of the network and 10% for validation. Another 1500 trajectories are used for testing.

VP-STO

The artificial network for imitating the VP-STO approach without additional obstacles has 10 inputs, namely the starting and goal configuration of the five actuated joints. The output layer has $D(N-2) = 40$ elements, where D is the number of actuated joints N and the number of via-points, i. e. the output represents the via-points for each joint, excluding starting and goal configuration. In the following, nets with different numbers of layers and layer sizes are trained and compared with respect to their performance on test trajectories. The hyperparameters for training are chosen based on empirical performance. Training for 150 epochs revealed that training and validation loss only slightly decrease after 25 epochs. A batch size of 100 provides a good balance between training efficiency and convergence speed. For the activation function, Rectified Linear Unit (ReLU) and hyperbolic tangent (tanh) activation are tested, but ReLU activation shows better performance regarding convergence. Consequently, the nets are trained for 25 epochs, using a batch size of 100 and ReLU activation functions after each hidden layer. Adam is selected as optimizer as it shows a significantly lower loss in the comparison with SGD, as presented in Figure 4.4.

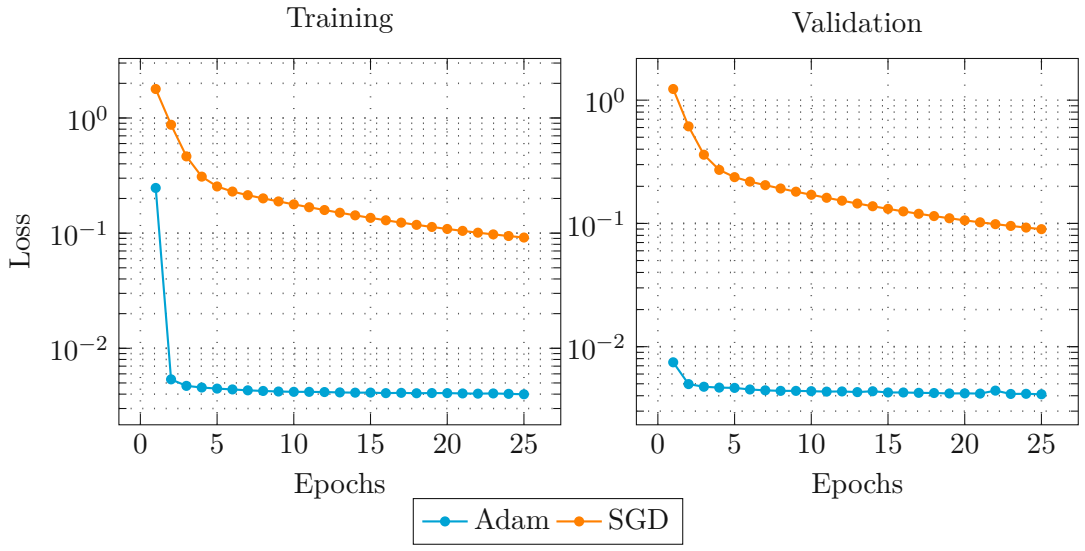


Figure 4.4: VP-STO approach without additional obstacles: Training loss and validation loss for different optimizers

In Figure 4.5, the loss progression on the training data and the validation data is presented. It can be observed that there is only little difference in the loss between the different net sizes. While larger nets generally yield lower losses, the spikes in the validation loss for the net with three layers and a layer size of 512 neurons suggest that it slightly overfits the training data.

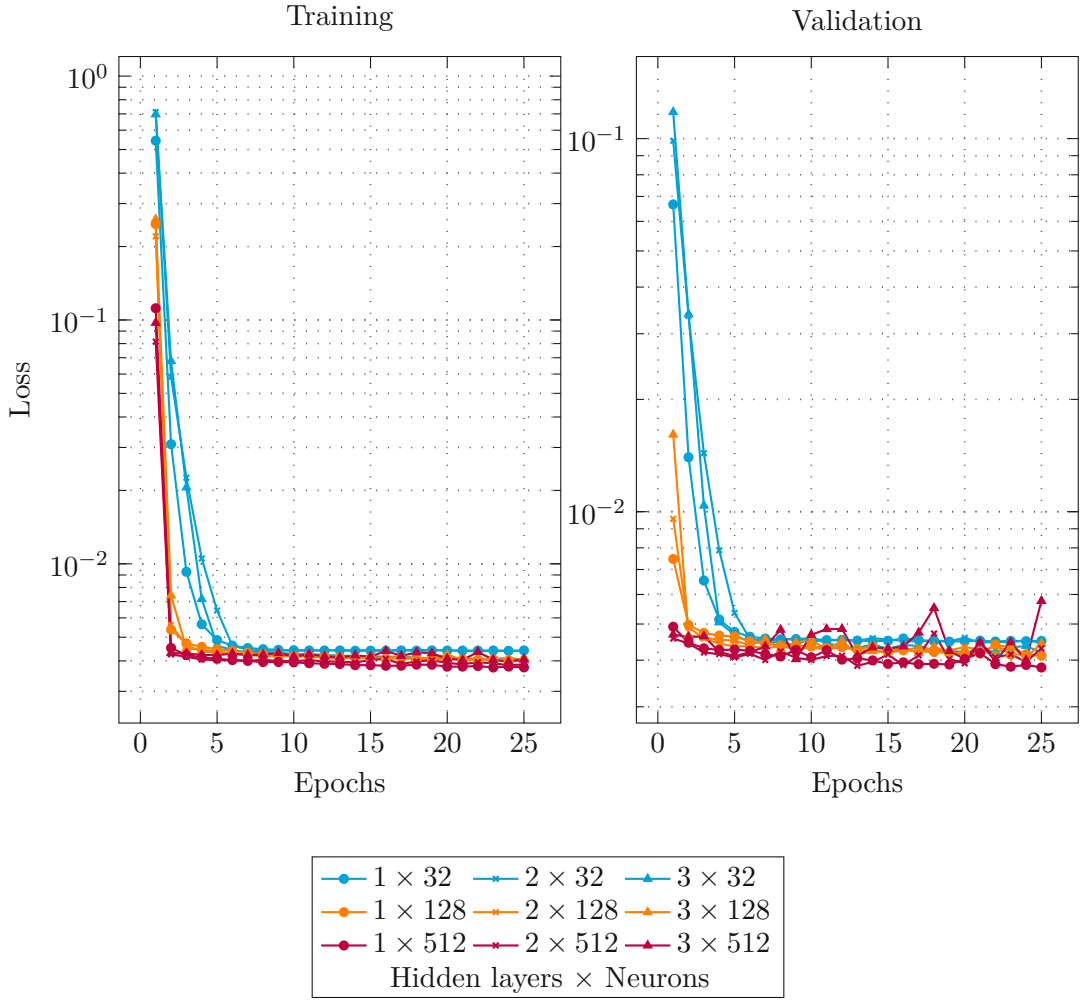


Figure 4.5: VP-STO approach without additional obstacles: Training loss and validation loss for different net sizes

To further evaluate their performance, the networks are used to generate trajectories from the test data set, which are compared regarding collisions and trajectory duration. The findings are presented in Table 4.4 and Figure 4.6. In these box plots, the vertical line in the middle marks the median value. The box comprises 50% of the data, its width is called interquartile range. The whiskers outside the box represent the expected variation of the data and extend 1.5 times the interquartile range to both sides or to the minimum or maximum value if the data points do not extend that far. All values falling

Table 4.4: VP-STO approach without additional obstacles: Collision rate for different net sizes

	1 layer	2 layers	3 layers
32 neurons	11.9%	10.1%	7.7%
128 neurons	7.5%	10.1%	8.4%
512 neurons	5%	4.7%	6.2%

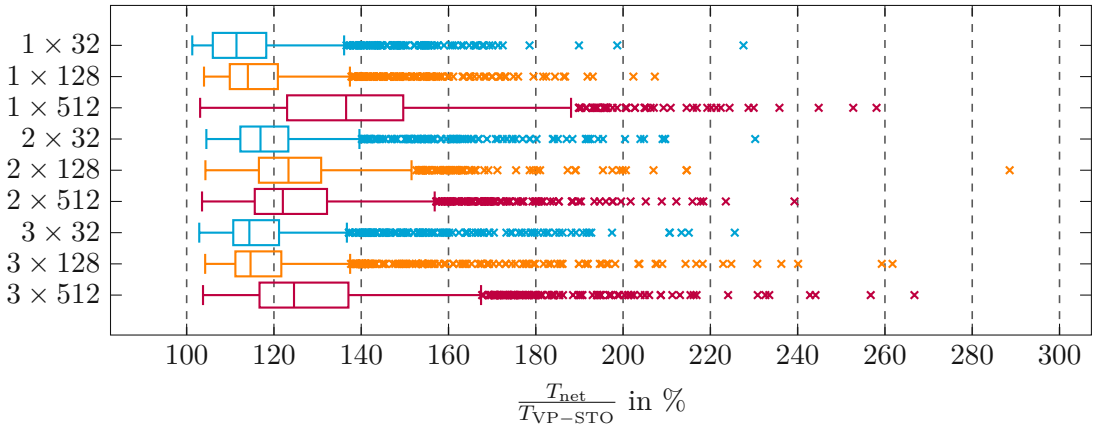
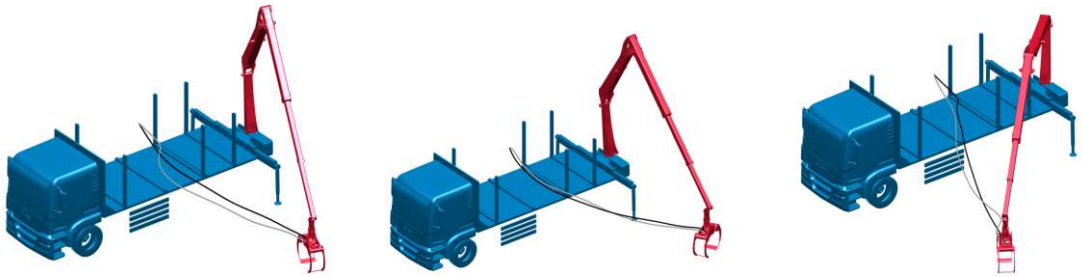


Figure 4.6: VP-STO approach without additional obstacles: Trajectory duration for collision-free trajectories generated by the net in relation to VP-STO

outside that range are called outliers and are marked with an x. Note that the networks are not specifically trained to avoid collisions or produce short trajectories. They are merely trained to imitate the VP-STO algorithm by minimizing the mean squared error between the predicted and the optimal via-points. Ideally, collision avoidance and short trajectories are learned implicitly through behavioral cloning. However, due to the choice of a mean squared error as loss function, a predicted via point can deviate above or below the optimal via point while resulting in the same loss. Consequently, despite achieving a low loss, deviations of predicted via points in unfavorable directions can lead to collisions and trajectories with greater curvature, yielding longer trajectory durations. Hence, low losses do not automatically imply short and collision-free trajectories.

In Table 4.4 a tendency towards less collisions for larger nets can be observed. Note that the 3×512 network showed signs of overfitting in the training loss, which might be the reason why the collision rate is higher than for the networks with 512 neurons and less layers. Figure 4.6 presents the generated trajectory duration with respect to VP-STO for all collision-free trajectories. It can be seen that while the trajectory duration varies significantly for all net sizes, smaller networks tend to generate shorter trajectories. Additionally, in the case of similar performance, smaller networks should be favored as they are more computationally efficient as less mathematical operations are required for each pass.

For imitating VP-STO without additional obstacles, a net size of one layer with 128 neurons represents a good trade-off between collision rate, trajectory duration and



(a) (b) (c)

Figure 4.7: VP-STO approach without additional obstacles: Trajectories generated by the 1×128 network (black) and the VP-STO algorithm (gray)

computational complexity with a collision rate of 7.5% and median trajectory duration of 114% compared to VP-STO. Three collision-free test trajectories generated by this network in comparison with VP-STO are presented in Figure 4.7 and Figure 4.8.

Furthermore, it is examined how many training samples are necessary to obtain good results. The network consisting of one layer with 128 neurons is trained with different amounts of training trajectories. From Figure 4.9, it can be observed that in general, the less training samples are available, the slower the convergence and the higher the loss to which the loss curve converges. However, the validation loss for 9000 training trajectories converges to a comparable value as for 18 000 training samples, indicating that similarly good performance can already be achieved with only half the training data.

To determine the computation time for one trajectory, the average time is calculated on the basis of 100 trajectories, each generated by the VP-STO algorithm and the network. The VP-STO algorithm takes 10.3 s on average to compute a trajectory while the network generates trajectories in 2.8 ms. Note that the VP-STO algorithm can be parallelized to enhance computation time, which has not been implemented for this runtime test. However, for c cores, computation time can be reduced to $\frac{10.3}{c}$s, i. e. it would require more than 3600 cores to achieve a speed similar to the network. The training time for this network is 17 s.

The comparison shows that for imitating the VP-STO algorithm without additional obstacles, the network with one layer and 128 neurons proves to be the best trade-off between collision rate, trajectory duration and computational efficiency. Larger networks generally yield lower losses but may suffer from overfitting and smaller networks tend to produce shorter trajectories. Furthermore, 9000 training trajectories are sufficient to obtain comparable results. Finally, the network can generate trajectories significantly faster than VP-STO, demonstrating its efficiency.

