robotics

Article

Adjustable and Adaptive Control for an Unstable Mobile Robot Using Imitation Learning with Trajectory Optimization

Christian Dengler * and Boris Lohmann

Automatic Control, Technical University of Munich, 80333 Munich, Germany; lohmann@tum.de

* Correspondence: c.dengler@tum.de

Received: 21 February 2020; Accepted: 22 April 2020; Published: 25 April 2020

Abstract: In this contribution, we develop a feedback controller in the form of a parametric function for a mobile inverted pendulum. The control both stabilizes the system and drives it to target positions with target orientations. A design of the controller based only on a cost function is difficult for this task, which is why we choose to train the controller using imitation learning on optimized trajectories. In contrast to popular approaches like policy gradient methods, this approach allows us to shape the behavior of the system by including equality constraints. When transferring the parametric controller from simulation to the real mobile inverted pendulum, the control performance is degraded due to the reality gap. A robust control design can reduce the degradation. However, for the framework of imitation learning on optimized trajectories, methods that explicitly consider robustness do not yet exist to the knowledge of the authors. We tackle this research gap by presenting a method to design a robust controller in the form of a recurrent neural network, to improve the transferability of the trained controller to the real system. As a last step, we make the behavior of the parametric controller adjustable to allow for the fine tuning of the behavior of the real system. We design the controller for our system and show in the application that the recurrent neural network has increased performance compared to a static neural network without robustness considerations.

Keywords: imitation learning; adaptive control; machine learning; mobile robot

1. Introduction

The control of mobile and unstable systems is, in most cases, divided into a stabilizing and a maneuvering part, e.g., [1]. This breakdown of the problem into two separate tasks makes analytic control designs manageable, however, the final performance of the system will be limited compared to holistic approaches. General holistic approaches are difficult to derive analytically for nonlinear systems. Thus, methods based on optimization and learning come into play that train a control law in the form of a parametric function based on a cost function. Most notably, methods emerged from the field of reinforcement learning, e.g., policy gradient methods like PPO [2], are used, as well as methods from the field of gradient-free optimization. For systems with relatively slow dynamics, nonlinear model predictive control can be used [3], which is, however, not suited for fast systems due to the continuous online optimization with non-deterministic computing time. In this work, we design a parametric controller for the position and orientation control of a mobile inverted pendulum (MIP), without the need to compute trajectories online.

The control laws derived based on the aforementioned approaches are generally trained in a simulation environment and often do not transfer well to the real system due to model inaccuracies, also called the reality gap. In order to overcome this challenge, different approaches have been developed. Some methods update or learn a highly precise model online, in order to reduce the reality gap [4,5]. Other approaches use adversarial attacks during the training of the controller in order to enforce robustness [6,7]. Other works focused on training a controller that is trained on a multitude of varying model dynamics, also called model ensembles [8–10]. The current research on overcoming the reality for this type of control law is mostly focused on methods of the domain of reinforcement learning. In this work, we tackle the reality gap problem for a less common class of design methods.

We use a method that is related to imitation learning [11] and has been referred to as a form of explicit model predictive control [12]. The method was also mentioned in [13], where a more complex and expensive optimization problem was then derived. In a second step, we extend the method to adapt online to model uncertainties by using a dynamic control law in the form of a recurrent neural network and by randomizing model parameters during training. Different from Peng et al. [9] who also use a dynamic control law, our method is based on the imitation learning framework instead of reinforcement learning, which allows us to shape the behavior of the system using equality constraints. With the objective of the application on the real system, we also make the behavior of the final control law adjustable by an external user via a trivial extension of the method.

2. Related Work

This section gives an overview of methods and research in two different areas related to our approach. The first area deals with the learning of parametric controllers that have a similar control performance in simulation and in the application on a real system. The second area is concerned with learning parametric controllers, based on a combination of imitation learning and trajectory optimization.

2.1. Training Robust Parametric Controllers

Most parametric controllers are trained in simulation due to the high cost of real-world data. The gap between the simulation model and reality can lead to poor performance or even failure of the controller when applied to the real system. Many articles have been published that either tackle the problem of reducing the reality gap or train a controller that can perform well despite the reality gap.

In order to improve the accuracy of the simulation model and as such reduce the reality gap, data concerning interactions with the real system have to be considered. Abbeel et al. [14] use an iteratively updated model to compute an improvement direction for the control parameters using a policy-gradient method. The step-size of the update is, however, determined on the real system. This is beneficial, as determining a good step size requires less interaction with the real system than estimating a gradient direction. The model is retrained after each update of the control parameters with new data. In Deisenroth and Rasmussen [4] a model in the form of a Gaussian Process is iteratively updated and a control improvement is done offline after each update, using gradient descent on the accumulated costs. In Golemo et al. [5], the model error is approximated using a recurrent neural network, trained on data from the real system, and a controller is then derived in simulation using the policy-gradient algorithm PPO [2].

Recently, more and more works focus on designing controllers that are robust with regards to model errors, rather than further increase the model accuracy. The methods can be roughly classified in methods that use a multitude of model parameters for training and methods that use a disturbing entity, called the adversary. A set of different model parameters is used in Mordatch et al. [15] to compute robust trajectories for a humanoid robot. As such, this work does not fall completely in the scope of robust parametric control, but rather robust trajectory generation. A non-parametric controller was used to deal with small system deviations online. Rajeswaran et al. [8] propose to train robust controllers using randomizing model parameters and a policy-gradient algorithm, while using only a subset of the worst trajectories to approximate the policy-gradient. This can be seen as a relaxation of a min-max formulation, common in nonlinear robust control [16]. An adversarial approach is introduced in Pinto et al. [7] to train a robust controller. Both the controller and the adversary are parametric functions that are trained using a policy-gradient algorithm. The approach also approximates the min-max problem, with the robust controller minimizing the total costs, and the adversary maximizing them again through bounded adversary actions. A different approach is presented in Yu et al. [17]. Instead of approximating a min-max solution, the parametric controller is trained with uncertain model parameters as additional inputs using the policy-gradient algorithm TRPO [18]. As the parameters of the real system are unknown during the application, an estimator/observer is trained in a second step, and the union of both estimator and controller are able to adapt to the real dynamics by guessing the current system parameters during application. Pattanaik et al. [6] fool the policy during training by perturbing the perceived state. In this case, the perturbations, or adversary attacks, do not result from a second controller but are either sampled randomly or computed based on the action-value function. Muratore et al. [10,19] use randomized model parameters to estimate the simulation optimization bias and prevent the learned policy from overfitting to the simulator. While the focus of Bousmalis et al. [20] lies on their vision component, the model parameters of the simulated robot arm are also randomized in order to improve the robustness. Peng et al. [9] train a recurrent neural network using reinforcement learning, while randomizing the model parameters. While the recurrent neural network only sees the current states and needs to learn a hidden state representation, the critic, i.e., the action value function, is omniscient in the sense that it has the model parameters as additional input. Chebotar et al. [21] also randomize model parameters, but focus their contribution on adapting the distribution of the model parameters iteratively by also including interactions of the real system. The policy is again learned using the policy-gradient algorithm PPO [2].

When looking at existing literature in this field, it is noticeable that the parametric controllers are solely trained using policy-gradient algorithms. In the present time, the most widely used methods to train a parametric controller stem from the field of reinforcement learning. However, alternatives exist and every method has its own strengths and weaknesses. Some methods from the field of gradient-free optimization, most notably evolution strategies, have proven to be competitive or even superior in certain cases [22,23]. Reinforcement learning algorithms learn by exploring in action-space and by increasing the probability of better than average actions, while evolution strategies explore in parameter space and increase the probability of better than average parameters [24]. Yet another class of algorithms to learn parametric controllers, which is in the focus of our contribution, is a combination of trajectory optimization and supervised learning. This can be seen as a special case of imitation learning. This last class of algorithms proved to be the best suited for the task at hand for reasons mentioned later, but methods focusing on robustness and transferability have to date not been investigated upon to the best of our knowledge.

2.2. Imitation Learning with Trajectory Optimization

As the name imitation learning suggests, the field is concerned with learning a parametric control law that imitates the behavior of a teacher, based on demonstrations on how to solve the task. In most of the current literature concerned with imitation learning, the expert is considered to be a human. A survey on applied methods with a human expert can be found in [11]. Since we will not have a human expert, we only consider, with a few exceptions, related works where the teacher is not a human but an optimizer. The approach of training a controller by combining trajectory optimization with supervised learning is far from new, yet no umbrella name for these methods has been established in literature. In this paper, we refer to these methods as imitation learning with trajectory optimization. The following literature contains works from both the control and machine learning communities.

An early application of this method is found in [25], where a neural network is trained as a controller for a wheeled robot. The task is to follow a desired path while avoiding obstacles with results also shown in the real application. An implementation with results in simulation for a chemical process can be found in Åkesson et al. [26]. An application for the control of a semi-active suspension system of the same method is shown in [12] where the authors report improved control performance with regards to their previous analytic control design.

Ross et al. [27] tackle a fundamental problem of imitation learning: the states visited by the expert during the demonstration and later used for supervised learning and the states that the controller will visit during execution are not distributed equally. Therefore, they propose an iterative approach where the controller samples states during execution and the expert provides actions for those states, rather than showing his own solution to the task. The training data obtained in this way is aggregated in a growing data-set. The method, called DAGGER for data-set aggregation, was proposed for use with human experts. He et al. [28] use DAGGER with an artificial expert. In each iteration their expert generates actions that are suboptimal, yet close to the current controller, with the intention of creating actions that can be learned faster than the optimal action. They show small improvements with regards to an expert that always generates optimal actions. Laskey et al. [29] propose adding noise to an artificial expert's demonstrations with the aim of learning a policy that is able to recover from errors. The power of the noise is optimized such that the distribution of expert trajectories approaches the distribution of trajectories produced by the parametric controller in the loop. The approach is called DART for "disturbances for augmenting robot trajectories".

Mordatch and Todorov [13] combine the trajectory optimization step with the supervised learning step by combining their objectives into a single optimization problem: minimizing costs of states and actions (trajectory optimization) and learning a parametric controller that can reproduce similar actions (supervised learning step). They then solve the optimization problem by iterating between regularized trajectory optimization and supervised learning. On simulation tasks, they show a decrease in the expected costs for their method, compared to a separation of trajectory optimization and supervised learning. In a later work [30], Mordatch et al. show the performance for more complex simulations and add a small number of recurrent states as well as sampled noise to their optimization setting.