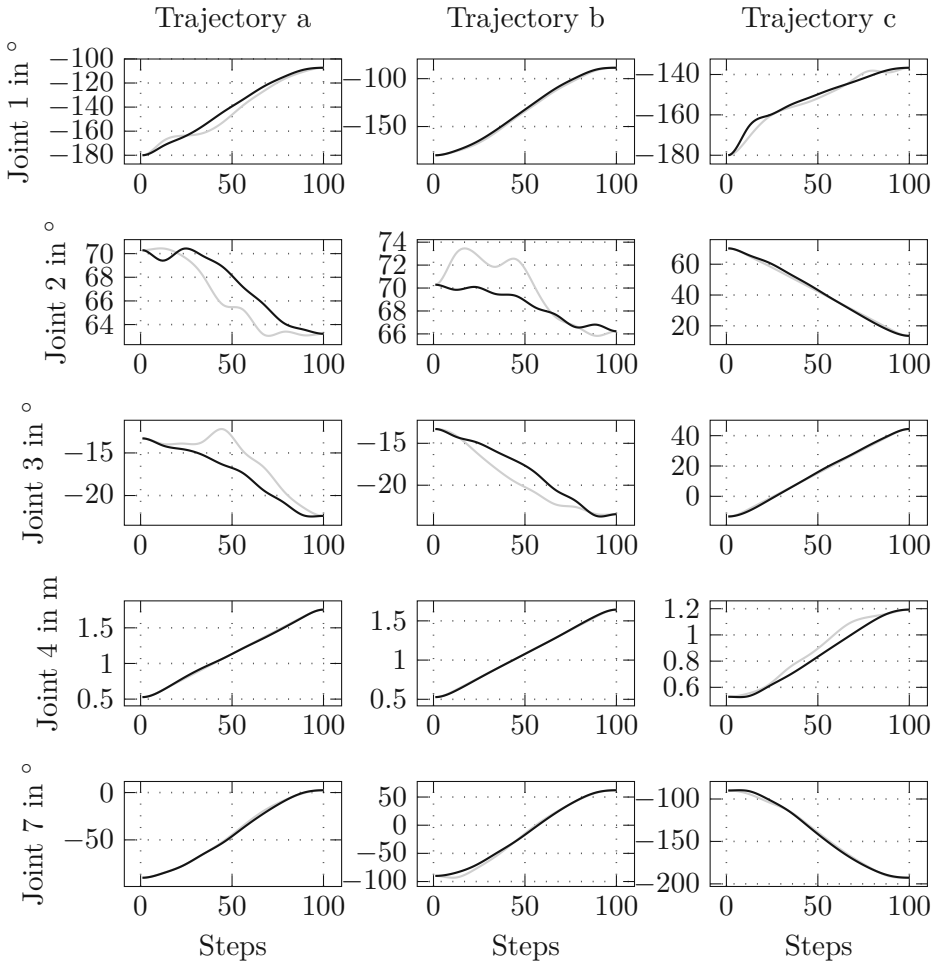


Figure 4.8: VP-STO approach without additional obstacles: Trajectories generated by the 1×128 network (black) and the VP-STO (gray) algorithm, illustrated for each joint

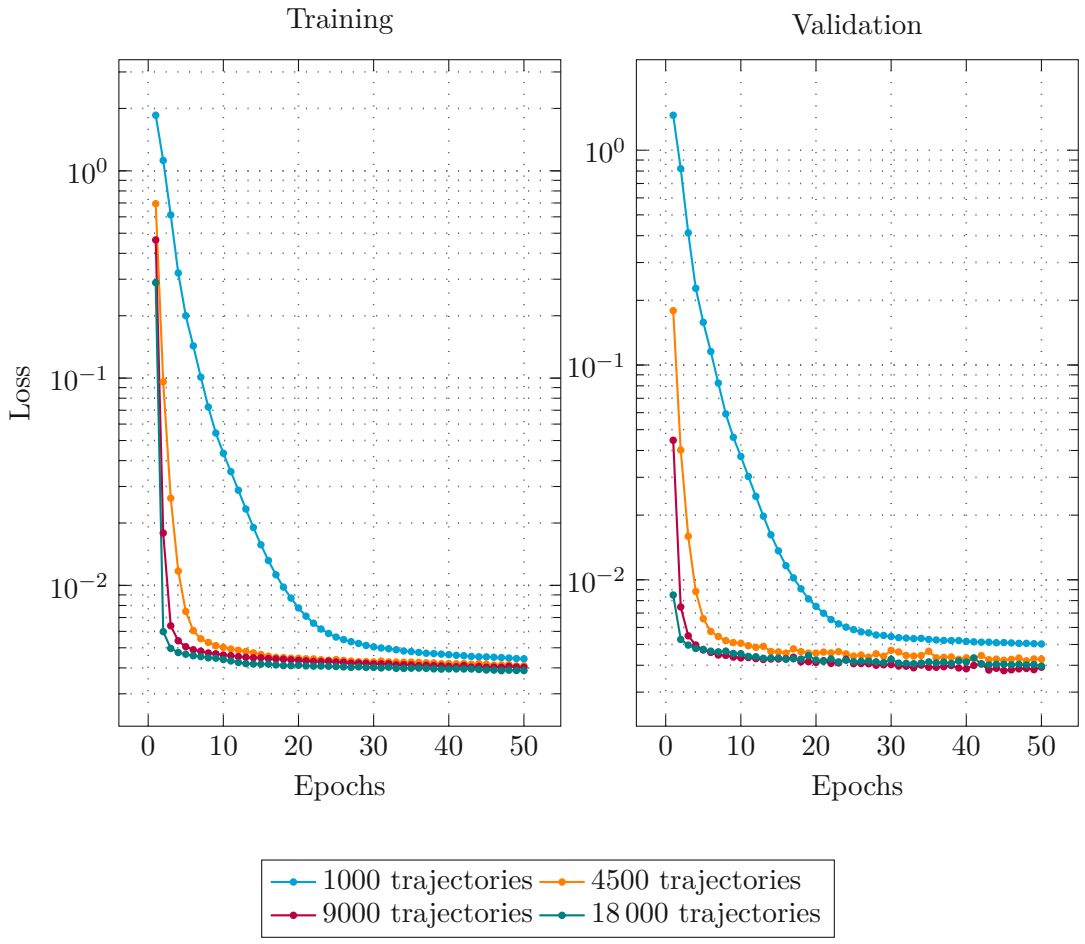


Figure 4.9: VP-STO approach without additional obstacles: Training loss and validation loss for different amounts of training data

Optimal Control Formulation

For imitating the optimal control solution without additional obstacles, the artificial network has 21 inputs: the current configuration and velocity and the goal configuration of all seven joints. The net outputs the control input, hence the output layer has five elements. To determine an adequate number of epochs, a network is trained for 150 epochs, showing that neither training nor validation loss significantly decrease after 50 epochs, making this value a reasonable choice for the following comparison. The other hyperparameters are chosen based on empirical performance as well. The networks for this scenario are trained for 50 epochs, with a batch size of 100, ReLU activation functions after each hidden layers and Adam optimizer, using trajectories computed by an iterative Linear Quadratic Regulator. Since the trajectory is obtained iteratively in this approach, each step serves as a sample. With 80 steps on average per trajectory, this results in approximately 1 440 000 samples. As each trajectory can have a different number of steps required to reach the goal configuration, the actual number of samples depends on the trajectories. In the following, networks of different sizes are trained and evaluated considering their performance on the test set, consisting of 1500 trajectories.

Similar to the VP-STO networks, the networks for imitating the solution for the optimal control formulation are not specifically trained to avoid collisions. In contrast to VP-STO, where the generated trajectories inherently comply with the crane's limitations, the optimal control networks neither receive any feedback about exceeding limitations as they are merely trained to minimize the mean squared error between the predicted and the optimal control input. Furthermore, trajectory generation with the networks on the test trajectories is terminated after the number of steps predetermined by the iLQR algorithm, potentially resulting in a goal error. The performance on the test trajectories is evaluated with respect to collision rate, compliance to limitations and goal error, the results presented in Table 4.5, Figure 4.11, Figure 4.12 and Figure 4.13.

The training and validation loss for different net sizes are shown in Figure 4.10. The networks with 32 neurons and those with one layer show a higher loss than larger networks. However, the losses are very low in general and the absolute differences between the loss curves is minimal. The one-layer networks with 32 and 512 neurons produced three and two highly infeasible trajectories from the test set, respectively, exceeding limitations by more than $10^{38}\%$. The 1×128 does not show this problem to this extent, but for some trajectories, the limits are exceeded by more than 100%. This suggests that one layer might not be sufficient to adequately imitate the solution for the optimal control formulation. The one-layer networks are therefore excluded from the following comparison.

The collision rate, presented in Table 4.5, shows no tendency to less collisions for larger layers. The 2×128 network shows the lowest collision rate with 5.5%, the 2×512 network the highest with 9.3%.

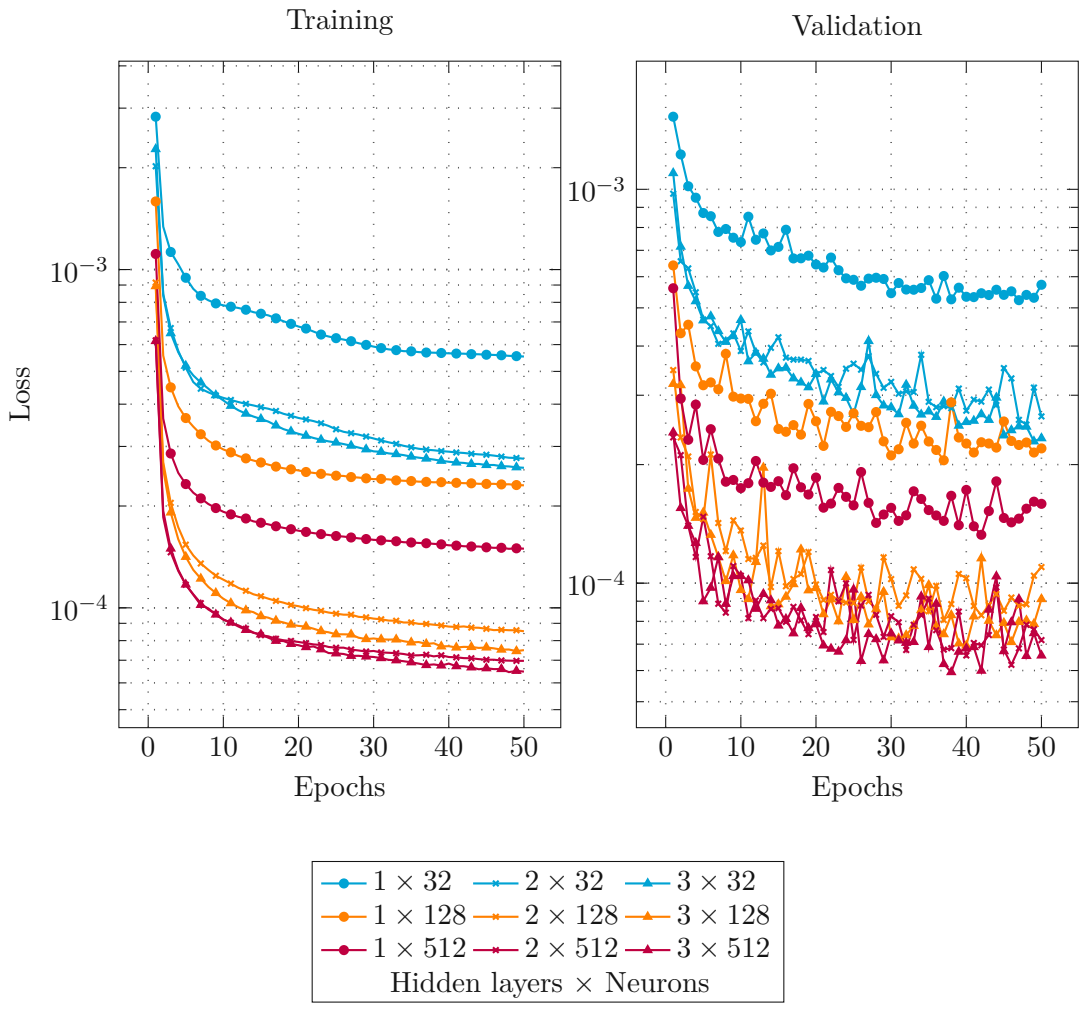


Figure 4.10: Optimal control approach without additional obstacles: Training loss and validation loss for different net sizes

Table 4.5: Optimal control approach without additional obstacles: Collision rate for different net sizes

	2 layers	3 layers
32 neurons	8.3%	7.3%
128 neurons	5.5%	8.7%
512 neurons	9.3%	7.5%

To visualize the extent to which limitations are exceeded, the difference between the actual joint value and its corresponding limit is normalized by the permissible range, while deviations in velocity and control input are normalized by their respective maximum values. Figure 4.11 presents the maximum violation of limitations. Note that this figure only visualizes values where the respective limit is actually exceeded. It can be observed that the variance in velocity limit violation is especially high for 32-neuron networks. Regarding the joint and control input limitations, the 2×512 and the 3×128 networks yield the lowest median values and the velocity limit violation stays below 20% for both

networks.

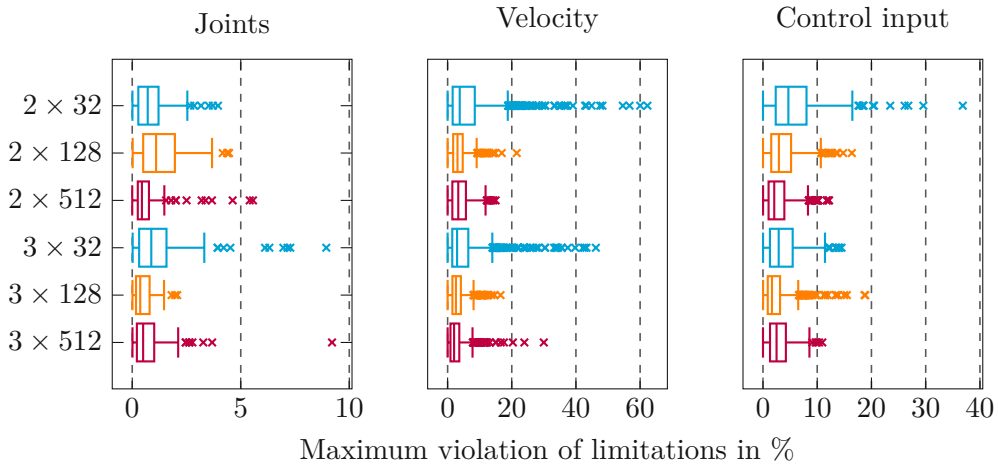


Figure 4.11: Optimal control approach without additional obstacles: Violation of joint, velocity and control input limitations

To evaluate how well the goal state is reached, the error between the gripper's end position and the desired position is examined and presented in Figure 4.12. Additionally, the remaining energy in the system is calculated as a measure for the deviation and oscillation of the gripper due to the non-actuated joints, see Figure 4.13. For this, the sum of the remaining potential and kinetic energy of the system in its end state is normalized so that a value of 0 corresponds to the equilibrium state and a value of 1 corresponds to a deflection of 90° from the equilibrium state with zero velocity, i. e. a large deviation from the desired end state. It can be observed that the 3×32 network yields the smallest position errors as well as the lowest energy values while larger network tend to generate higher deviations in the goal state. The 3×512 network shows especially high variance.

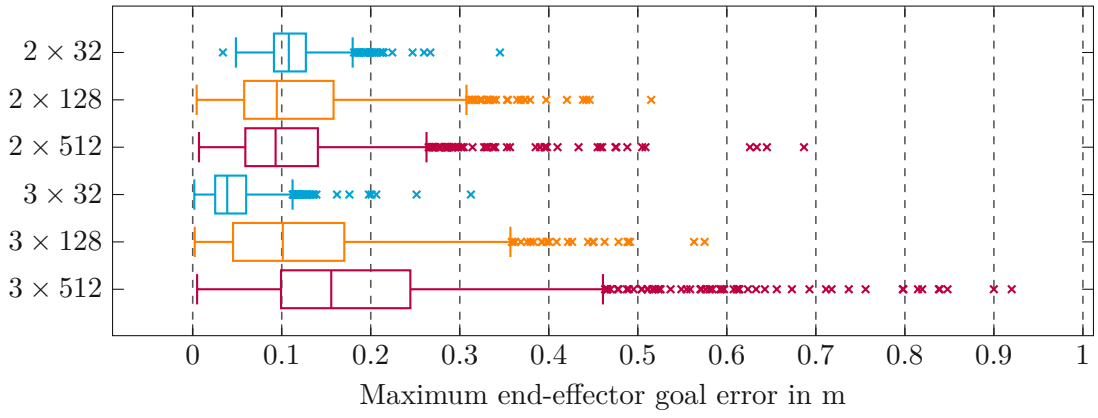


Figure 4.12: Optimal control approach without additional obstacles: End-effector goal error

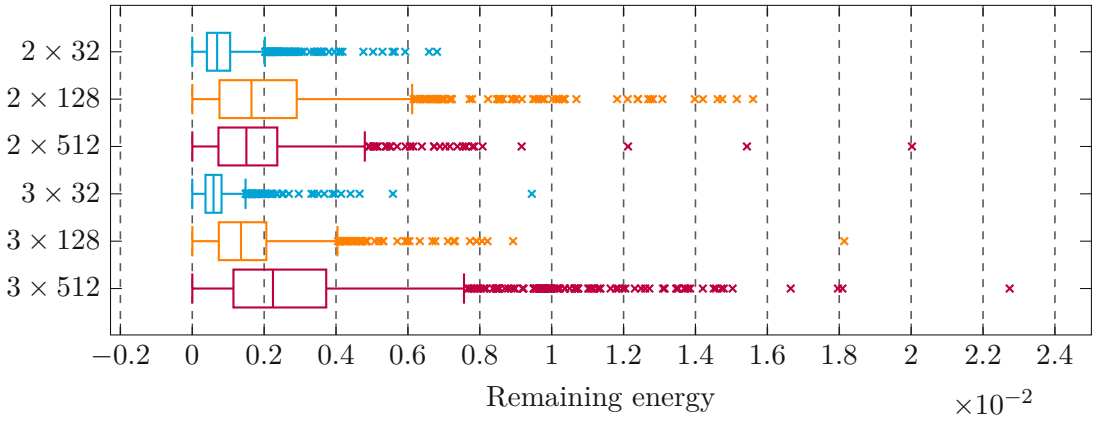


Figure 4.13: Optimal control approach without additional obstacles: Normalized remaining energy in the non-actuated system in the end state

Although the 3×32 network reaches the goal configuration more precisely, it shows poor performance regarding limitation compliance, yielding up to 46% violation of velocity limitations. The network with three layers and 128 neurons poses a good trade-off between collision rate, compliance to limitations and goal error. It shows a collision rate of 7.5% and median limit violations of 0.4% for joint limits, 2.6% for velocity limits and 1.7% for control input limits. The median deviation from the goal position is 10.1 cm. To further evaluate how each joint contributes to the goal state error, the goal errors for each joint are calculated. The results are presented in Figure 4.14. Note that the errors for joint q_4 are specified in cm or cm/s, whereas for the other joints the errors are specified in $^\circ$ or $^\circ/s$. It can be observed that the joints q_5 and q_6 contribute most to the goal velocity error, suggesting a remaining oscillation at the end of some trajectories due to the two non-actuated joints. However, the median values are low with 0.3°/s and 0.1°/s, respectively. Regarding the joint error, q_2 and q_5 show the highest deviations from their desired position. Three trajectories generated by the network and the iLQR algorithm are visualized in Figure 4.15 and Figure 4.16.

The iLQR algorithm takes 2.22 s on average to compute a solution for the optimal control problem without additional obstacles, whereas generating a trajectory with the net takes 0.23 s. Note that generating a trajectory with this approach involves repeatedly calling the network, hence trajectory generation takes longer than with the VP-STO network. Computing a single step with the network takes 2.1 ms on average. All values are based on measurements of 100 trajectories. Training the 3×128 network takes approximately 54 min.

Summing up, it is difficult to observe a tendency regarding performance of the different networks. Although the training and validation losses are lower for larger networks, the differences are minimal. One-layer networks prove to be inadequate for this scenario, generating highly unfeasible trajectories. 32-neuron layer networks, on the other hand, perform best considering goal errors but do not comply well with system limitations. Hence, for imitating the optimal control solution without additional obstacles, the 3×128 network proves to be a good fit regarding collision rate, compliance to joint limits and

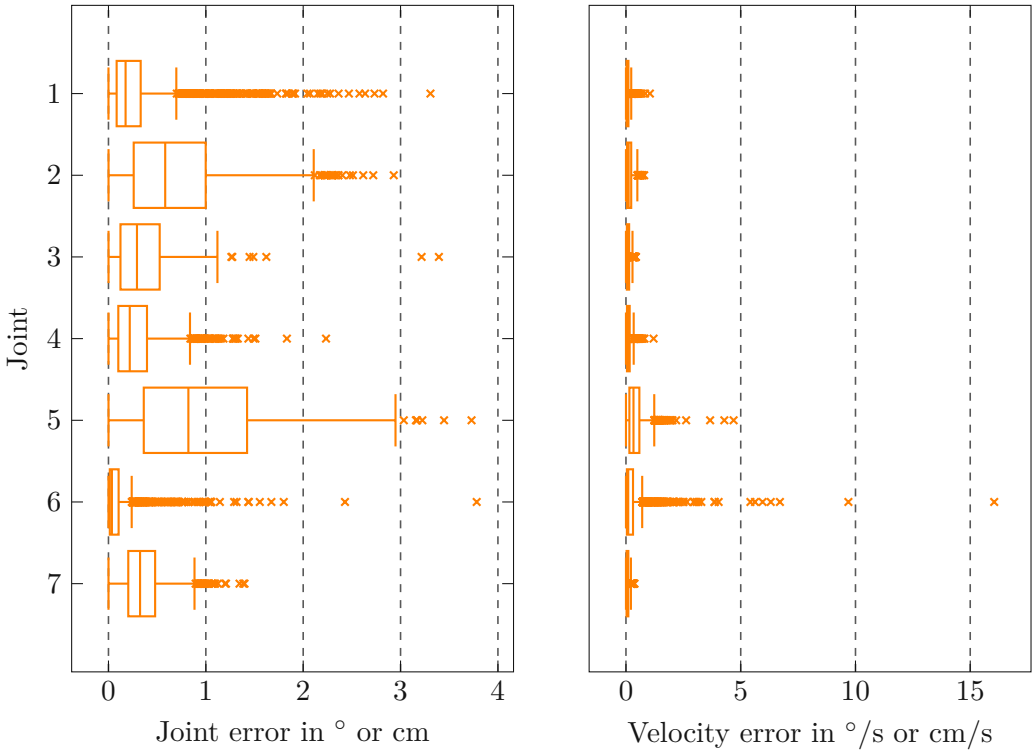
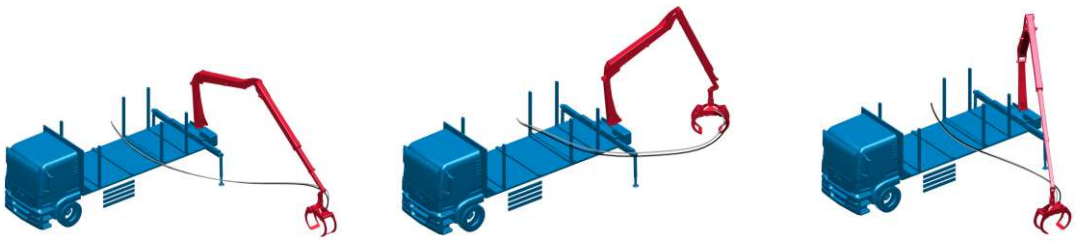


Figure 4.14: Optimal control approach without additional obstacles: Goal state error of all joints for the 3×128 network

precision. The need for larger networks in comparison with VP-STO in Section 4.2.1 indicates that the task of imitating the optimal control solution is more complex. This and the fact that the trajectory is obtained iteratively in this case, requiring more network calls than for the VP-STO approach, lead to a higher computation time. However, the network still proves to be significantly faster than the iLQR algorithm.



(a) (b) (c)

Figure 4.15: Optimal control approach without additional obstacles: Trajectories generated by the 3×128 network (black) and the iLQR algorithm (gray)

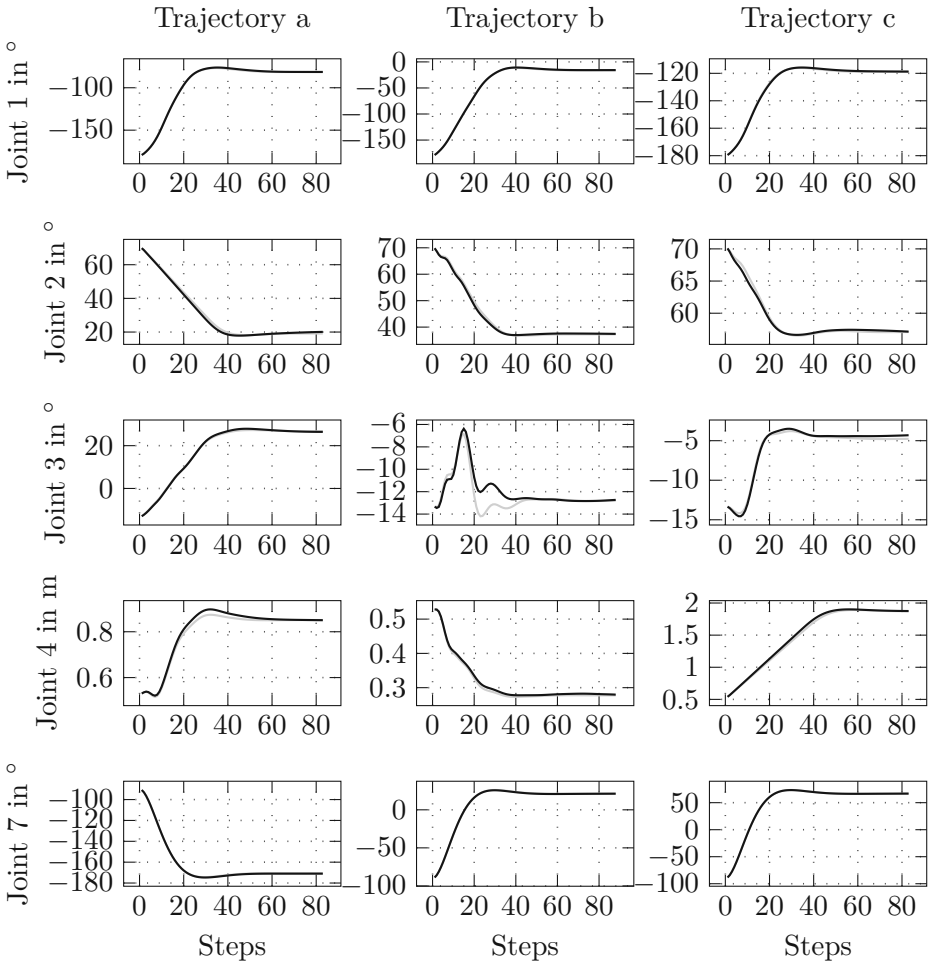


Figure 4.16: Optimal control approach without additional obstacles: Trajectories generated by the 3×128 network (black) and the iLQR algorithm (gray), illustrated for each joint

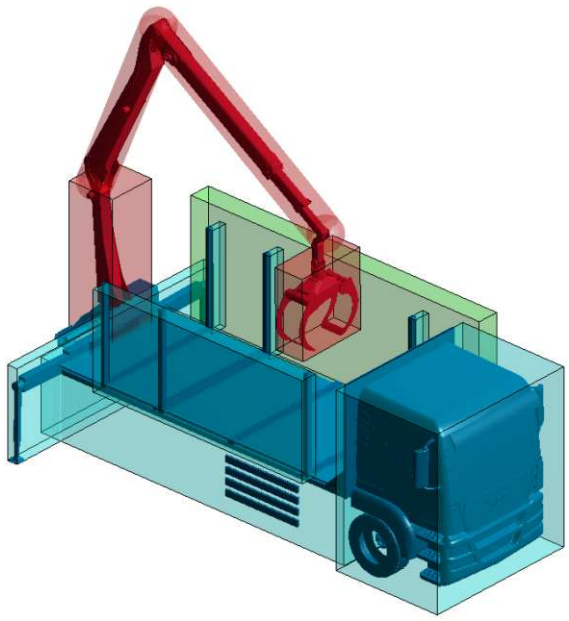


Figure 4.17: Collision model for the timber crane with one additional obstacle

4.2.2 One Additional Obstacle: Variable Height

In the next scenario, an additional obstacle in the form of a wall beside the truck is added, as shown in Figure 4.17. In contrast to the static truck, whose presence is learned implicitly, the wall is variable in height, while its position and other dimensions remain constant. The obstacle height serves as an additional input to the artificial neural network. As this makes the motion planning task more complex for the network, more training data is generated. For both VP-STO and the optimal control approach, 58 000 trajectories with 6 different wall heights are used for training and validation in a 90/10 split and 1500 are used for testing.

VP-STO

As in Section 4.2.1, the network for imitating VP-STO has 40 output elements, is trained with a batch size of 100, ReLU activation functions and Adam as optimizer. The input layer is extended by one element to provide the network with information about the obstacle height. A training run for 150 epochs demonstrated a noticeable decrease in loss between 25 and 50 epochs for this scenario, therefore the networks in the following comparison are trained for 50 epochs instead of 25 as before.

From Figure 4.18 it can be seen that all networks with layer sizes of 32 neurons yield higher losses than the other networks, suggesting that they are too small to sufficiently represent the policy for the underlying task.

In comparison with the scenario without obstacles from Section 4.2.1, the collision rate, presented in Table 4.6, is significantly higher in general. To further evaluate the severity of the collisions, the collision depth of the gripper into the wall is examined and presented in Figure 4.19, as this case accounts for a large proportion of collisions.