Levine and Koltun [31,32] formulate a similar idea as a stochastic optimization problem by minimizing the Kullback-Leibler divergence between the distribution of actions produced by the controller and the actions of a distribution with high probability of low costs. They name their approach guided policy search. Guided policy search is also used by Zhang et al. [33] for the obstacle avoidance of a simulated quadrotor. However, they use a simulation of a model predictive controller (MPC) instead of optimizing the trajectory as a whole, and the supervised learning step uses only a subset of the states that the MPC was allowed to use. They extend their algorithm for the simulated quadrotor problem in Kahn et al. [34] to use an adaptive MPC during data generation and call their approach PLATO. As such, the final controller works on states that are available during real flights instead of the augmented state representation available to the MPC. Guided policy search also allows Levine et al. [35] to learn a convolutional neural network controller for a physical robot arm, using only pixel data as control inputs.

We refrain from using the previously mentioned methods, coupling supervised learning and trajectory optimization into a single objective, due to the increase in computation time and complexity and our focus on transferability to the physical system.

3. Design of an Adaptive and Adjustable Controller Using Imitation Learning

In this section, we present a new approach for training a parametric controller offline that is robust to model uncertainties by adapting to the system's behavior during execution. The approach uses randomized dynamics and a controller with internal states, e.g., a recurrent neural network. The idea has similarities to [9], however, we use imitation learning with trajectory optimization instead of reinforcement learning. Different from [9], the approach that is introduced in this section can use time-dependent cost functions and equality constraints to define the behavior of the system. The inclusion of equality constraints is essential for the control design of our system, as is explained later in Section 5.1. While the final controller has recurrent states to account for model uncertainties similar to [9], the training method and therefore the admissible problem setting differ fundamentally. Moreover, the presented approach consists of a sequence of well established and understood sub-problems, whereas the success of reinforcement learning algorithms depends on hyper-parameters that can be difficult to tune [36].

Training a controller using supervised learning on optimized trajectories does not work for controllers with recurrent states. We leverage the problem by using ideas similar to DAGGER [27] and DART [29]. The approach uses an intermediate policy with model parameters as input, similar to the universal policy in [17], which is necessary to reduce computation time to an acceptable level. The intermediate policy is called the oracle network and acts as a teacher for the final recurrent controller. The oracle network itself can only be used in simulation, whereas the recurrent controller is suitable for both simulation and application.

The approach is divided into three consecutive optimizations and does not require the real system in the loop. The three main parts are

1. Trajectory optimization with randomized model parameters.
2. Training an intermediate oracle network.
3. Training of a controller with internal states.

We also show a trivial extension of the approach, in order to make the final controller adjustable by the operator. In the case of, e.g., a neural network controller, the parameters of the controller are not interpretable as is the case, e.g., for PID controllers. By making the controller adjustable, we do not have to repeat the whole optimization if only a small change in the behavior is required.

In the following we assume a dynamics model that is discretized in time and continuous in the state and action space. We denote the system state at time t by $\mathbf{x}_t \in \mathbb{R}^n$ and the control signal, or action $\mathbf{u}_t \in \mathbb{R}^m$. We also assume for simplicity that the dynamics model is deterministic, as is most often the case for models that are derived from laws of physics. The dynamics also depend on constant but possibly uncertain parameters $\mathbf{p}$, e.g., friction coefficients, moments of inertia. The nonlinear state space model is of the form

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t; \mathbf{p}). \tag{1}$$

3.1. Trajectory Optimization

The first part of our method consists of creating and storing optimal trajectories starting from relevant initial states. We also limit ourselves to trajectories over a fixed time horizon T. A distribution $D_{x0}(\mathtt{x})$ over relevant starting states $\mathbf{x}_0$ is defined by the control engineer. If a policy is directly derived using supervised learning on the trajectories, it is necessary to include starting states in $D_{x0}(\mathtt{x})$ with velocities that are diverging from the desired state or trajectory. Otherwise, the learned policy will not be able to recover once diverging, e.g., because of a perturbation or simply because of modeling errors. In order to have a robust policy at the end, we also want to define a distribution over uncertain model parameters $D_p(\mathtt{p})$. Moreover, the control engineer has to define the time horizon T over which the trajectories are optimized and provide a cost function $c(\mathbf{x}_t, \mathbf{u}_t)$ and end costs $c_T(\mathbf{x}_T)$ defining the behavior and goal of the control.

Let $\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_T]$ and $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{T-1}]$ be the states and control inputs of a trajectory. An optimal trajectory for a given starting state $\mathbf{x}_0 \sim D_{x0}(\mathbf{x})$ and parameter set $\mathbf{p} \sim D_p(\mathbf{p})$ is calculated by solving:

$$(\mathbf{X}^*, \mathbf{U}^*) = \arg\min_{\mathbf{X},\mathbf{U}} \sum_{t=0}^{T-1} c(\mathbf{x}_t, \mathbf{u}_t) + c_T(\mathbf{x}_T) \tag{2a}$$

$$s.t.\ \mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t; \mathbf{p}) \tag{2b}$$

$$\mathbf{x}_{t=0} = \mathbf{x}_0 \tag{2c}$$

$$\mathbf{eq}(\mathbf{X}, \mathbf{U}) = \mathbf{0} \tag{2d}$$

$$\mathbf{ieq}(\mathbf{X}, \mathbf{U}) \leq \mathbf{0}. \tag{2e}$$

Equations (2d) and (2e) represent equality and inequality constraints, which can be used to consider e.g., bounded control inputs or to enforce convergence to a specific terminal state $\mathbf{x}_{T,\text{ref}}$. If the control signals are not smooth, the inequalities can also be used to enforce smooth signals. This is advised as the control will be approximated by a smooth function approximator later.

The problem in Equation (2a) can be solved by either indirect or direct methods [37]. We choose a direct method called multiple shooting [38], as the method works naturally with discretized dynamics and direct methods are known to converge better than indirect methods for initializations that are far from the optimal solution. On the other hand, direct methods are known to produce less accurate solutions. For direct multiple shooting, all equality constraints, i.e., Equations (2b) and (2d) are handled using Lagrange multipliers. Either sequential quadratic programming or interior point methods, also called barrier methods, can be used to solve the problem including inequality constraints [39]. We use an open-source solver IPOPT [40], which implements an interior point solver.

It is important to note that the optimal action $\mathbf{u}^*$ for a state $\mathbf{x}$ can be ambiguous due to the finite optimization horizon T, as well as the constraints (2d)–(2e). The finite horizon and the optimization constraints can cause the optimal $\mathbf{u}^*$ to not only depend on a state $\mathbf{x}$ but also on the time t, i.e., whether the state $\mathbf{x}$ was visited early or late in the trajectory. As an example, consider the problem

$$(\mathbf{X}^*, \mathbf{U}^*) = \arg\min_{\mathbf{X},\mathbf{U}} \sum_{t=0}^{3} x^2 \tag{3a}$$

$$s.t.\ x_{t+1} = x_t + u_t \tag{3b}$$

$$x_0 = 1;\ x_3 = 1 \tag{3c}$$

where the constraint $x_3 = 1$ forces the trajectory to return to a region of high costs. The solution to the problem is $\mathbf{X}^* = [1, 0, 0, 1]$ and $\mathbf{U}^* = [-1, 0, 1]$. In this artificial example, the state $x = 0$ is visited at $t \in 1, 2$, and the optimal actions are $u_1 = 0, u_2 = 1$.

Thus, in order to provide unambiguous training targets, only the first optimal state-action pair $(\mathbf{x}_0^*, \mathbf{u}_0^*)$ would have to be used. That would however be computationally very expensive, which is why we will use all state-action pairs $(\mathbf{X}^*, \mathbf{U}^*)$ during training. We believe that for any problems with non-neglectable optimization time, the trade-of of introducing ambiguous state-action targets by using all state-action pairs of the trajectories can be justified by the reduction in computation time.

The optimization in Equation (2) is performed multiple times for different $\mathbf{x}_0 \sim D_{x0}(\mathrm{x}), \mathbf{p} \sim D_p(\mathrm{p})$ until a sufficient coverage of the distributions is achieved. The N optimal trajectories, together with the corresponding model parameters $\mathbf{p}$ are stored for later use as tuples $(\mathbf{X}_k^*, \mathbf{U}_k^*, \mathbf{p}_k);\ k \in \{1, \ldots, N\}$.

3.2. Oracle Training

After a multitude of optimal trajectories has been obtained, we train a controller that uses both the state $\mathbf{x}_t$ and model parameters $\mathbf{p}$ as inputs, similar to Yu et al. [17]. This controller can only be used in simulation when the model parameters are known. However, it would be problematic to guess good parameters when applying the control to the real system. Therefore, this controller is only an intermediate result and will be used as a teacher for the recurrent controller.

The control law is represented by a parametric function $\mathbf{u}_t = \mathbf{g}(\mathbf{x}_t, \mathbf{p}; \boldsymbol{\Theta})$. Our intermediate controller is learned using supervised learning on state-action pairs of the optimal trajectories. This is a simple regression problem and optimal parameters can be found by solving a regression problem:

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} \kappa \sum_{k=1}^{N} \sum_{t=0}^{T-1} \left\| \mathbf{u}_{t,k} - \mathbf{g}(\mathbf{x}_{t,k}, \mathbf{p}_k; \boldsymbol{\Theta}) \right\|^2 \tag{4}$$

with $\kappa = \frac{1}{N \cdot (T-1)}$.

In order to avoid overfitting onto the data, it is generally advised to use cross-validation, i.e., to split the data into a dataset for training and a dataset to test the model performance.

The best way to solve (4) depends on the parametrization of $\mathbf{g}(\ldots)$. We use fully connected neural networks in this paper and solve Equation (4) using stochastic gradient descent with the optimizer Adam [41]. We call this intermediate parametric controller the oracle, as it is provided with the full model information to generate its output.