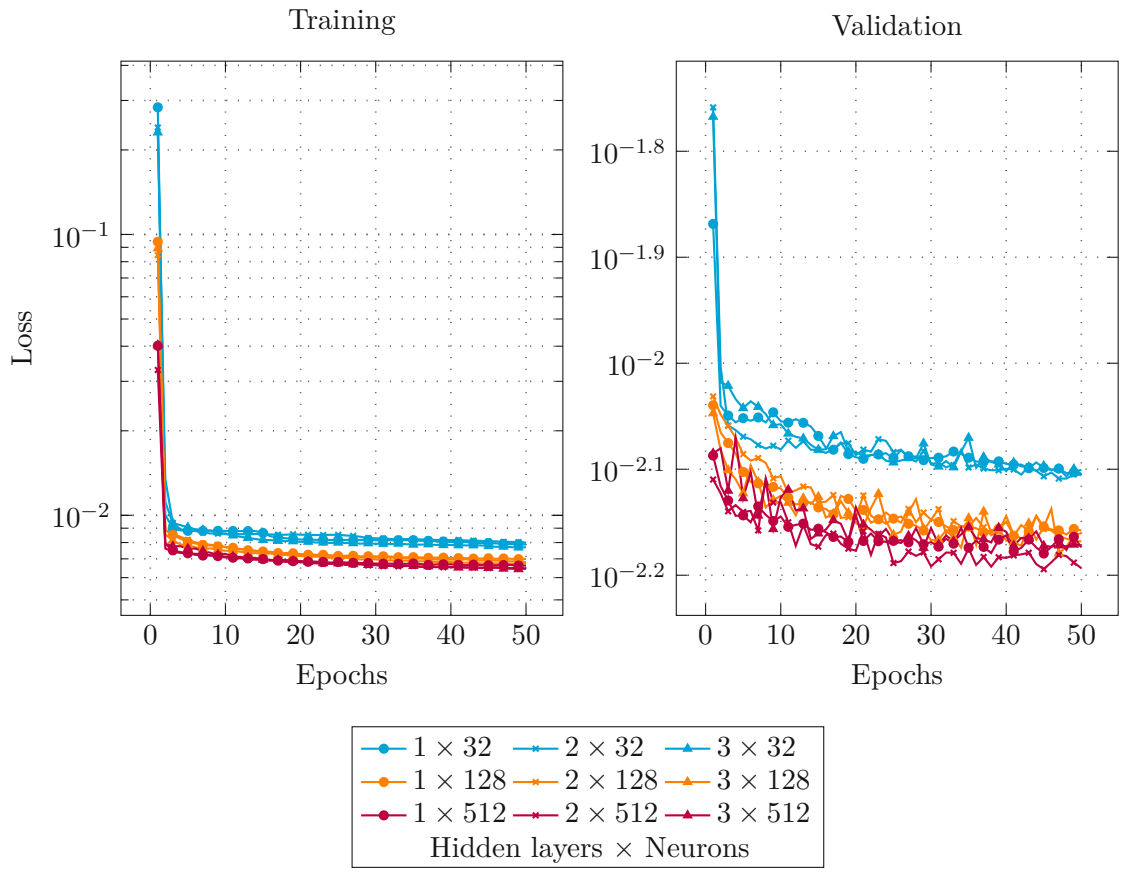


Figure 4.18: VP-STO approach with one additional obstacle: Training loss and validation loss for different net sizes

Table 4.6: VP-STO approach with one additional obstacle: Collision rate for different net sizes

	1 layer	2 layers	3 layers
32 neurons	33.9%	28.5%	26.8%
128 neurons	18.7%	18.1%	38.1%
512 neurons	25.5%	18.4%	23.9%

It can be observed that the collision rate as well as the collision depth in the case of an collision is highest for the 32-neuron networks, supporting the assumption that they are not suitable for this scenario. The two-layer networks with 128 and 512 neurons demonstrate the most favorable collision behavior, with a collision rate of 18.1% and a median depth of 5.4 cm for the 2×128 network and a 18.4% collision rate and 6.9 cm median depth for the 2×512 network.

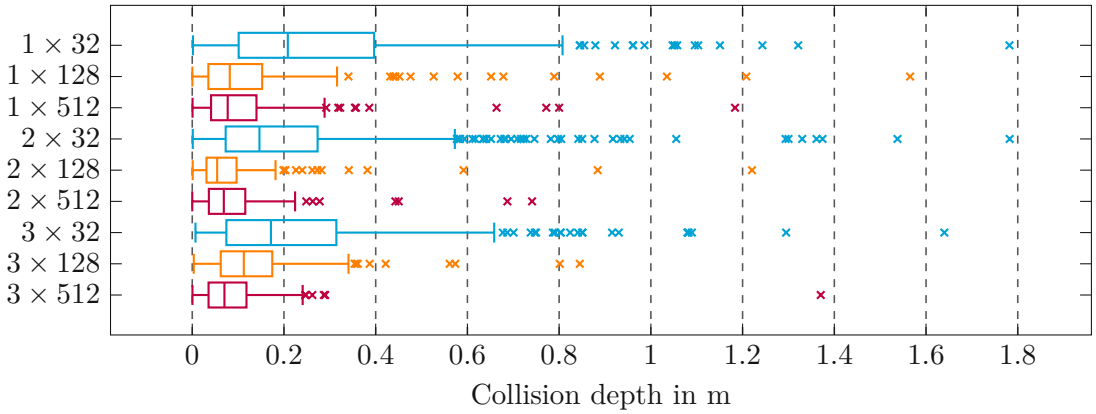


Figure 4.19: VP-STO approach with one additional obstacle: Collision depth in the case of a collision with the wall

In Figure 4.20, the trajectory duration for collision-free trajectories in comparison to VP-STO are presented. It shows that the different networks perform similarly with respect to trajectory duration, with both the 2×128 and the 2×512 network yielding the lowest median of 111.9% of the trajectory duration of VP-STO. An interesting thing can be observed in the plot for the 3×512 network. It seems to generate shorter collision-free trajectories than VP-STO, although the latter is supposed to minimize trajectory duration. On closer examination, it is only one trajectory with a duration of 97.4% compared to VP-STO. This can happen if the VP-STO algorithm finds a collision-free trajectory when the standard deviation is already very low, i. e. there is not much room to modify the trajectory to further minimize duration. A solution to this problem could be to restart the algorithm in these cases, with the obtained trajectory as initialization and a high standard deviation. However, this demonstrates that the network learns to generalize well from the training data.

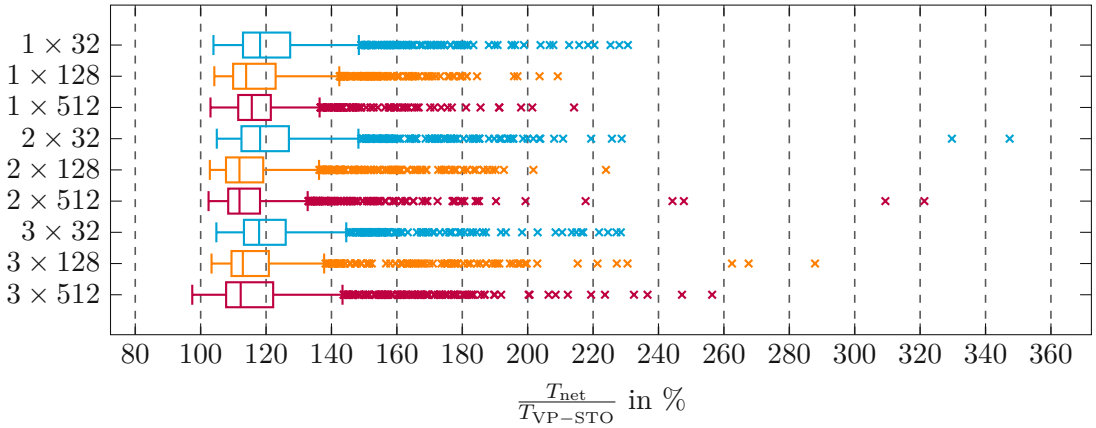
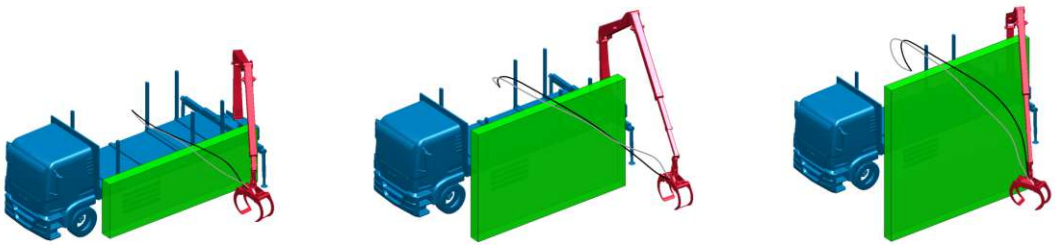


Figure 4.20: VP-STO approach with one additional obstacle: Trajectory duration for collision-free trajectories generated by the net in relation to VP-STO

With the 2-layer-128-neuron network being more computationally efficient than its 512-neuron counterpart, it presents the most suitable choice for imitating the VP-STO algorithm with one obstacle of variable height. Three trajectories generated with this network for different wall heights can be seen in Figure 4.21 and Figure 4.22.



(a) (b) (c)

Figure 4.21: VP-STO approach with one additional obstacle: Trajectories generated by the 2×128 network (black) and the VP-STO algorithm (gray) for different wall heights

The average computation time for VP-STO with one additional obstacle is 14.49 s, whereas the network generates trajectories in 4.3 ms on average, based on measurements of 100 trajectories. It takes about 80 s to train the network.

Summarizing, the comparison shows that adding an obstacle with variable height poses a significantly more complex motion planning problem for the network, reflected in higher collision rates and the need for larger networks. However, it can be observed that the network is indeed able to adapt its trajectories to the wall's height, as visualized in Figure 4.21. The best performing network is the 2×128 network, combining the lowest collision rate and collision depth with the shortest median trajectory duration among the tested networks. Furthermore, it can be observed that the 3×128 is even able to generate shorter collision-free trajectories than the VP-STO algorithm, proving that neural networks can compensate for possible difficulties of VP-STO.

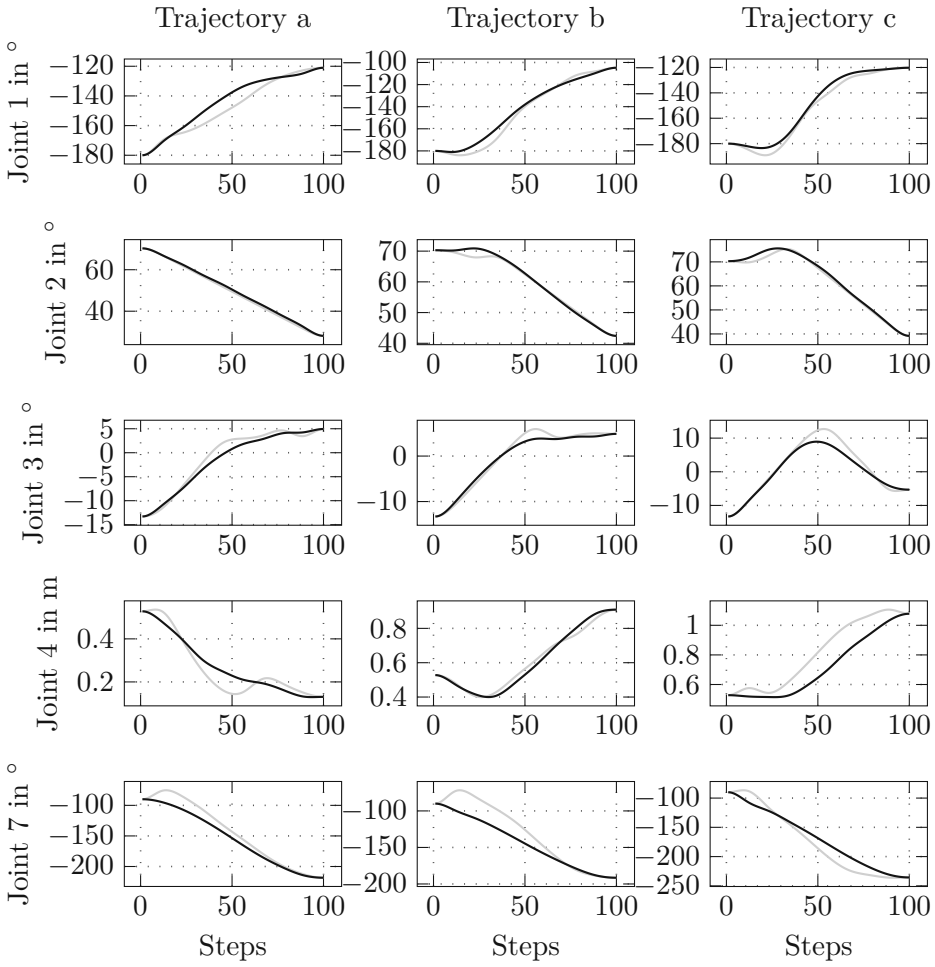


Figure 4.22: VP-STO approach with one additional obstacle: Trajectories generated by the 2×128 network (black) and the VP-STO algorithm (gray) for different wall heights, illustrated for each joint

Optimal Control Formulation

The network used for imitating the optimal control solution without additional obstacles from Section 4.2.1 is extended by one input for the variable wall height, yielding 22 inputs. The output stays the same, representing the control input for the five actuated joints. The hyperparameters for training are also adopted as they provide a good balance between convergence speed and training efficiency in this scenario as well, hence the networks are trained for 50 epochs, with a batch size of 100, ReLU activation functions and Adam optimizer. With 80 steps per trajectory and 58 000 training trajectories, approximately 4 640 000 samples are used for training. Due to the large number of samples, training the optimal control networks takes significantly longer than training the VP-STO networks. In the following, different net sizes are compared regarding their performance on 1500 test trajectories. The training and validation losses are presented in Figure 4.23.

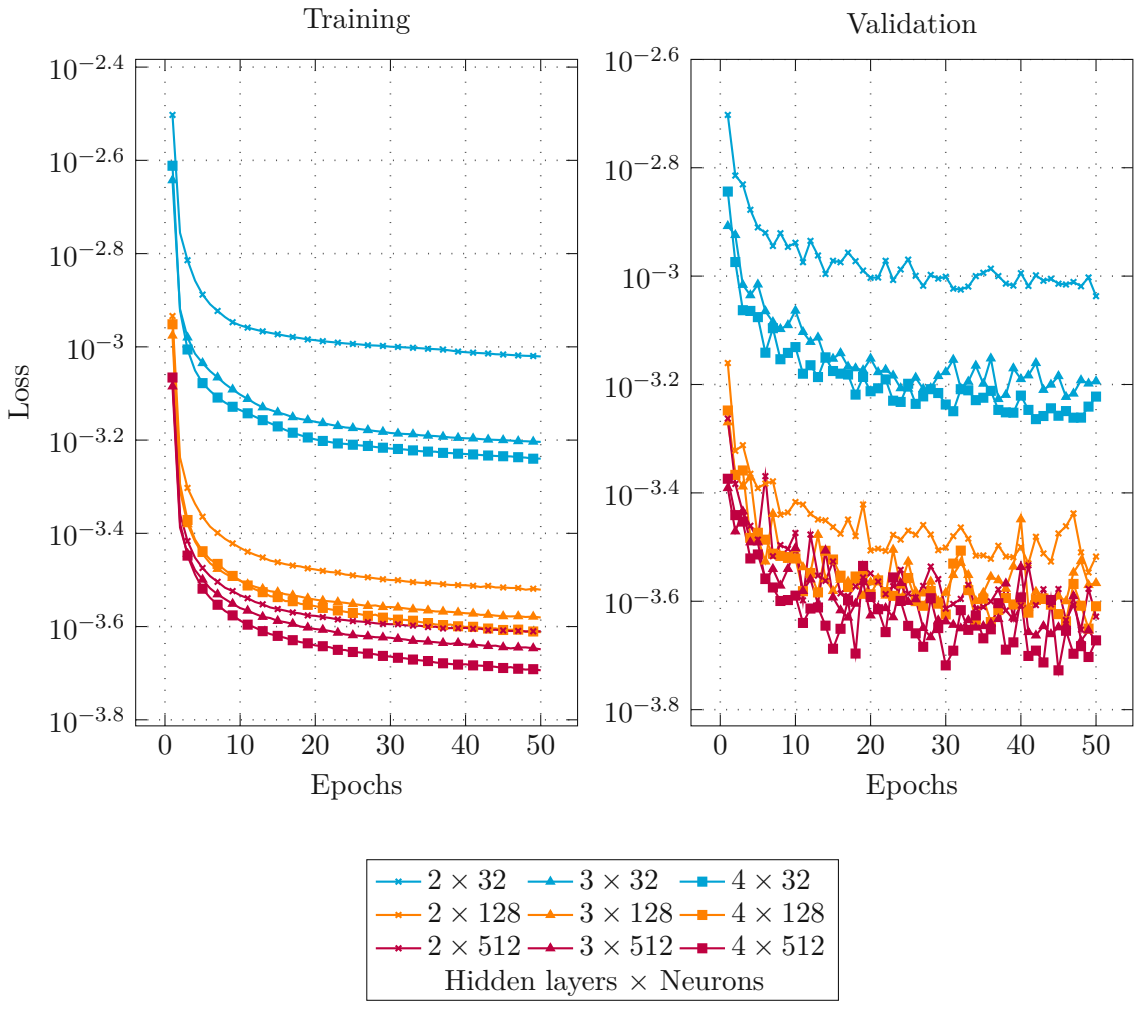


Figure 4.23: Optimal control approach with one additional obstacle: Training loss and validation loss for different net sizes

As motion planning with one variable height obstacle poses a more complex task than without obstacles, only larger networks are considered for this scenario. It can clearly be seen that the 32-neuron networks yield higher losses than larger networks. The 2×128 generates several highly unfeasible trajectories, exceeding limitations by more than 10^{38}%. Furthermore, the 2×32 and 3×32 network show more than 200% limitation violation for some test trajectories, making them unsuitable for this scenario. Therefore, these networks are excluded from the evaluation.

In Table 4.7, the collision rate for the remaining networks is presented and in Figure 4.24, the collision distance in the case of a collision of the gripper with the wall is shown. Regarding the collision depth, the networks show similar results, while the 4×512 network yields the lowest collision rate of 22.1%.

Table 4.7: Optimal control approach with one additional obstacle: Collision rate for different net sizes

	2 layers	3 layers	4 layers
32 neurons			29.5%
128 neurons		33.4%	34.8%
512 neurons	32.5%	24.1%	22.1%

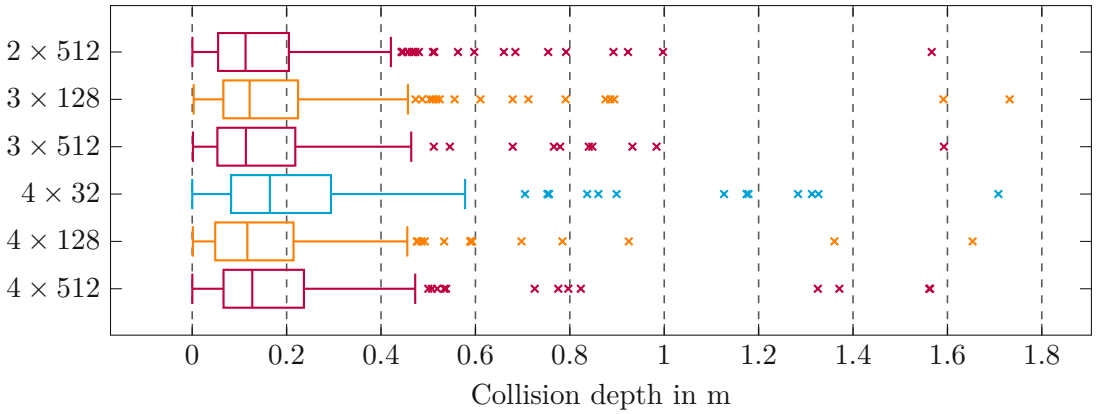


Figure 4.24: Optimal control approach with one additional obstacle: Collision depth in the case of a collision with the wall

Figure 4.25, Figure 4.26 and Figure 4.27 show the violation of limitations, the distance between the end position and the actual goal position of the gripper and the normalized remaining energy in the system in the end state as a measure for the deviation and oscillation of the gripper due to the non-actuated joints, respectively. It can be seen that the 32-neuron network neither complies well with limitations nor shows good performance regarding the goal error. The other networks show similar performance, with the 3×512 network having the lowest median values for joint, velocity and control input violation. It also demonstrates low end-effector goal errors and low energy values for the non-actuated joints in the end state.

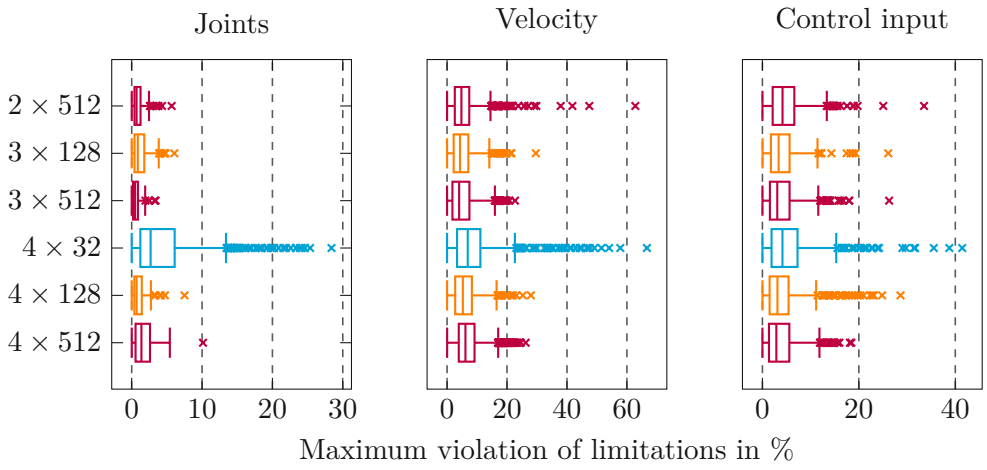


Figure 4.25: Optimal control approach with one additional obstacle: Violation of joint, velocity and control input limitations

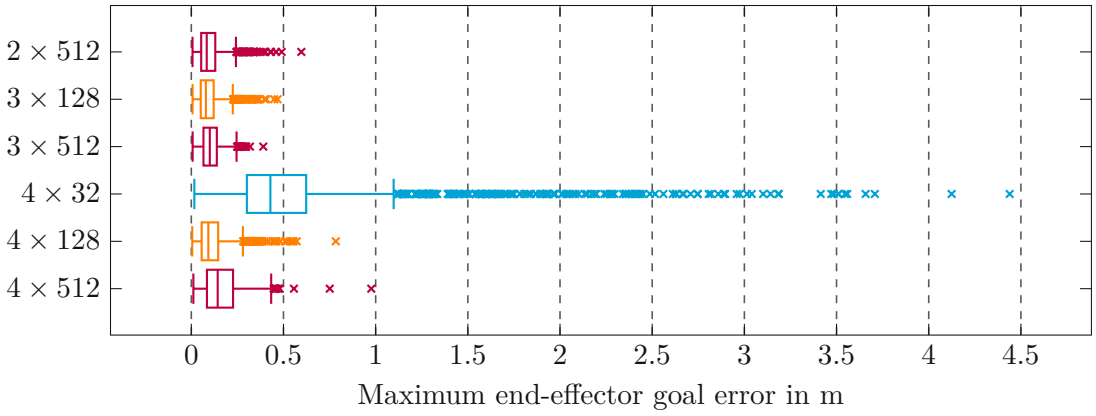


Figure 4.26: Optimal control approach with one additional obstacle: End-effector goal error

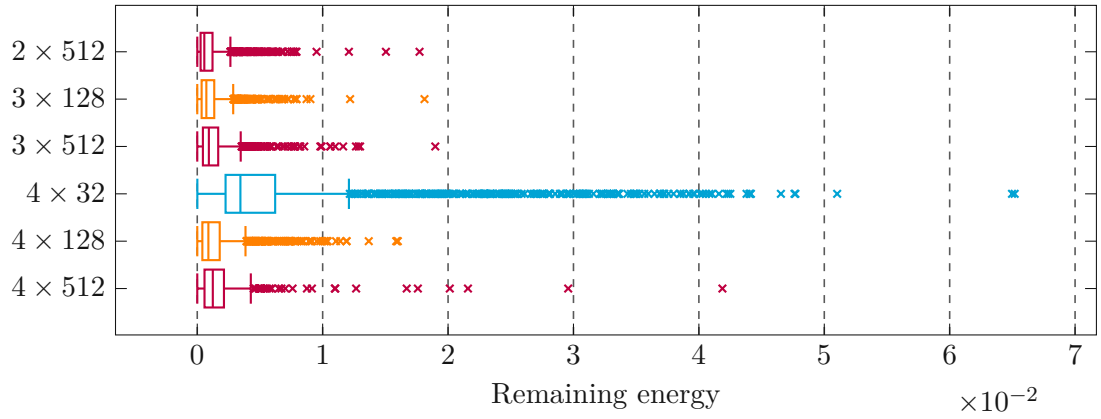
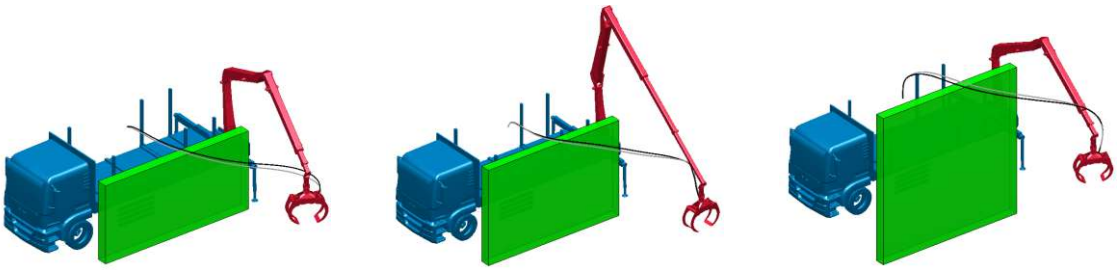


Figure 4.27: Optimal control approach with one additional obstacle: Normalized remaining energy in the non-actuated system in the end state

Therefore, the 3×512 can be regarded as a good choice for imitating the optimal control solution with one additional obstacle of variable height. It has a collision rate of 24.1%, a median collision depth of 11.6 cm, median violation values of 0.5% for joint limits, 4% for velocity limits and 3.1% for control input limits and a median goal error of 10 cm. Three trajectories generated by the iLQR algorithm and the network are visualized in Figure 4.28 and Figure 4.29.



(a) (b) (c)

Figure 4.28: Optimal control approach with one additional obstacle: Trajectories generated by the 3×512 network (black) and the iLQR algorithm (gray) for different wall heights

For further examination of how each joint contributes to the goal error, Figure 4.30 presents the error broken down by joint. Note that the error for q_4 is specified in cm, the others in degree. It can be observed that the two non-actuated joints yield the largest velocity errors with a median value of 0.51°/s for q_5 and 0.56°/s for q_6, causing small oscillations in the end configuration for some trajectories.

Based on measurements of 100 trajectories the iLQR algorithm takes approximately 2.05 s to generate one trajectory, the network 0.69 s. Computing one step with the network takes 7.7 ms on average. Training time amounts to approximately 17 h 46 min.