3.3. Training a Robust Recurrent Network

As previously mentioned, the control law obtained in Section 3.2 is not suited for use on a physical system, as it requires knowledge of the underlying parameters, which we assume uncertain. We circumvent this problem by learning a control law that adapts to the dynamics of the system. As in Peng et al. [9], the new control law will use the history of past states and actions to generate the new control signal:

$$\mathbf{u}_t = \mathbf{g}(\mathbf{x}_t, \mathbf{u}_{t-1}, \mathbf{x}_{t-1}, \ldots, \mathbf{u}_0, \mathbf{x}_0; \boldsymbol{\Theta}). \tag{5}$$

as the state transitions and thus the history $(\mathbf{x}_t, \mathbf{u}_{t-1}, \mathbf{x}_{t-1}, \ldots, \mathbf{u}_0, \mathbf{x}_0)$ are different for each parameter set $\mathbf{p}$, a control law in the form of (5) can produce different control outputs for different $\mathbf{p}$.

The parametrization of a function in the form of (5) would be inefficient, therefore a common approach is to learn a representation of the past history as an internal state $\mathbf{h}_t$, also called hidden state. A recurrent neural network is an example for such a representation. The general structure of recurrent neural networks is:

$$\mathbf{h}_t = \mathbf{r}_1(\mathbf{x}_t, \mathbf{h}_{t-1}; \boldsymbol{\Theta}_1) \tag{6a}$$

$$\mathbf{u}_t = \mathbf{r}_2(\mathbf{h}_t; \boldsymbol{\Theta}_2). \tag{6b}$$

We mostly use a shorter notation with $\boldsymbol{\Theta} = [\boldsymbol{\Theta}_1^T, \boldsymbol{\Theta}_2^T]^T$, combining (6a) and (6b) into a single function. The shorter notation for a recurrent network is then

$$[\mathbf{u}_t^T, \mathbf{h}_t^T]^T = \mathbf{r}(\mathbf{x}_t, \mathbf{h}_{t-1}; \boldsymbol{\Theta}). \tag{7}$$

Unlike the system state $\mathbf{x}_t$, the learned hidden state is generally not interpretable as a physical quantity.

The training of the parameters $\boldsymbol{\Theta}$ for recurrent neural networks using supervised learning is usually done using backpropagation through time. A loss between training inputs and training outputs is defined and a gradient is computed over a sequence of data. For regression problems, the mean squared error is used.

$$[\boldsymbol{\Theta}_1^*, \boldsymbol{\Theta}_2^*] = \underset{\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2}{\arg\min}\, \kappa \sum_{k=1}^{N} \sum_{t=0}^{T-1} \left\| \mathbf{u}_{t,k} - \mathbf{r}_2(\mathbf{h}_{t,k}; \boldsymbol{\Theta}_2) \right\|^2 \tag{8a}$$

$$s.t.\ \mathbf{h}_{t,k} = \mathbf{r}_1(\mathbf{x}_{t,k}, \mathbf{h}_{t-1,k}; \boldsymbol{\Theta}_1) \tag{8b}$$

$$\mathbf{h}_{(-1),k} = \mathbf{0}. \tag{8c}$$

Training recurrent neural networks is computationally more expensive, as the gradient is computed by backpropagating through a sequence, which is similar to backpropagating through a deep neural network with the depth growing according to the sequence length. For long sequences, it is common to accept a bias in the gradient direction by truncating long sequences into smaller chunks [42]. Computing the loss gradient over truncated sequences is called truncated backpropagation through time (TBPTT).

A naive training of a recurrent neural network on sequences generated during the trajectory optimization will not lead to a good control law in the closed loop. This is due to the fact that sequences in the training data and the sequences seen during execution of the controller stem from different distributions. All training sequences are sequences included in optimal trajectories. Once the controller deviates from those trajectories, no training data is available. To leverage the problem, we use a mixture of ideas from DAGGER [27] and DART [29] together with our artificial expert trained according to Section 3.2. We train our recurrent neural network on sequences seen during execution of the recurrent neural network, but with actions recommended by the oracle network. Our small changes to DAGGER are the fact that we do not accumulate datasets, but learn for a limited number of iterations on the last data-set only. This is motivated by the fact that data generation using our artificial expert is cheap and early trajectories will not be relevant for the sequences that the final controller will produce. Early trajectories can thus be discarded for reasons of efficiency. Similar to DART we also add a small noise to the data gathering to have more variance in the sequences, especially close to convergence. However, we add our noise to the learning controller instead of to the expert. That way the distribution of sampled trajectories is always close to trajectories of the learning controller and we do not have to minimize the Kullback-Leibler divergence between two distributions, as is done in DART. Pseudo-code for generating training data for the recurrent neural network is give in Algorithm 1. Pseudo-code of our DAGGER and DART combination, which we abbreviate in the following as disturbed oracle imitation (DOI), is shown in Algorithm 2.

A visual comparison of the three algorithms DAGGER, DART and DOI is given in Figure 1. DAGGER generates training data on the trajectory created by the controller in the loop and accumulates all data. DART generates data for a distribution of trajectories generated by an expert, with the distribution variance adapted to the trajectories of the controller. DOI generates data on trajectories close to trajectories of the controller in the loop, but does not accumulate past data.

Algorithm 1 Generating training data for the recurrent neural network

```
Inputs: $\mathbf{g}, D_{x0}, D_{p0}, \epsilon$
$\mathbf{x}_0 \sim D_{x0}(\mathbf{x}), \mathbf{p} \sim D_{p0}(\mathbf{p}), \mathbf{h}_{(-1)} = \mathbf{0}$
for t = 0 to $T$ do
    $\hat{\mathbf{u}}_t = \mathbf{g}(\mathbf{x}_t, \mathbf{p})$                                ▷ Evaluate oracle
    $\mathbf{u}_t, \mathbf{h}_t = \mathbf{r}(\mathbf{x}_t, \mathbf{h}_{t-1}; \mathbf{\Theta})$                ▷ Evaluate controller
    $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t; \mathbf{p}) + \mathcal{N}(\mathbf{0}, \epsilon^2 \mathbf{I})$        ▷ Disturbed dynamics
end for
return $\mathbf{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_{t-1}], \hat{\mathbf{U}} = [\hat{\mathbf{u}}_0, \ldots, \hat{\mathbf{u}}_{t-1}]$
```

Algorithm 2 Disturbed Oracle Imitation (DOI)

```
for epoch = 1 to $N_{epoch}$ do
    $\mathcal{D} \leftarrow \emptyset$
    for traj = 1 to $N_{traj}$ do
        sample sequence $\mathbf{X}_{\text{traj}}, \hat{\mathbf{U}}_{\text{traj}}$
        $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{X}_{\text{traj}}, \hat{\mathbf{U}}_{\text{traj}})$
    end for
    for gd = 1 to $N_{gd}$ do
        Update $\mathbf{\Theta}$ using TBPTT.
    end for
end for
```

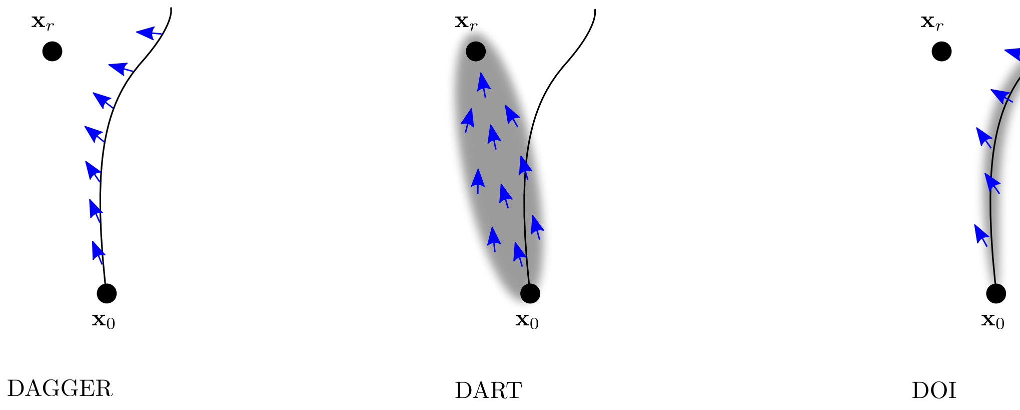


Figure 1. Visual comparison of DAGGER, DART and DOI. $\mathbf{x}_0$ inidicates the intial state and $\mathbf{x}_r$ a reference state with low costs. The black line indicates a trajectory sampled using the parametric controller in the loop. Blue arrows indicate the training data created for each approach, possibly generated for a distribution of trajectories, indicated by a blurred area.

At this point, the reason for the oracle network has to be clarified, since it is a reasonable assumption that one could just solve the optimization problem in Equation (2) to generate the training targets $\hat{\mathbf{u}}_t$ for each $\mathbf{x}_t$ in Algorithm 1. The reason is again the computational burden, as only the first action of the optimized trajectory would be used. Solving this many trajectory optimizations would only be feasible for small problems. Also, problems occur when a state is visited by the recurrent controller in early iterations where the trajectory optimization is unfeasible due to constraints. By creating the intermediate network, all $T-1$ actions in the optimized trajectories can be used.

For the problem in this paper presented in Section 4, the trajectory optimization is the most expensive part and $T = 500$. A direct use of the optimizer to create training targets $\hat{\mathbf{u}}_t$ had to be interrupted due to the extremely slow progress.

3.4. Adding Adjustable Behavior

The training of parametric controllers can require a lot of computation time. If the controller does not perform in the desired manner on the real system, the lengthy procedure needs to be repeated and time is lost. By a trivial extension of the presented method, we show how we can make the behavior of the controller adjustable. In the case of simple controllers, e.g., PID controllers, it is common practice to tune the final behavior in the closed loop with the real system [43]. For parametric controllers, however, simply tuning the parameters is not appropriate, as the influence of the change cannot be predicted and the number of parameters is usually high.

The behavior of the controller in our approach is defined by the cost function in Equation (2a). In a similar fashion as we design a controller to work on multiple models, we can design a controller to work on multiple cost functions.

In an initial step, the cost function is augmented to depend on an additional parameter λ in a bounded range, e.g., $\lambda \in [-1, 1]$. The influence of the parameter can be, e.g., to shift between penalizing large control signals and penalizing the state costs. The optimal trajectories are then initialized with random initial states and model parameters as well as a cost function parameter sampled from a uniform distribution covering the range. This can also be interpreted as augmenting the state $\mathbf{x}_t$ with an additional state λ, with dynamics $\lambda_{t+1} = \lambda$.

The oracle network $\mathbf{g}_\lambda(\mathbf{x}_t, \mathbf{p}, \lambda; \mathbf{\Theta})$ is then trained using supervised learning on the optimal trajectories. The recurrent controller has the state $\mathbf{x}$ and λ as inputs:

$$[\mathbf{u}_t^T, \mathbf{h}_t^T]^T = \mathbf{r}_\lambda(\mathbf{x}_t, \lambda, \mathbf{h}_{t-1}; \mathbf{\Theta}). \tag{9}$$

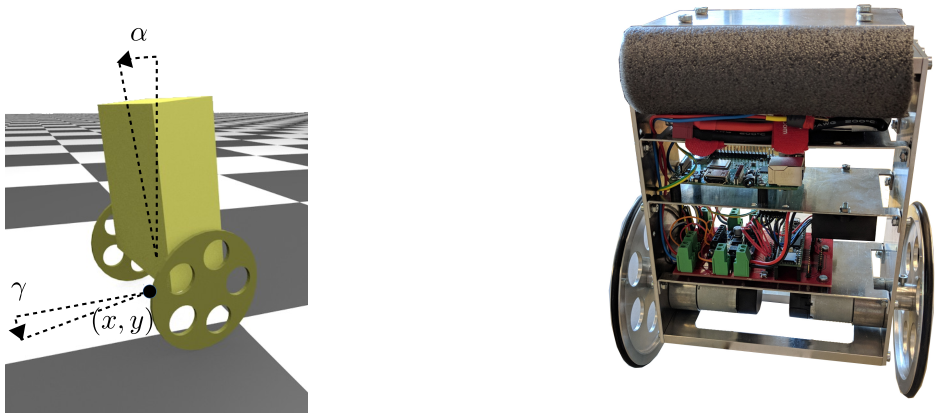
The parameter λ is constant during execution and set by the control engineer. It can be used to adjust the behavior without training a new controller.

4. Task and Model Description

In this section, the mobile inverted pendulum is shortly presented and the control task is clarified. The mathematical model that is used to create the controller is also provided.

4.1. System and Task

The mobile inverted pendulum (MIP) is an unstable system with a non-holonomic contraint, described by a nonlinear model. A schematic and a picture of the hardware are shown in Figure 2. The wheels are accelerated by two DC motors. The sensors include a gyroscope, an accelerometer and encoders attached to the motors. The actuation signal for the motors and the sensor signals are handled by a microcontroller, which also communicates with a Raspberry Pi. The Raspberry Pi is used to evaluate the control law, as memory and computing power of the microcontroller are insufficient for the evaluation of our neural network.



(**a**) Schematic view of the MIP (**b**) The real MIP

Figure 2. The mobile inverted pendulum (MIP).

Because of the challenging system properties of the MIP, it has been extensively studied in the control community. Control laws for the mobile inverted pendulum are often designed around a linearized model and the system is only operated within small tilt angles and target positions in close proximity of the MIP. Controllers based on linearized models can be found, e.g., in [44,45]. An example of a control design based on a nonlinear model is given in Ha et al. [46], however results are only given in simulation.

The task considered in this work is to design a controller for the MIP allowing it to autonomously drive to a desired position, not necessarily in close proximity, and take a desired orientation. The controller furthermore has to stabilize the tilt angle while driving and it has to be computationally cheap enough to be evaluated in real time ($\delta t = 0.01$ s) on the Raspberry Pi.

Let x and y be the position coordinates in an inertial coordinate system I and γ the yaw angle of the MIP. While we want to drive to any position x_r, y_r and orientation γ_r in the inertial coordinate system later, it is sufficient to learn a controller that can drive to a single point and orientation e.g., $[x_r, y_r, \gamma_r] = \mathbf{0}$. A coordinate transform in Equation (10) allows us to place any reference point different from $\mathbf{0}$ in the inertial coordinate system I into the origin of a second coordinate system B, and we can then use x_B, y_B and γ_B for control.

$$[\delta x, \delta y]^T = [x - x_r, y - y_r]^T \tag{10a}$$

$$\begin{bmatrix} x_B \\ y_B \end{bmatrix} = \begin{bmatrix} \cos(\gamma_r) & \sin(\gamma_r) \\ -\sin(\gamma_r) & \cos(\gamma_r) \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} \tag{10b}$$

$$\gamma_B = \gamma - \gamma_r. \tag{10c}$$

4.2. Mathematical Model

The mathematical model of the MIP consists of the body dynamics and the electric drive model. We adopt the model presented in Pathak et al. [47] for our rigid body dynamics. The model has seven states: $\mathbf{x} = [x, y, \gamma, \alpha, \dot{\alpha}, v, \dot{\gamma}]^T$. The state variables consist of the position in the inertial coordinates x, y, the yaw angle γ and its time derivative $\dot{\gamma}$, the tilt angle α with time derivative $\dot{\alpha}$ and the forward velocity v. The equations of the body dynamics model are derived in [47] are repeated in the Appendix A.

The electric drive model consists of a standard model for DC motors found, e.g., in [48], with parameters determined through identification using measurement data. We neglect the current dynamics in our motor model, as they are much faster than the rest of the system's dynamics. We also add a heuristic friction model with four parameters $c_{\text{fric},1\text{-}4}$. We combine the motor torque constant k_2 with the resistance, to reduce the number of parameters to identify, since we are only interested in the input-output relationship and not real physical values. Our friction and drive model are given in Equation (11).

$$\tau_{\text{fric}} = c_{\text{fric},1} \cdot \tanh(c_{\text{fric},2} \cdot \omega) \cdot e^{-c_{\text{fric},3} \cdot \omega^2} + c_{\text{fric},4} \cdot \omega \tag{11a}$$

$$\tau = k_2 \cdot (u - k_1 \cdot \omega) - \tau_{\text{fric}} \tag{11b}$$

The identified and measured parameters for drive, friction and body dynamics are given in Table 1. The parameters c_z, I_{yy} and $c_{\text{fric},1}$ are assumed uncertain for the control design and are depicted as a range, since they are hard to determine with high precision and the model dynamics are sensitive to changes in their value.

Table 1. Model parameters of the mobile inverted pendulum.

Variable	Value	Unit	Description
M_b	1.76	kg	Mass of the body
M_w	0.147	kg	Mass of a wheel
R	0.07	m	Radius of the wheels
c_z	$0.07 \pm 20\%$	m	Height of the center of mass above wheel axis
b	0.09925	m	Half length between wheels
I_{xx}	0.0191	$\text{kg}\,\text{m}^2$	Moment of inertia, x-axis
I_{yy}	$0.0158 \pm 20\%$	$\text{kg}\,\text{m}^2$	Moment of inertia, y-axis
I_{zz}	0.0048	$\text{kg}\,\text{m}^2$	Moment of inertia, z-axis
I_{wa}	$3.6 \cdot 10^{-4}$	$\text{kg}\,\text{m}^2$	Moment of inertia. Wheel, y-axis
I_{wd}	$1.45 \cdot 10^{-3}$	$\text{kg}\,\text{m}^2$	Moment of inertia. Wheel, z-axis
k_1	0.018	V s	Motor constant
k_2	0.61	$\text{N}\,\text{m}\,\text{A}^{-1}$	Motor constant
$c_{\text{fric},1}$	$0.24 \pm 20\%$	$\text{N}\,\text{m}^{-1}$	Friction model constant
$c_{\text{fric},2}$	2.0	/	Friction model constant
$c_{\text{fric},3}$	0.4	/	Friction model constant
$c_{\text{fric},4}$	$8 \cdot 10^{-4}$	$\text{N}\,\text{s}\,\text{m}^{-1}$	Friction model constant

5. Application and Results

In this section, we first provide the settings and details of the approach presented in Section 3 for our application on the MIP. Then, a short analysis of training properties and results on the robustness are given both in simulation and on the real system. Our final recurrent controller of the form in Equation (9) is compared with different static controllers and the oracle controllers in terms of different robustness metrics. The oracle controllers are given only as a reference in the simulation results as a baseline for the performance of the recurrent controllers, however, as mentioned earlier, the oracle controllers cannot be used in the real application.

5.1. Control Design Details

The first part of our approach consists of the trajectory optimization. We optimize N = 10,000 trajectories for each controller. We show, however, that the task can be solved using less trajectories. The large number of trajectories reduces the influence of random sampling for our results. We sample three data sets. The first data-set, which is used to train a reference controller, has varying initial states with constant model parameters and a constant cost function. The second data-set has varying initial states and model parameters, with a constant cost function. The third data-set includes varying initial states, model parameters and the cost function parameter.

The initial states are sampled from uniform distributions. All entries of the initial state vector $\mathbf{x}$ are perturbed, which is important to include starting states of a falling MIP and thus to receive trajectories of the MIP recovering from falling. The initial positions are sampled in a radius of 1.1 m around the origin of the inertial coordinate system. The maximum range we consider for the control later is 1m; all target points outside of this range are projected back onto the radius.

The optimization horizon is 501 steps from $t = 0$ to $t = T = 500$, with a discrete step size of $\delta t = 0.01$ s. Each trajectory is thus 5s long. For the first two data-sets, we minimize the accumulated costs in Equation (12) with the cost functions specified in Equation (13).

$$\begin{aligned}(\mathbf{X}^*, \mathbf{U}^*) &= \underset{\mathbf{X},\mathbf{U}}{\arg\min} \sum_{t=0}^{T-1} c(\mathbf{x}_t, \mathbf{u}_t) \\ &= \underset{\mathbf{X},\mathbf{U}}{\arg\min} \sum_{t=0}^{T-1} c_x(\mathbf{x}_t) + c_u(\mathbf{u}_t)\end{aligned} \tag{12}$$

$$\begin{aligned}c_x(\mathbf{x}_t) =& x_t^2 + y_t^2 + 0.5 \cdot (1 - \cos(\gamma_t)) + \alpha_t^2 \\ &+ 2\left(\alpha_t \cdot \dot{\alpha}_t + 0.05 \cdot \dot{\alpha}_t + 0.1 \cdot v_t^2 + 0.1 \cdot \dot{\gamma}_t^2\right) \end{aligned} \tag{13a}$$

$$c_u(\mathbf{u}_t) = u_{1,t}^2 + u_{2,t}^2 \tag{13b}$$

Our cost function penalizes states that are far from the origin with the term $x_t^2 + y_t^2$ as well as high motor voltages using $c_u(\mathbf{u}_t)$. We also penalize large angular velocities $\dot{\alpha}_t$ and $\dot{\gamma}_t$ and the driving velocity v_t. The cross term $\alpha_t \cdot \dot{\alpha}_t$ penalizes a falling motion, but rewards rising motions. The constant coefficients weight the importance of the different control goals against each other and were hand tuned by trial and error to produce a subjectively appealing behavior of the MIP.