To evaluate the performance of the iLQR algorithm and the neural network in the case of varying parameters, both are applied to a system that differs slightly from the

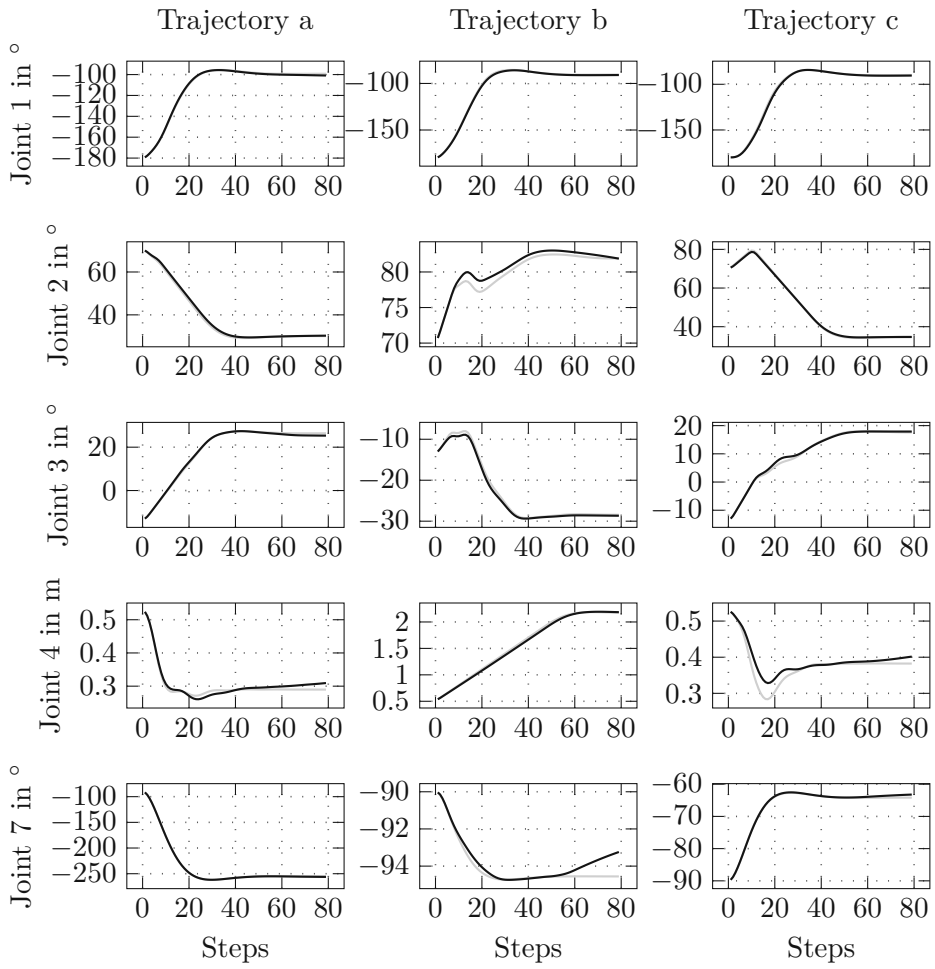


Figure 4.29: Optimal control approach with one additional obstacle: Trajectories generated by the 3×512 network (black) and the iLQR algorithm (gray) for different wall heights, illustrated for each joint

one the iLQR algorithm is familiar with and the network is trained on. In this way, the robustness and adaptability of each method can be assessed, providing insights into their generalization capabilities. In the following, the gripper's mass is adapted to 2.5 times the original mass and the center of gravity is shifted by 16 cm to simulate parameter variations. Both approaches are evaluated using the test data set. The iLQR's performance is evaluated based on whether a solution is found despite the varying parameters. If the algorithm has not converged after 90 steps, trajectory generation is considered unsuccessful. The network is called iteratively to generate a trajectory and is also stopped after 90 steps if the goal state is not reached before.

The iLQR algorithm successfully finds a solution for 945 out of 1500 trajectories, yielding a success rate of 63%. None of the network-generated trajectories reach the goal state within 90 steps, therefore the collision behavior, limitation violations and the goal error

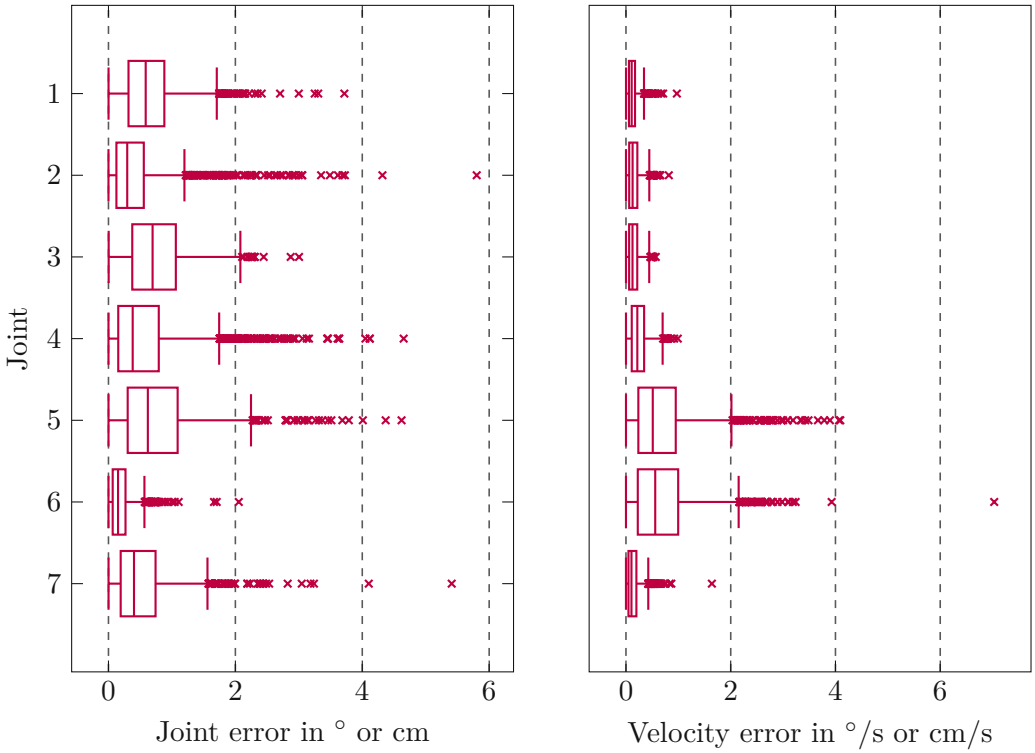


Figure 4.30: Optimal control approach with one additional obstacle: Goal state error of all joints for the 3×512 network

are examined, which are presented in Figure 4.31, Figure 4.32, Figure 4.33 and Figure 4.34. Although the network yields a collision rate of only 11.9%, which is significantly lower than the 24.1% without parameter variations, the median collision depth is considerably higher with 17.7 cm, in contrast to 11.6 cm before. Furthermore, the median goal error increases from 10 cm to 18.3 cm and the median violation values are 0.9% for joint limits, 7.4% for velocity limits and 3.9% for control input limitations, which are also higher than without parameter variations.

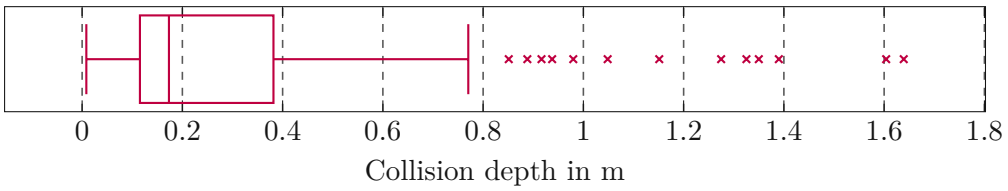


Figure 4.31: Optimal control approach with one additional obstacle: Collision depth in the case of a collision with the wall for different parameters than used for training

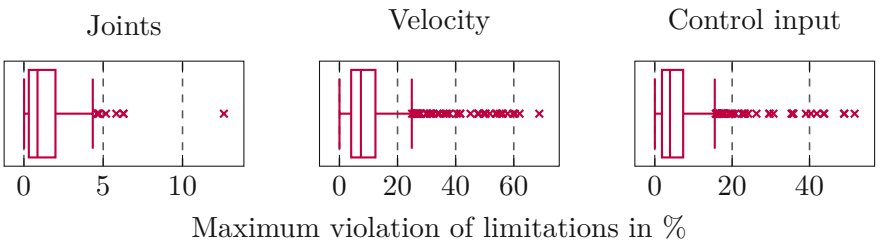


Figure 4.32: Optimal control approach with one additional obstacle: Violation of joint, velocity and control input limitations for different parameters than used for training

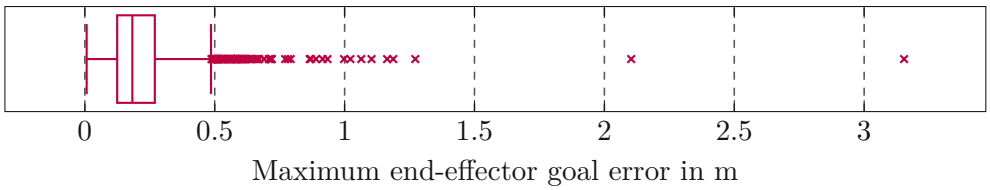


Figure 4.33: Optimal control approach with one additional obstacle: End-effector goal error for different parameters than used for training

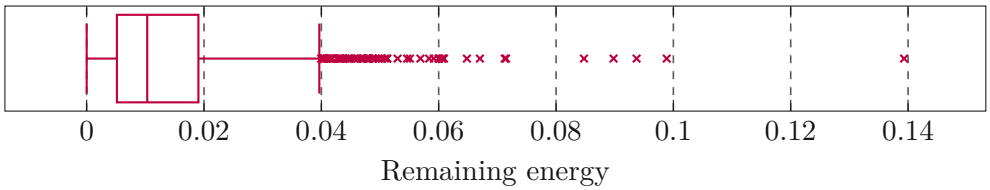


Figure 4.34: Optimal control approach with one additional obstacle: Normalized remaining energy in the non-actuated system in the end state for different parameters than used for training

The evaluation shows that smaller networks for imitating the optimal control solution with one variable height obstacle generate unfeasible trajectories, exceeding limits by 200% and higher. Overall, the network with three layers and 512 neurons demonstrated the best performance for this scenario, showing that the network is able to adapt to varying wall heights. Concerning the robustness with respect to varying parameters, the iLQR algorithm yields a success rate of 63% while the network shows a significantly higher median collision depth and median goal error.

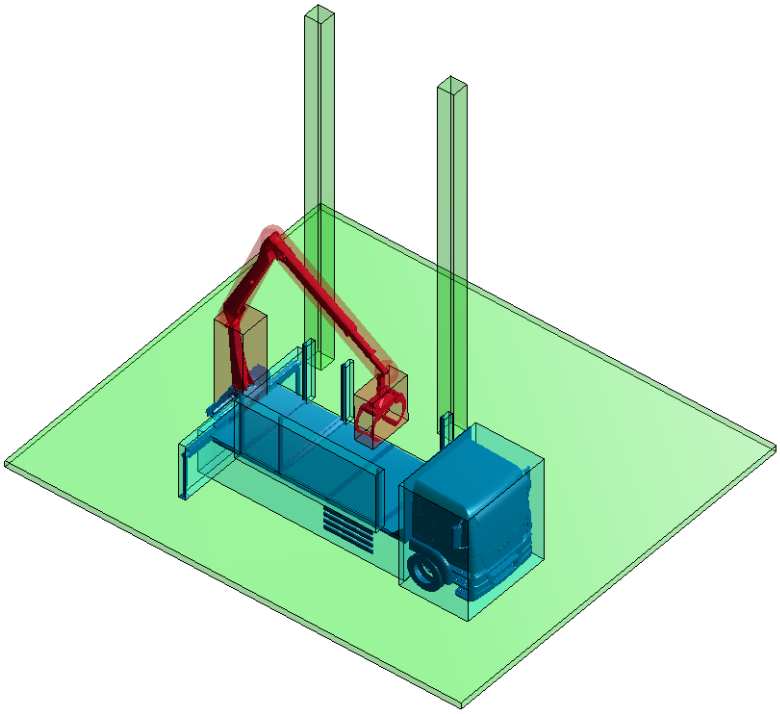


Figure 4.35: Collision model for the timber crane with two additional obstacles

4.2.3 Two Additional Obstacles: Variable Position

For the last scenario, two obstacles as illustrated in Figure 4.35 are added beside the truck, representing trees. Their size is fixed, only their position along the ground varies, yielding four additional input elements to the networks, two for each tree. Additionally, the ground is added as an obstacle to prevent the algorithms from generating trajectories below the trees. The networks are trained with 140 different tree configurations and 1000 trajectories each. As the trajectories are invariant to the order in which the trees' positions are used as the input for the network, training data can be doubled by utilizing the same trajectories for both combinations of the trees' positions. This yields a total of 280 000 trajectories, which are used for training and validation in a 90/10 split. For testing, another 10 tree configurations with 150 trajectories each are used, yielding 1500 test trajectories.

As this scenario poses a very complex motion planning scenario, it is difficult to obtain solutions for the optimal control formulation with the iLQR algorithm as it has problems converging. For this reason, the following analysis will be carried out only on the network for imitating the VP-STO algorithm.

VP-STO

The VP-STO network has 14 input elements, four for the two trees' positions and ten for start and goal configuration of the five actuated joints. Training one network for 150 epochs shows that although training loss keeps decreasing after 50 epochs, validation loss stagnates. To avoid overfitting, training for the following comparison is therefore stopped after 50 epochs. Furthermore, a batch size of 100 trajectories and ReLU activation functions after each hidden layer are adopted from the other scenarios as they showed good performance regarding convergence and training efficiency. As the scenario with two movable obstacles constitutes a complex motion planning task, larger networks are considered.

Networks with 128, 512 and 1024 neurons with two, three, four and five layers each are considered for the following comparison. As some of the loss curves for the different net sizes are very similar, only six of the twelve curves are shown as representatives in Figure 4.36 for the sake of clarity.