For the described task, however, we were unable to design a cost function describing the desired behavior on its own, which is why we had to add end constraints. The end constraints force our trajectories to end in the origin $\mathbf{x}_T = \mathbf{0}$ at step $T = 500$. Moreover, adding end constraints increased the convergence speed of the optimization in our case.

The necessity of including end constraints for this problem is in fact the main reason we chose the approach of imitation learning with trajectory optimization learning over reinforcement learning approaches. Equality constraints are not possible with reinforcement learning approaches, which are based on probabilistic reasoning. Recent progress in that area has been made to include inequality constraints only [49]. For our problem, the equality and inequality constraints, previously Equations (2d) and (2e), are summarized in Equation (14).

$$[x_T, y_T, 1 - \cos(\gamma_T), \alpha_T, \dot{\alpha}_T, v_T, \dot{\gamma}_T, u_{1,T-1}, u_{1,T-1}] = \mathbf{0}^T \tag{14a}$$

$$-1 \leq u_{1,t} \leq 1 \tag{14b}$$

$$-1 \leq u_{2,t} \leq 1 \tag{14c}$$

$$-0.002 \leq u_{1,t-1} - 2 \cdot u_{1,t} + u_{1,t+1} \leq 0.002 \tag{14d}$$

$$-0.002 \leq u_{2,t-1} - 2 \cdot u_{2,t} + u_{2,t+1} \leq 0.002 \tag{14e}$$

As was mentioned earlier, we use equality constraints in Equation (14a) to force convergence of the trajectories to the desired state in finite time. The inequalities (14b) and (14c) are required since the maximum voltage that can be provided to the motors is restricted. The inequalities (14d) and (14e) are used to prohibit non-continuous control signals that would lead to larger training errors during the approximation with a smooth function approximator. The constraints restrict the second derivative of the control signals, expressed as a finite difference scheme.

After two data-sets of 10,000 trajectories are created, two controllers are trained on this data using supervised learning. The first controller $\mathbf{g}(\mathbf{x})$ is trained on the dataset with constant model parameters and only takes the state as input. The second controller is our oracle controller $\mathbf{g}(\mathbf{x}, \mathbf{p})$ and is trained on the dataset with randomized model parameters as explained in Section 3.2. We use fully connected neural networks with two hidden layers of 128 neurons each. The hidden layers include tanh nonlinearities and a linear output layer. We use a random portion of 80% of the data as our training-set and the remaining data as our test set. We train the neural networks using a GPU over 10,000 epochs. We did not observe overfitting on the data during supervised learning, even when training on as few as 20 trajectories.

The recurrent neural network $\mathbf{r}(\mathbf{x}, \mathbf{h})$ uses three hidden layers: a recurrent tanh layer with 32 neurons, a static tanh layer with 64 neurons and a static tanh layer with 32 neurons. The output layer is again linear. For the data generation in Algorithm 1, we again use the trajectory length $T = 500$ and add a noise with standard deviation $\epsilon = 0.001$. We chose the amplitude of the noise in simulation by aiming for a disturbance that leads to trajectories that are subjectively not too far from the undisturbed case, yet the amplitude should not be so small that the noise is not visible. During each epoch, $N_{traj} = 500$ trajectories are sampled according to Algorithm 1. The training of the recurrent neural network was performed on truncated sequences of 50 time steps, and we performed $N_{gd} = 50$ parameter updates per epoch using Adam [41]. We ran Algorithm 2 for $N_{epoch} = 500$ epochs.

In order to adjust the behavior on the real system later, we use our third data-set to train a second recurrent controller with an additional input $\mathbf{r}_\lambda(\mathbf{x}, \lambda, \mathbf{h})$, as presented in Section 3.4. For this controller, the optimal trajectories are generated using a modified cost function with the adjusting parameter $\lambda \in [-1, 1]$:

$$\begin{aligned} c_x(\mathbf{x}_t, \lambda) =& 10^\lambda \left(x_t^2 + y_t^2 + 0.5 \cdot (1 - \cos(\gamma_t)) + \alpha_t^2 \right) \\ & + 2 \cdot 10^{-\lambda} (\alpha_t \cdot \dot{\alpha}_t + 0.05 \cdot \dot{\alpha}_t \\ & + 0.1 \cdot v_t^2 + 0.1 \cdot \dot{\gamma}_t^2). \end{aligned} \tag{15}$$

This cost function is equal to (13) for $\lambda = 0$. For $\lambda \in]0, 1]$, the velocities are penalized less and the position error is stronger penalized, which leads to faster transition behavior. For $\lambda \in [-1, 0[$, velocities are penalized stronger, leading to a slower transition behavior.

5.2. Results in Simulation

For the following analyses and comparisons, we create a new test-set by optimizing 2000 trajectories that were not included in any training data previously. This set of trajectories contains initial states around 1 m of the origin and random model parameters sampled from the same distribution $D_p(\mathrm{p})$ as for the supervised training.

To assess the robustness of a controller, we evaluate two metrics. The first metric is the mean of the accumulated costs over the initial states and model parameters in the test-set

$$J_{\mathbb{E},c} = \mathbb{E}[\sum_{t=0}^{T} c(x_t, u_t) | x_0 \sim D_{x0}(\mathrm{x}), \mathbf{p} \sim D_p(\mathrm{p})]. \tag{16}$$

The cost function $c(x_t, u_t)$ is the same that was used for the trajectory optimization. A lower value for $J_{\mathbb{E},c}$ means that the controller is closer to the optimal trajectories.

The second metric is the highest accumulated costs subtracted from the optimal accumulated costs with initial states and model parameters from the test-set.

$$J_{\max,c} = \max\left(\sum_{t=0}^{T} c(x_t, u_t) - \sum_{t=0}^{T} c(x_t^*, u_t^*)\right) \tag{17}$$

The metric $J_{\max,c}$ is used to compare the worst case performance of the controllers. Again, smaller values are better.

In order to also quantify the violation of the end constraint, we define the cost function

$$c_T(\mathbf{x}) = x^2 + y^2 + (1 - cos(\phi))^2 + \alpha^2 + \dot{\alpha}^2 + v^2 + \dot{\phi}^2. \tag{18}$$

The cost function $c_T(\mathbf{x})$ is only evaluated using the final state at $t = T$. We use $c_T(\mathbf{x})$ for two metrics that quantify the mean and highest $c_T(\mathbf{x})$ of all simulations with the respective controllers.

$$J_{\mathbb{E},c_T} = \mathbb{E}[c_T(x_T)|x_0 \sim D_{x0}(\mathrm{x}), \mathbf{p} \sim D_p(\mathrm{p})] \tag{19a}$$

$$J_{\max,c_T} = \max c_T(x_T). \tag{19b}$$

As a first analysis, the influence of the number of trajectories and the number of training epochs on the performance on the oracle network is analyzed. Figure 3 shows the expected value of accumulated costs $J_{\mathbb{E},c}$ of oracle controllers, trained by supervised learning on a different number of trajectories N for different numbers of epochs N_{gd}. For a small number of trajectories N, the best control performance is reached after a few epochs of supervised learning and the performance starts degrading with more training. Thus, a decrease in the error during supervised learning does not necessarily lead to an increase in the closed-loop performance. Including more trajectories leads to an increased performance in the closed loop. Including more than 5000 trajectories, however, did not lead to a noticeable increase in the closed loop performance for our case. The only case that produced a controller unable to stabilize the system was with as low as 20 trajectories.

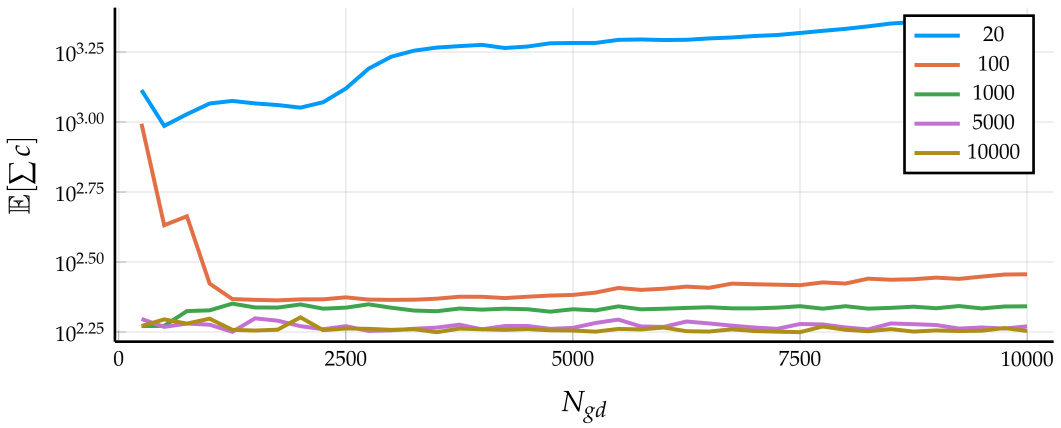


Figure 3. Mean accumulated costs of oracle controllers $\mathbf{g}(\mathbf{x}, \mathbf{p})$ trained on different numbers of trajectories over N_{epoch} epochs. The number of trajectories used during training is given in the line label.

We use the oracle controller that uses all 10000 trajectories to train our recurrent controller $\mathbf{r}(\mathbf{x}, \mathbf{h})$ using DOI. The mean simulation costs of the recurrent controller is depicted in Figure 4, computed every 5 training episodes. The costs are decreasing exponentially with the number of training episodes in our case.

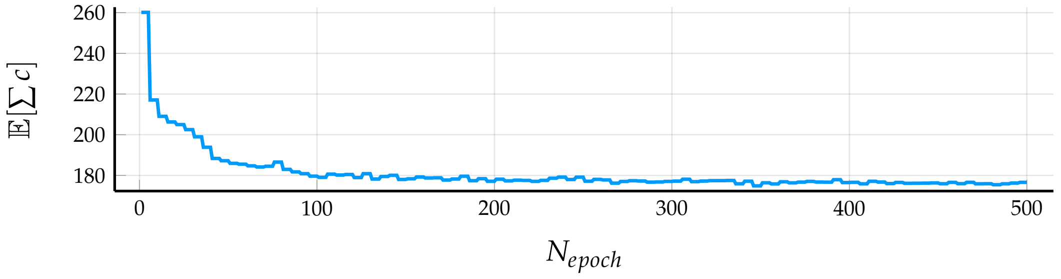


Figure 4. Mean accumulated costs of the recurrent controller $\mathbf{r}(\mathbf{x}, \mathbf{h})$ trained using DOI over the number of epochs N_{epoch}.

For each type of controller, i.e., static, oracle, recurrent and adaptive recurrent controllers, we extract the controllers with the best average closed loop performance that we obtained in the training on 10,000 trajectories for the following comparison. We evaluate the metrics in Equations (16)–(19b) on the test-set of 2000 initial states and model parameters. The values for each controller are shown in Table 2. The oracle network slightly outperforms the simple controller without parameter information in terms of mean costs and maximum above optimal costs. The recurrent controller performs better even than the oracle network in terms of both mean costs and maximal above optimal costs. For the mean accumulated costs this is unexpected, as the recurrent controller did not have a direct access to the cost function during its training. For the maximum above optimal accumulated costs, we believe that the recurrent layer is able to average out poor actions in individual states by acting on the past history instead of only acting on the current state. The adjustable controllers do not perform as well as the non-adjustable controllers in terms of mean costs for the case $\lambda = 0$. Also, for the adjustable case the recurrent controller outperforms the static controller with regards to $J_{\max,c}$.

Table 2. Mean and maximal accumulated costs for different controllers in simulation. Controllers with gray font can not be used in practice and are only given as a reference.

	$\mathbf{g}(\mathbf{x})$	$\mathbf{g}(\mathbf{x}, \mathbf{p})$	$\mathbf{r}(\mathbf{x}, \mathbf{h})$	$\mathbf{g}_\lambda(\mathbf{x}, 0, \mathbf{p})$	$\mathbf{r}_\lambda(\mathbf{x}, 0, \mathbf{h})$	opt.
$J_{\mathbb{E},c}$	181.47	178.33	176.58	185.81	185.75	155.163
$J_{\max,c}$	712.03	520.95	121.38	494.27	182.67	0
$J_{\mathbb{E},c_T}$	0.133	0.102	0.090	0.070	0.051	0
$J_{\max,c_T}$	1.10	2.13	1.34	1.092	0.524	0

5.3. Control Performance in the Application

The controllers, trained in simulation, are transferred unchanged to the real system. To evaluate the control performance, we record measurements of each controller for a test trajectory and evaluate the costs in Equation (13). Our test trajectory is 220 s long and contains 10 random target locations. Every 10 s, the target location changes from the origin to one of the target locations and then back to the origin after another 10 s. The accumulated costs over our test trajectory, i.e., $\sum_t c_x(\mathbf{x}_t)$ and $\sum_t c_u(\mathbf{u}_t)$, for different controllers are given in Table 3 with lower values indicating a better control performance. The recurrent controllers $\mathbf{r}(\mathbf{x}, \mathbf{h})$ and $\mathbf{r}_\lambda(\mathbf{x}, 0, \mathbf{h})$, trained on various model dynamics, achieve a better performance than the static controller with accumulated costs reduced by 20% and 22.7%. The controller $\mathbf{r}_\lambda(\mathbf{x}, 0.3, \mathbf{h})$ is also represented in the table, and performs slightly worse than $\mathbf{r}_\lambda(\mathbf{x}, 0, \mathbf{h})$. This is expected as the cost functions (13a) that is used for the evaluation of the values in Table 3 is different from the costs in Equation (15) for $\lambda \neq 0$.

Measurement data for the test trajectory for the static controller $\mathbf{g}(\mathbf{x})$, the recurrent controller $\mathbf{r}(\mathbf{x},\mathbf{h})$ and the adjusted controller $\mathbf{r}_\lambda(\mathbf{x}, 0.3, \mathbf{h})$ are shown in Figures 5–7, respectively. In the application, the approaching of the target position is slower than in simulation due to oscillations in the tilt angle at low velocities. The remaining position error is most pronounced for the static controller in Figure 5, e.g., when approaching the new target position after 30 s. The position error is visibly reduced using a recurrent controller $\mathbf{r}(\mathbf{x},\mathbf{h})$ as can be seen comparing Figure 6 with Figure 5. The performance increase is also reflected in a reduced value for $\sum_t c_x$ in Table 3. However, even after 10 s the target location is not reliably reached for the static and the recurrent controller, as the controllers try to reduce the angular velocity of the tilt angle rather than drive towards the target position. We therefore use the adjustable controller $\mathbf{r}_\lambda(\mathbf{x}, \lambda, \mathbf{h})$ and increase the adjusting parameter λ to give more weight to the position and less weight to velocities. As can be seen in Table 3, this leads to an increase in the accumulated costs compared to the controller $\mathbf{r}(\mathbf{x},\mathbf{h})$ according to the cost function in Equation (13), caused by higher velocities and control signals. However, the behavior is subjectively better due to faster and more accurate approaching of the target location as is seen in Figure 7. Increasing λ also increases the speed of convergence and accuracy of the yaw angle, as can be seen by comparing Figure 7 with Figures 5 and 6 respectively.

Table 3. Accumulated costs for different controllers on a test trajectory in the real application .

	$\mathbf{g}(\mathbf{x})$	$\mathbf{r}(\mathbf{x},\mathbf{h})$	$\mathbf{r}_\lambda(\mathbf{x},0,\mathbf{h})$	$\mathbf{r}_\lambda(\mathbf{x},0.3,\mathbf{h})$
$\sum c_x$	5221.86	3957.26	4062.04	4131.76
$\sum c_u$	1674.85	1566.88	1265.35	1431.55
$\sum c$	6896.71	5524.14	5327.39	5563.31

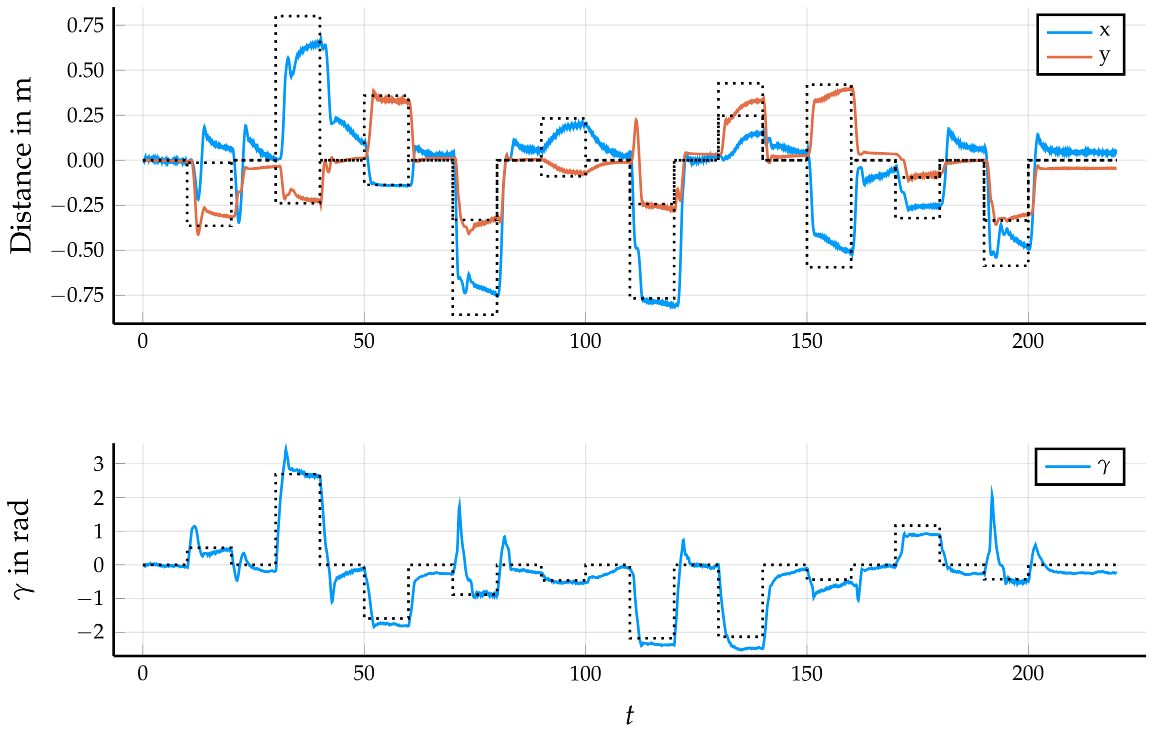


Figure 5. Measurement data for an application of a static neural network controller $\mathbf{g}(\mathbf{x})$. Units are in meters for the position coordinates x and y (top plot) and radian for γ (bottom plot).

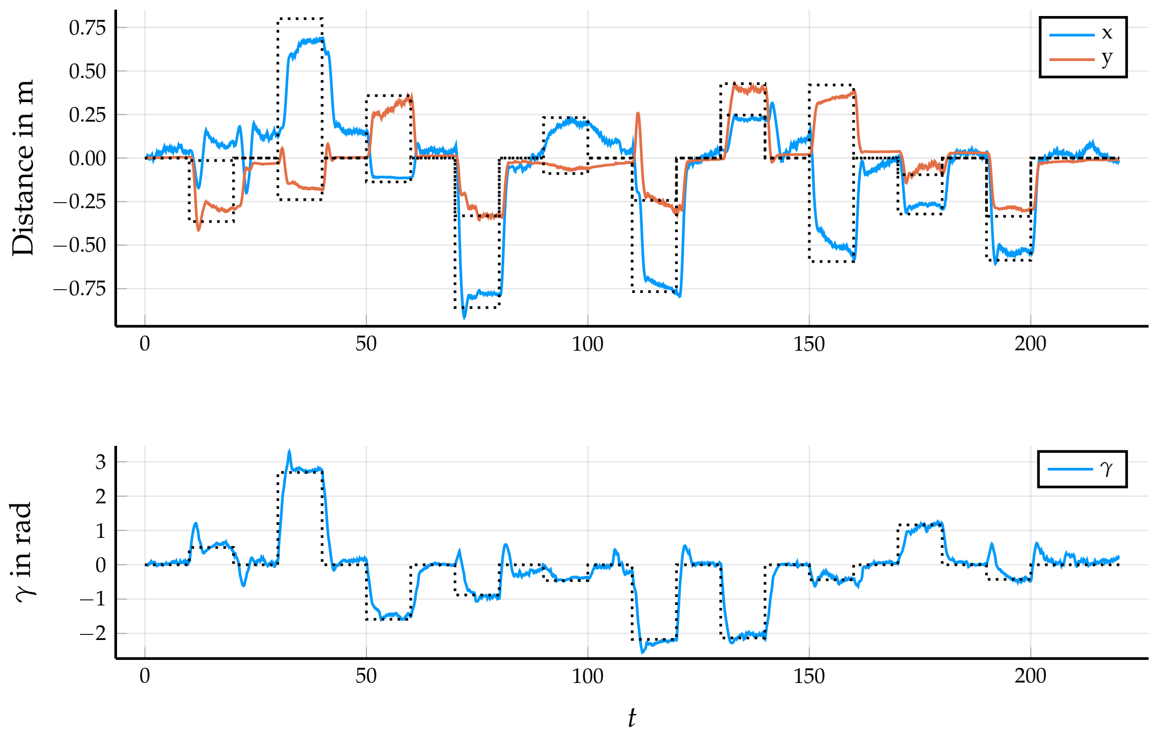


Figure 6. Measurement data for an application of a recurrent neural network controller $\mathbf{r}(\mathbf{x}, \mathbf{h})$. Units are in meters for the position coordinates x and y (top plot) and radian for γ (bottom plot).