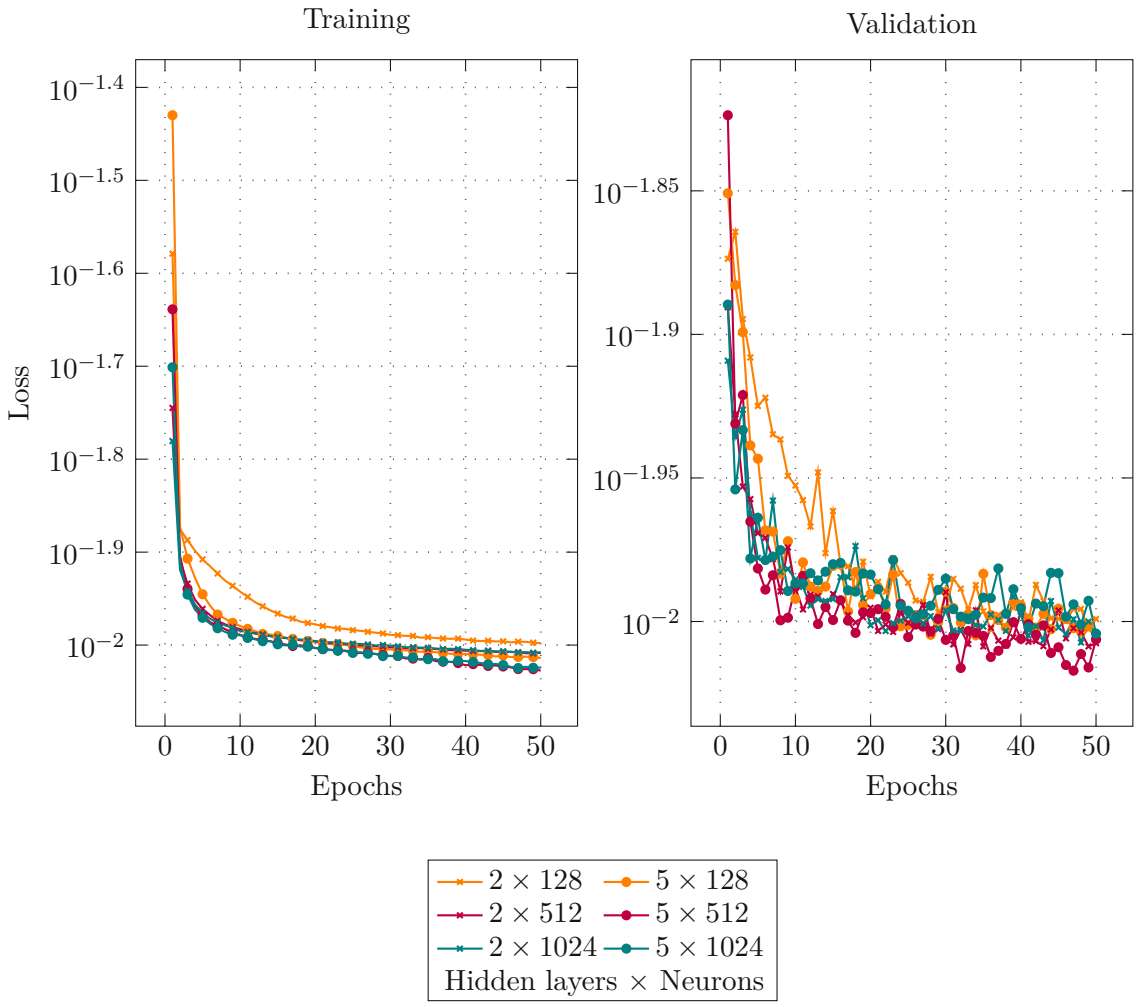


Figure 4.36: VP-STO approach with two additional obstacles: Training loss and validation loss for different net sizes

Table 4.8 and Figure 4.37 present the collision behavior of the networks on the test set. Figure 4.37 shows the overlapping volume of the two bodies involved in the collision as a measure for the severity of the impact. The high collision rates and the high variance in the collision volume suggest that the networks do not learn the underlying task well. This might be due to an insufficient amount of training data as only 1000 trajectories are used for each tree configuration. It can be observed that all networks yield similar maximum values for the collision volume. This is the case when the gripper and one of the trees completely overlap, yielding a collision volume of $0.38\,\mathrm{m}^3$. The 5×128 network yields the lowest collision rate of 53.9% and a median collision volume of $0.0054\,\mathrm{m}^3$.

Table 4.8: VP-STO approach with two additional obstacles: Collision rate for different net sizes

	2 layers	3 layers	4 layers	5 layers
128 neurons	71.1%	66.5%	60.5%	53.9%
512 neurons	66.1%	70.9%	62.5%	65.8%
1024 neurons	62.9%	55.9%	58.8%	55.8%

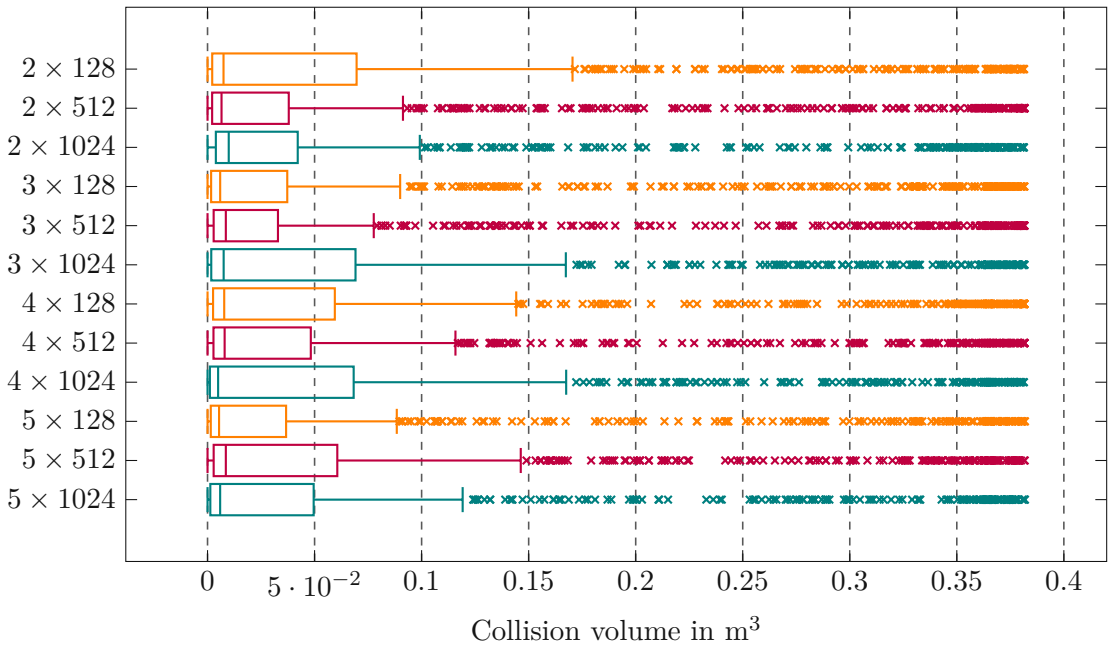


Figure 4.37: VP-STO approach with two additional obstacles: Collision volume of the two collision bodies

The trajectory duration in relation to the VP-STO algorithm is presented in Figure 4.38. Note that the 3×128, the 4×128, the 4×512, the 4×1024, the 5×512 and the 5×1024 network generate one or two trajectories from the test set that yield a trajectory duration of more than seven times the duration of VP-STO. These are not visualized in Figure 4.38 to be able to better evaluate the plot. It can be observed that all but the 3×512 network

generate trajectories that are shorter than those computed by VP-STO. As mentioned in Section 4.2.2, this can happen if the VP-STO algorithm finds a collision-free trajectory when $\mathbf{\Sigma}_{\text{via}}$ is already low, leading to a trajectory that is not as short as possible. However, the networks seem to compensate for these difficulties also in this scenario by learning to generalize. One of these trajectories, yielding a duration of 29.4% in relation to VP-STO, is visualized in Figure 4.39, the corresponding joint trajectories are presented in Figure 4.40. The 2×512 network yields the lowest variance regarding trajectory duration, the 5×1024 network yields the lowest median of 113.4%.

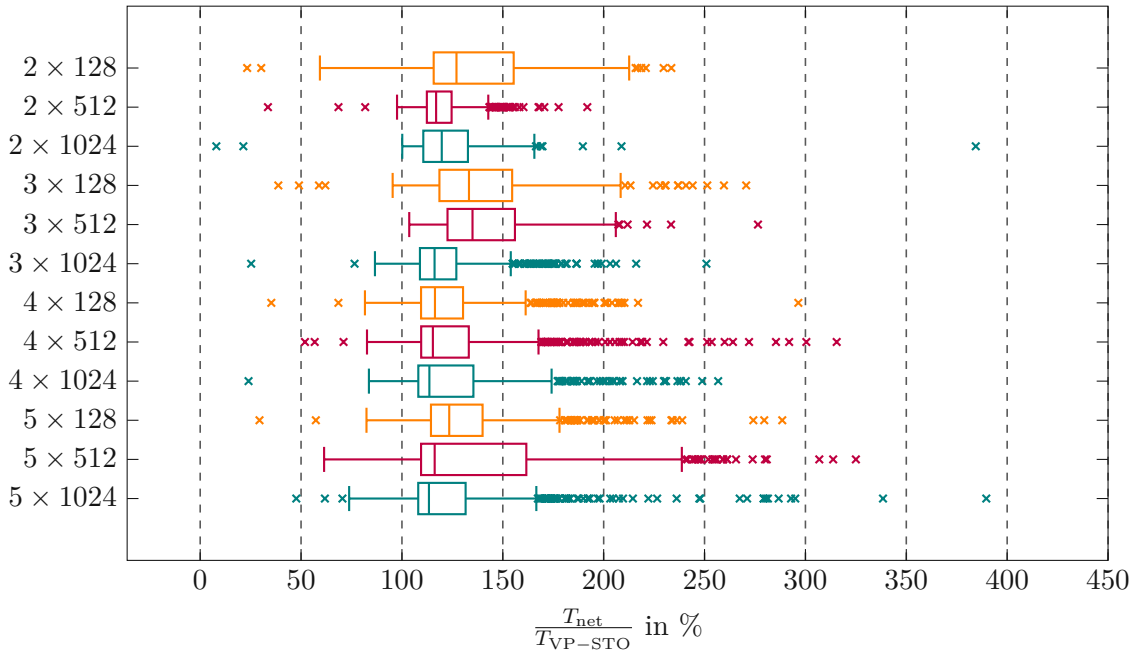


Figure 4.38: VP-STO approach with two additional obstacles: Trajectory time for collision-free trajectories generated by the net in relation to VP-STO

Three collision-free trajectories generated by the 5×128 network are visualized in Figure 4.41 and Figure 4.42, along with those computed by the VP-STO algorithm. Measurements of the computation time based on 100 trajectories yield 16.5 s for VP-STO and 4.7 ms for the network for one trajectory, showing the clear advantage of the net regarding computational complexity. Training this network takes approximately 15 min.

For imitating the VP-STO algorithm with two additional obstacles with varying position, none of the trained networks show satisfactory behavior as the collision rates on the test trajectories are high. However, it is possible that performance improves with more training trajectories for each tree configuration as Figure 4.9 shows that training and validation loss can be improved by using more training data. Figure 4.9 suggests that at least 4000 trajectories for each configuration are required to yield good results. However, it can be observed that the networks are able to generalize, yielding shorter collision-free trajectories than the ones found with VP-STO. Furthermore, the network shows a significantly lower computation time.

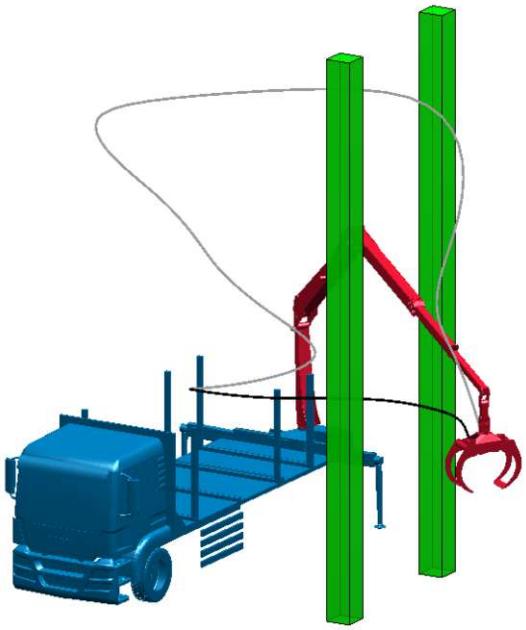


Figure 4.39: VP-STO approach with two additional obstacles: Shorter trajectory generated by the 5×128 network (black) than by VP-STO (gray)

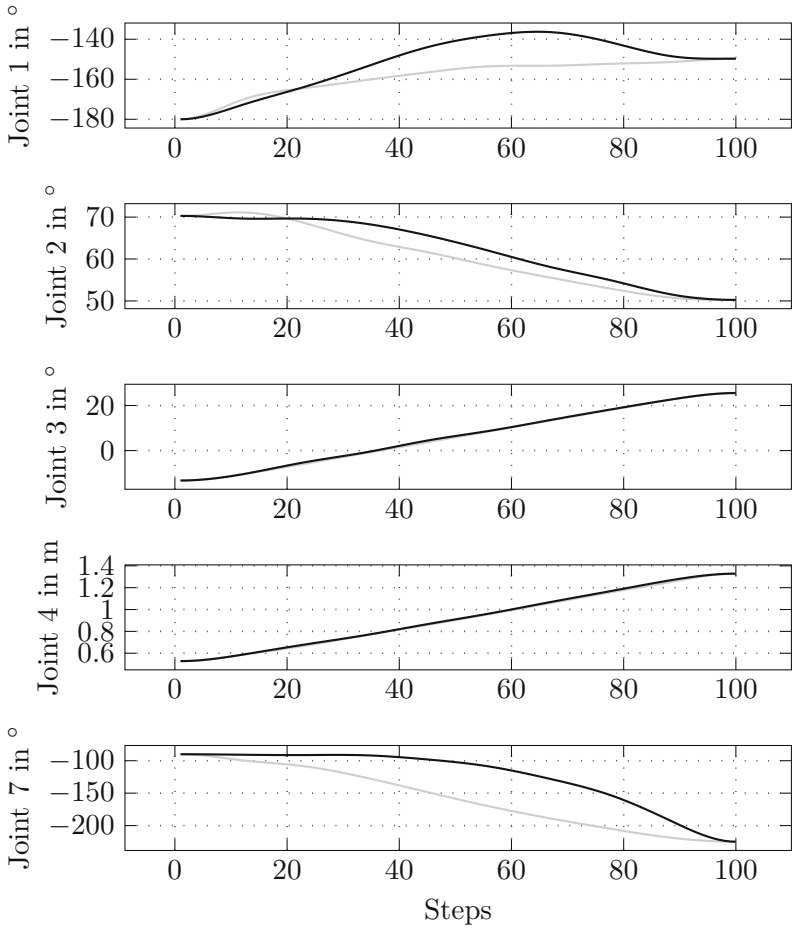
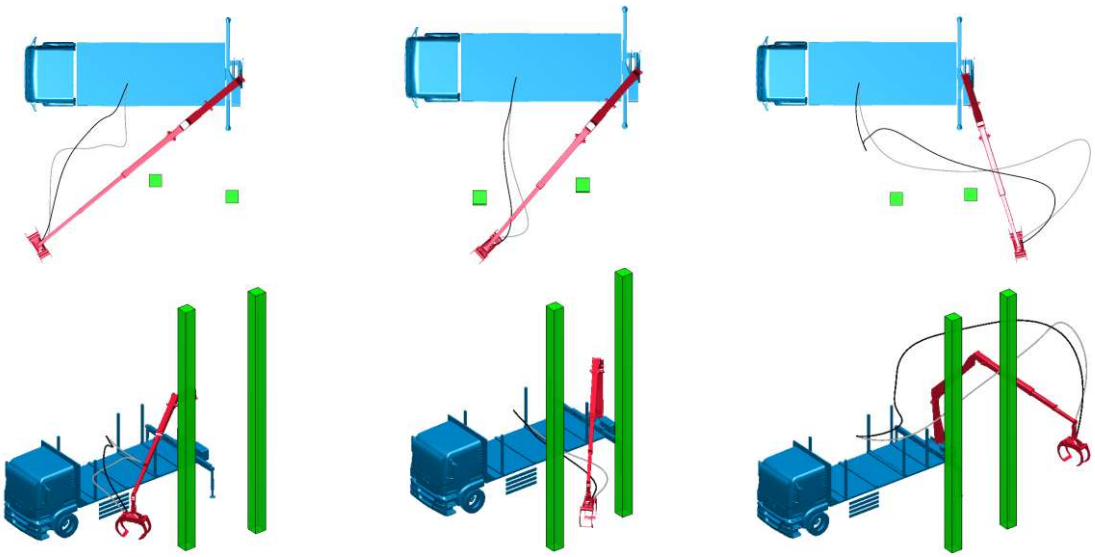