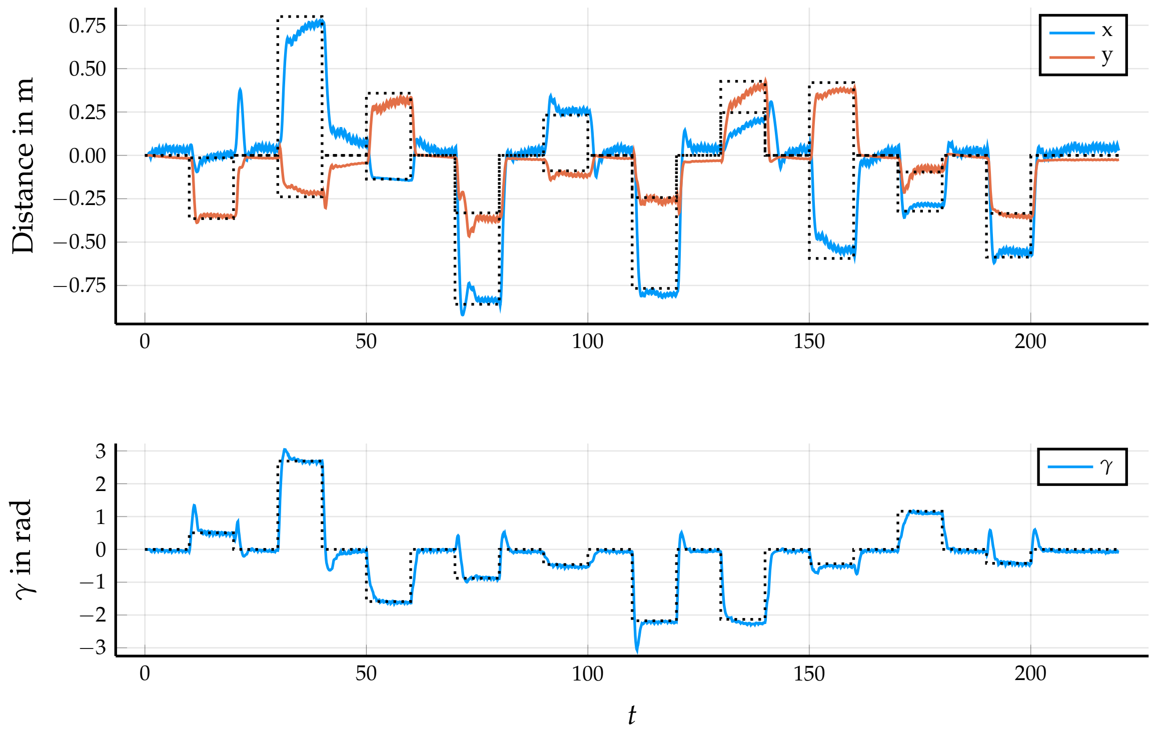


Figure 7. Measurement data for an application of an adjustable recurrent neural network controller $\mathbf{r}_\lambda(\mathbf{x}, \lambda, \mathbf{h})$ with $\lambda = 0.3$. Units are in meters for the position coordinates x and y (top plot) and radian for γ (bottom plot).

For a qualitative impression of the control performance of the final controller $\mathbf{r}_\lambda(\mathbf{x}, \lambda, \mathbf{h})$ with $\lambda = 0.3$, a sequence of images of the MIP during the test trajectory is shown in Figure 8 for the real application and in Figure 9 for the same controller in the simulation. For the real application, a visualisation of the recorded measurement data is provided for each timestep as well, with the target position depicted as a static green MIP. The time delay between each image is 0.25 s. Comparing both, we see that the dynamic manoeuvre is performed almost identically with the largest differences close to the position of rest. For a further qualitative impression, a video of the MIP using the adjustable controller is accessible at https://youtu.be/MwVZgRJSnXg.

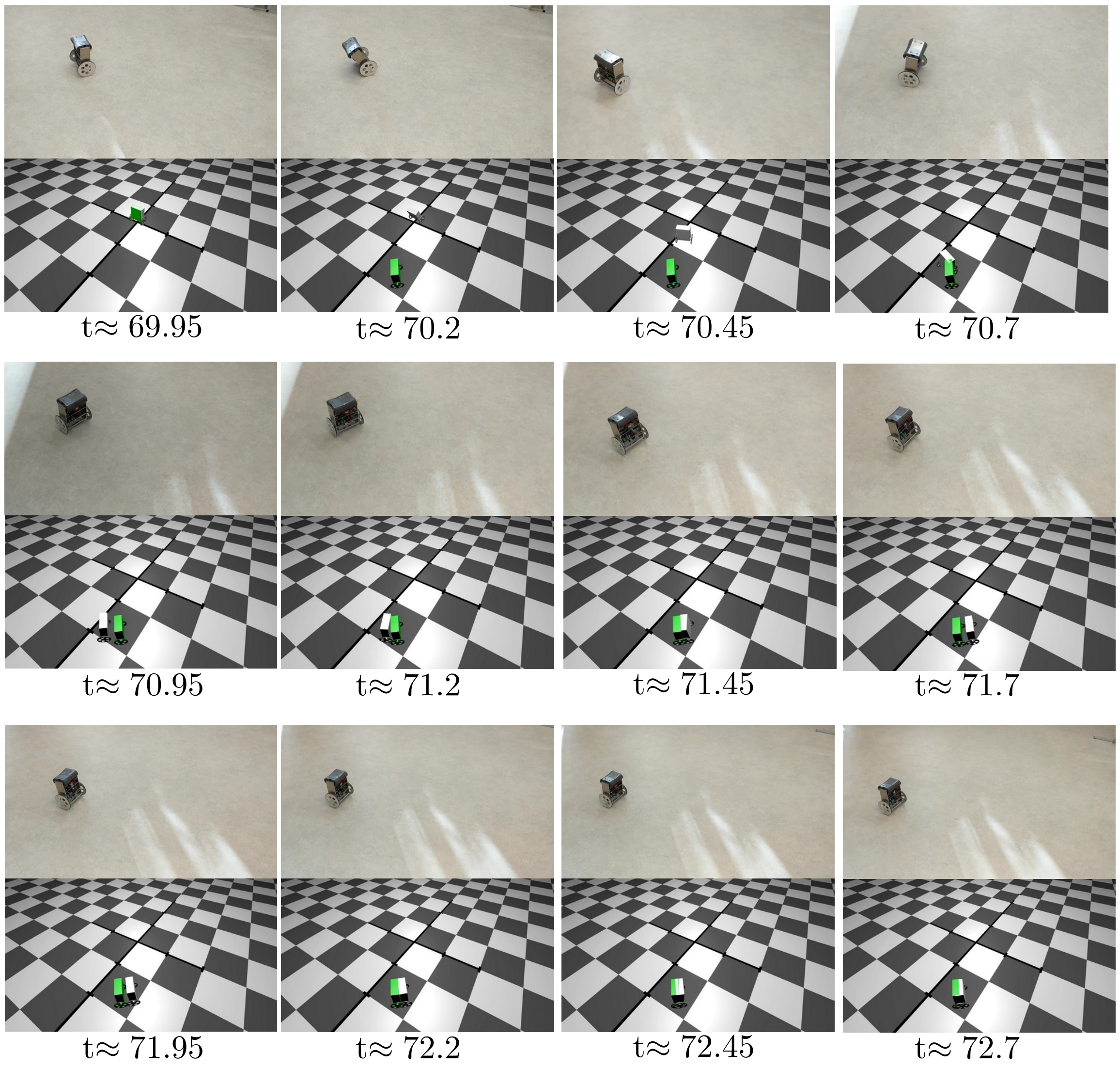


Figure 8. Image sequence showing a manoeuvre of the real MIP using the recurrent control structure $\mathbf{r}_\lambda(\mathbf{x}, \lambda, \mathbf{h})$ with $\lambda = 0.3$. The top image shows the real system and attached below is a visualization of the measurement data (gray MIP), also showing the target position as a green MIP.