Figure 4.40: VP-STO approach with two additional obstacles: Shorter trajectory generated by the 5×128 network (black) than by VP-STO (gray), illustrated for each joint



(a) (b) (c)

Figure 4.41: VP-STO approach with two additional obstacles: Trajectories generated by the 5×128 network (black) and the VP-STO algorithm (gray) in top view and side view

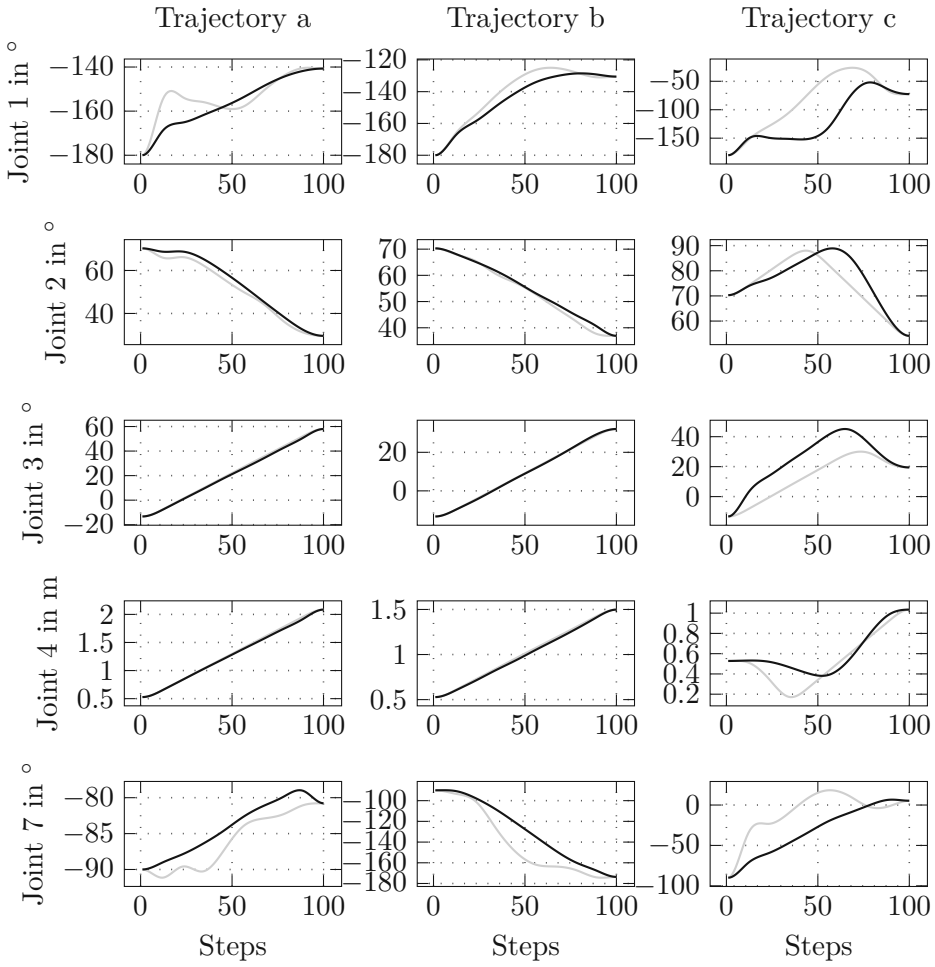


Figure 4.42: VP-STO approach with two additional obstacles: Trajectories generated by the 5×128 network (black) and the VP-STO algorithm (gray), illustrated for each joint

5 Conclusion

In this thesis, two approaches for learning-based motion planning for a timber crane in three different environmental settings were examined. Both approaches employ behavioral cloning, an imitation learning method, with an artificial neural network. One approach is to imitate the Via-point-based Stochastic Trajectory Optimization (VP-STO) algorithm described in [23] with a neural network that generates a trajectory in one shot by predicting via-points for each joint. The other method involves imitating the solution for an optimal control problem, which computes the trajectory iteratively by predicting the control input for the current system state.

Both approaches demonstrate good performance for the setting without additional obstacles and the scenario with one additional obstacle of variable height, indicating that the networks are capable of learning the motion planning task across varying environmental conditions through behavioral cloning. However, in the case of two additional obstacles with variable position, the network underperformed, with more than half of the generated trajectories resulting in collisions. Increasing the amount of training data might improve the network's performance, as indicated by an analysis of training trajectories required to obtain satisfactory results in the obstacle-free scenario.

Regarding network architecture, it can be observed that the more complex the motion planning task becomes, the larger the networks are required to be to ensure good performance.

The neural networks show a significant advantage considering computational speed over the optimization algorithms, making them a promising alternative for real-time applications. Furthermore, the networks are able to account for challenges of the VP-STO algorithm, such as producing non-time-optimal trajectories in complex environments. The networks learn to generalize and are able to find shorter, collision-free trajectories in these cases.

However, the networks are not robust against parameter variations as they yield high collision depths and high goal errors in the presence of parameter values they were not trained on, while the optimizing algorithm still manages to successfully generate about two-thirds of the trajectories.

For future studies, some enhancements are proposed:

- **Improving the training data:** The results show that the networks imitating VP-STO sometimes produce trajectories that are shorter than the ones computed by the algorithm itself, indicating that the data the networks are trained on is not optimal. Ensuring that VP-STO consistently outputs the shortest possible trajectory would make the networks' training more efficient and potentially yield improved results. The key challenge here is to develop an automated method to verify whether a generated trajectory is optimal.

- **Adapting the loss function for training:** In the case of training the VP-STO network, the loss function could involve not only minimizing the position error but also the curvature error between the predicted and optimal via-points, possibly improving the precision with which the network learns desired trajectories.
- **Using reinforcement learning:** Performance is assessed on the basis of collision rate and trajectory duration of the net-generated trajectories in the case of the VP-STO networks and on the basis of collision rate, limitation violation and goal error for the optimal control networks. However, the networks are only trained to minimize the mean squared error between predicted and optimal via-points and control inputs, respectively. Hence a low loss does not guarantee minimization of the criteria mentioned before. To account for these aspects specifically, reinforcement learning could be applied instead of imitation learning, ensuring that lower losses directly translate to better performance with respect to the specified criteria.

Furthermore, even though the produced trajectories are not as precise as the optimizing algorithms, they can still serve as initial guesses for optimizers, allowing for faster convergence in complex environments.

Bibliography

[1] K. M. Lynch and F. C. Park, *Modern robotics*. Cambridge University Press, 2017.

[2] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[3] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*, Kobe, Japan: IEEE, May 2009. [Online]. Available: http://dx.doi.org/10.1109/ROBOT.2009.5152817.

[4] J. Schulman *et al.*, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014. [Online]. Available: http://dx.doi.org/10.1177/0278364914528132.

[5] Y. Yang, J. Pan, and W. Wan, "Survey of optimal motion planning," *IET Cyber-Systems and Robotics*, vol. 1, no. 1, pp. 13–19, Jun. 2019. [Online]. Available: http://dx.doi.org/10.1049/iet-csr.2018.0003.

[6] S. LaValle, "Planning algorithms," *Cambridge University Press*, vol. 2, pp. 3671–3678, 2006.

[7] X. Xiao, B. Liu, G. Warnell, and P. Stone, "Motion planning and control for mobile robot navigation using machine learning: A survey," *Autonomous Robots*, vol. 46, no. 5, pp. 569–597, Mar. 2022. [Online]. Available: http://dx.doi.org/10.1007/s10514-022-10039-8.

[8] Q. Le Lidec, W. Jallet, I. Laptev, C. Schmid, and J. Carpentier, "Enforcing the consensus between trajectory optimization and policy learning for precise robot control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, London, UK: IEEE, May 2023. [Online]. Available: http://dx.doi.org/10.1109/ICRA48891.2023.10160387.

[9] M. Soori, B. Arezoo, and R. Dastres, "Artificial intelligence, machine learning and deep learning in advanced robotics, a review," *Cognitive Robotics*, vol. 3, pp. 54–70, 2023. [Online]. Available: http://dx.doi.org/10.1016/j.cogr.2023.04.001.

[10] H. Pulver, F. Eiras, L. Carozza, M. Hawasly, S. V. Albrecht, and S. Ramamoorthy, "Pilot: Efficient planning by imitation learning and optimisation for safe autonomous driving," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, Czech Republic: IEEE, Sep. 2021. [Online]. Available: http://dx.doi.org/10.1109/IROS51168.2021.9636862.

[11] M. Srinivasan, A. Chakrabarty, R. Quirynen, N. Yoshikawa, T. Mariyama, and S. D. Cairano, "Fast multi-robot motion planning via imitation learning of mixed-integer programs," *IFAC-PapersOnLine*, vol. 54, no. 20, pp. 598–604, 2021. [Online]. Available: http://dx.doi.org/10.1016/j.ifacol.2021.11.237.

[12] J. Tenhumberg, D. Burschka, and B. Bäuml, "Speeding up optimization-based motion planning through deep learning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, Japan: IEEE, Oct. 2022. [Online]. Available: http://dx.doi.org/10.1109/IROS47612.2022.9981717.

[13] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, Jun. 1999. [Online]. Available: http://dx.doi.org/10.1016/S1364-6613(99)01327-3.

[14] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," vol. 1, D. Touretzky, Ed., 1988. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf.

[15] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018. [Online]. Available: http://dx.doi.org/10.1561/2300000053.

[16] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, Jun. 2010. [Online]. Available: http://dx.doi.org/10.1177/0278364910371999.

[17] T. Osa, N. Sugita, and M. Mitsuishi, "Online trajectory planning and force control for automation of surgical tasks," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 675–691, Apr. 2017. [Online]. Available: http://dx.doi.org/10.1109/TASE.2017.2676018.

[18] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 48–66, Feb. 2021. [Online]. Available: http://dx.doi.org/10.1109/TRO.2020.3006716.

[19] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada: IEEE, May 2019. [Online]. Available: http://dx.doi.org/10.1109/ICRA.2019.8793889.

[20] J. Carius, F. Farshidian, and M. Hutter, "Mpc-net: A first principles guided policy search," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2897–2904, Apr. 2020. [Online]. Available: http://dx.doi.org/10.1109/LRA.2020.2974653.

[21] A. Reske, J. Carius, Y. Ma, F. Farshidian, and M. Hutter, "Imitation learning from mpc for quadrupedal multi-gait control," May 2021. [Online]. Available: http://dx.doi.org/10.1109/ICRA48506.2021.9561444.

[22] M. Kazim, J. Hong, M.-G. Kim, and K.-K. K. Kim, "Recent advances in path integral control for trajectory optimization: An overview in theoretical and algorithmic perspectives," *Annual Reviews in Control*, vol. 57, p. 100 931, 2024. [Online]. Available: http://dx.doi.org/10.1016/j.arcontrol.2023.100931.

[23] J. Jankowski, L. Brudermüller, N. Hawes, and S. Calinon, "Vp-sto: Via-point-based stochastic trajectory optimization for reactive robot behavior," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, London, UK: IEEE, May 2023. [Online]. Available: http://dx.doi.org/10.1109/ICRA48891.2023.10160214.

[24] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, Apr. 1988. [Online]. Available: http://dx.doi.org/10.1109/56.2083.

[25] E. Gilbert and C.-P. Foo, "Computing the distance between general convex objects in three-dimensional space," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 1, pp. 53–61, 1990. [Online]. Available: http://dx.doi.org/10.1109/70.88117.

[26] T. A. Howell, B. E. Jackson, and Z. Manchester, "Altro: A fast solver for constrained trajectory optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China: IEEE, Nov. 2019. [Online]. Available: http://dx.doi.org/10.1109/IROS40897.2019.8967788.

[27] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Learning movement primitives," in *Robotics Research. The Eleventh International Symposium*. Springer Berlin Heidelberg, 2005, pp. 561–572. [Online]. Available: http://dx.doi.org/10.1007/11008941_60.

[28] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, Feb. 2013. [Online]. Available: http://dx.doi.org/10.1162/NECO_a_00393.

[29] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 2, pp. 286–298, Apr. 2007. [Online]. Available: http://dx.doi.org/10.1109/TSMCB.2006.886952.

[30] B. Zheng, S. Verma, J. Zhou, I. W. Tsang, and F. Chen, "Imitation learning: Progress, taxonomies and challenges," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 5, pp. 6322–6337, May 2024. [Online]. Available: http://dx.doi.org/10.1109/TNNLS.2022.3213246.

[31] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys*, vol. 50, no. 2, pp. 1–35, Apr. 2017. [Online]. Available: http://dx.doi.org/10.1145/3054912.

[32] I. N. da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. d. R. Alves, *Artificial Neural Networks: A Practical Course*. Springer International Publishing Switzerland, 2017.

[33] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[34] A. Wichert and L. Sa-Couto, *Machine Learning - A Journey to Deep Learning*. WORLD SCIENTIFIC, Dec. 2020. [Online]. Available: http://dx.doi.org/10.1142/12201.

[35] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks for Perception*. Elsevier, 1992, pp. 65–93. [Online]. Available: http://dx.doi.org/10.1016/B978-0-12-741252-8.50010-8.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference for Learning Representations*, San Diego, USA, 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1412.6980.

[37] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[38] M.-P. Ecker, "Iterative linear quadratic regulator for collision-free trajectory optimization and model predictive control of a timber crane," M.S. thesis, TU Wien, 2022.

[39] J. Angeles, *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms*. Springer New York, 2003. [Online]. Available: http://dx.doi.org/10.1007/b97597.

[40] L. Sciavicco and B. Siciliano, *Modelling and control of robot manipulators*. Springer Science & Business Media, 2012.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, im Mai 2024

Marlene Birkelbach