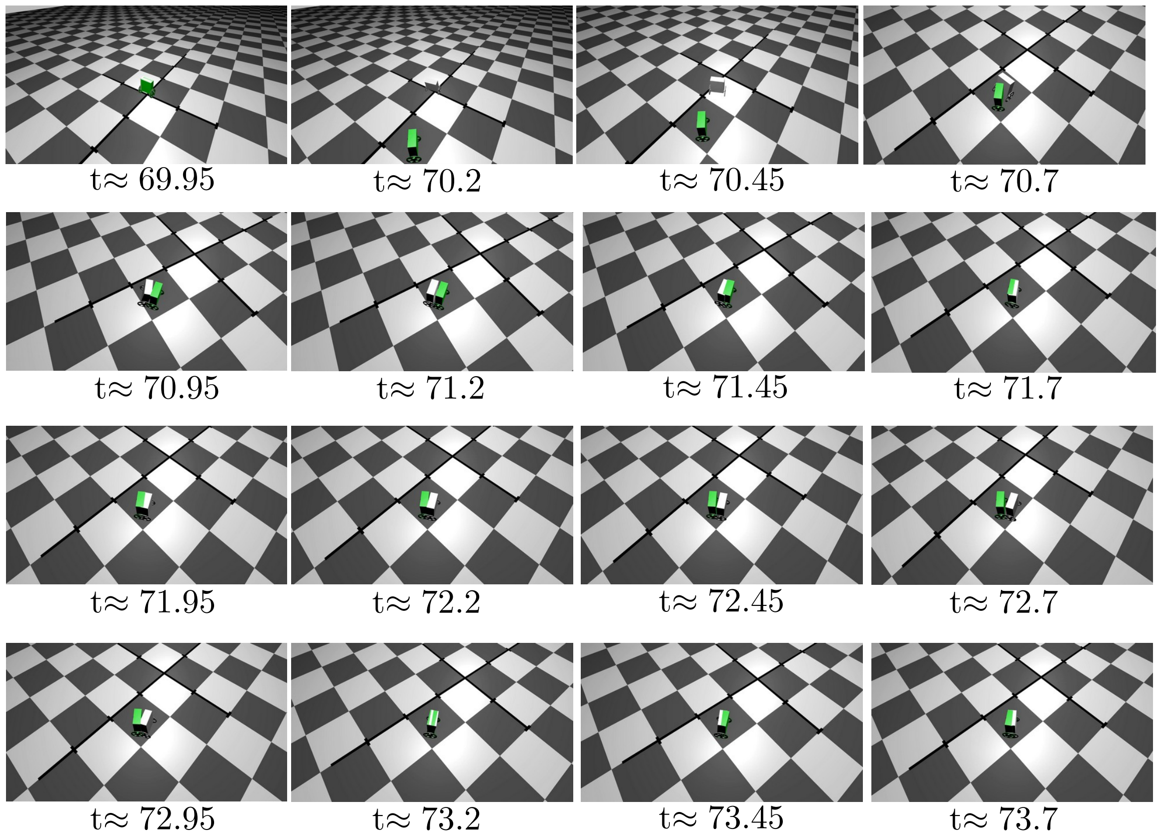


Figure 9. Image sequence showing a manoeuvre of the MIP in simulation using the recurrent control structure $\mathbf{r}_\lambda(\mathbf{x}, \lambda, \mathbf{h})$ with $\lambda = 0.3$. The target position is shown as a green MIP.

5.4. Outlook of Application Specific Variations

Our controller was trained to drive the MIP towards static target positions and orientations. By continuously shifting the target position, a trajectory or a moving target can be followed by this type of controller as well. An application suited for this type of controller is, e.g., driving after a person based on an on-board camera. However, the MIP will be driving slightly behind the target position in the case of a moving target, as it was trained to reach targets with zero velocity. This effect can be seen in the video at https://youtu.be/MwVZgRJSnXg. If a more precise trajectory tracking is required, one has to include training data of trajectories that were optimized to follow non stationary targets. Other tasks, e.g., ascending slopes or avoiding collisions, also require the addition of optimized trajectories specific for that task.

6. Conclusions

We developed a parametric feedback controller for the mobile inverted pendulum using imitation learning with optimized trajectories. The controller is able to stabilize the system and drive to target positions within a certain radius without the need to compute trajectories online. The optimal trajectories used in the training were generated using varying model parameters in simulation and the controller used a recurrent structure in order to adapt its behavior to the real system. In order to train a recurrent controller using imitation learning, it is necessary to have training targets on state sequences generated by the recurrent controller itself. We therefore trained an intermediate oracle controller with full information of the model parameters that acts as a teacher to the recurrent controller. We show an improvement of the robustness of the controller both in simulation and in the real application by comparing it to a controller without the recurrent structure. Finally, an additional input allows us to easily adjust the behavior of the recurrent controller in the application.

Author Contributions: Conceptualization, methodology, software, validation, formal analysis, investigation, data curation, writing—original draft preparation, visualization: C.D.; resources, writing—review and editing, supervision, project administration, funding acquisition: B.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by the German Research Foundation DFG, SFB768.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DAGGER	Dataset aggregation
DART	Disturbances for augmenting robot trajectories
DOI	Disturbed oracle imitation
MIP	Mobile inverted pendulum
MPC	Model Predictive Control
TBPTT	Truncated backpropagation through time

Appendix A. Rigid Body Dynamics Model

The rigid body dynamics model of a MIP that is derived in Pathak et al. [47] is repeated in the following. The input is a vector of torques applied to the right and left wheels $\boldsymbol{\tau} = [\tau_r, \tau_l]$. The model parameters with description are depicted in Table 1. The state space model is

$$\dot{x} = \cos(\alpha) \cdot v \tag{A1a}$$
$$\dot{y} = \sin(\alpha) \cdot v \tag{A1b}$$
$$\dot{\gamma} = \dot{\gamma} \tag{A1c}$$
$$\dot{\alpha} = \dot{\alpha} \tag{A1d}$$
$$\ddot{\alpha} = f_5(\mathbf{x}) + g_5(\mathbf{x})(\tau_r + \tau_l) \tag{A1e}$$
$$\dot{v} = f_6(\mathbf{x}) + g_6(\mathbf{x})(\tau_r + \tau_l) \tag{A1f}$$
$$\ddot{\gamma} = f_7(\mathbf{x}) + g_7(\mathbf{x})(\tau_r - \tau_l) \tag{A1g}$$

with the abbreviations

$$D(\alpha) = M_b^2 \cos^2(\alpha) c_z R^2 + \left((-M_b^2 - 2M_w M_b) c_z^2 - 2I_{yy} M_w - IyyM_b \right) R^2 - 2M_b c_z^2 I_{wa} - 2I_{yy} I_{wa} \tag{A2a}$$
$$G(\alpha) = (-M_b c_z^2 + I_{zz} - I_{xx}) R^2 \cos^2(\alpha) + (M_b c_z^2 + I_{xx} + 2I_{wd} + 2b^2 M_w) R^2 + 2b^2 I_{wa} \tag{A2b}$$
$$K(\alpha) = M_b R^2 c_z \sin(\alpha) \left(-4I_{yy} - 3M_b c_z^2 + I_{xx} - I_{zz} \right) + M_b R^2 c_z \sin(3\alpha) \left(I_{xx} - I_{zz} + M_b c_z^2 \right) \tag{A2c}$$
$$H = \left(\frac{1}{2} M_b R^2 + M_w R^2 + I_{wa} \right) (I_{zz} - I_{xx}) - M_b c_z^2 (M_w R^2 + I_{wa}) \tag{A2d}$$

$$f_5(\mathbf{x}) = \frac{\sin(2\alpha)\dot{\gamma}^2 H - M_b c_z g \sin(\alpha)\left(M_b R^2 + 2(I_{wa} + M_w R^2)\right)}{D(\alpha)} + \frac{M_b^2 c_z^2 R^2 \sin(2\alpha)\dot{\alpha}^2}{2D(\alpha)} \tag{A3a}$$

$$f_6(\mathbf{x}) = K(\alpha)\dot{\gamma}^2 + \frac{M_b^2 c_z^2 R^2 g \sin(2\alpha) - 2\sin(\alpha)\dot{\alpha}^2 M_b R^2 c_z\left(I_{yy} + M_b c_z^2\right)}{2D(\alpha)} \tag{A3b}$$

$$f_7(\mathbf{x}) = \frac{-R^2\dot{\gamma}}{G(\alpha)}\left(\sin(2\alpha)\dot{\alpha}(I_{xx} - I_{zz} + M_b c_z^2) + \sin(\alpha) M_b c_z v\right) \tag{A3c}$$

$$g_5(\mathbf{x}) = \frac{M_b R^2 + 2M_w R^2 + 2I_{wa} + M_b \cos(\alpha) c_z R}{D(\alpha)} \tag{A4a}$$

$$g_6(\mathbf{x}) = -\frac{R\left(M_b \cos(\alpha) c_z R + I_{yy} + M_b c_z^2\right)}{D(\alpha)} \tag{A4b}$$

$$g_7(\mathbf{x}) = \frac{Rb}{G(\alpha)}. \tag{A4c}$$

References

1. Yang, C.; Li, Z.; Li, J. Trajectory planning and optimized adaptive control for a class of wheeled inverted pendulum vehicle models. *IEEE Trans. Cybern.* **2012**, *43*, 24–36. [CrossRef] [PubMed]
2. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
3. Faulwasser, T.; Weber, T.; Zometa, P.; Findeisen, R. Implementation of nonlinear model predictive path-following control for an industrial robot. *IEEE Trans. Control Syst. Technol.* **2016**, *25*, 1505–1511. [CrossRef]
4. Deisenroth, M.; Rasmussen, C.E. PILCO: A model-based and data-efficient approach to policy search. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), Bellevue, WA, USA, 28 June–2 July 2011; pp. 465–472.
5. Golemo, F.; Taiga, A.A.; Courville, A.; Oudeyer, P.Y. Sim-to-Real Transfer with Neural-Augmented Robot Simulation. In Proceedings of the Conference on Robot Learning, Zurich, Switzerland, 29–31 October 2018; pp. 817–828.
6. Pattanaik, A.; Tang, Z.; Liu, S.; Bommannan, G.; Chowdhary, G. Robust deep reinforcement learning with adversarial attacks. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, Stockholm, Sweden, 10–15 July 2018; pp. 2040–2042.
7. Pinto, L.; Davidson, J.; Sukthankar, R.; Gupta, A. Robust adversarial reinforcement learning. *arXiv* **2017**, arXiv:1703.02702.
8. Rajeswaran, A.; Ghotra, S.; Ravindran, B.; Levine, S. Epopt: Learning robust neural network policies using model ensembles. *arXiv* **2016**, arXiv:1610.01283.
9. Peng, X.B.; Andrychowicz, M.; Zaremba, W.; Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 1–8.
10. Muratore, F.; Treede, F.; Gienger, M.; Peters, J. Domain randomization for simulation-based policy optimization with transferability assessment. In Proceedings of the Conference on Robot Learning, Zurich, Switzerland, 29–31 October 2018; pp. 700–713.
11. Hussein, A.; Gaber, M.M.; Elyan, E.; Jayne, C. Imitation learning: A survey of learning methods. *ACM Comput. Surv. (CSUR)* **2017**, *50*, 21. [CrossRef]

12. Dessort, R.; Chucholowski, C. Explicit model predictive control of semi-active suspension systems using Artificial Neural Networks (ANN). In *8th International Munich Chassis Symposium 2017*; Pfeffer, P.E., Ed.; Springer Fachmedien Wiesbaden: Wiesbaden, Germany, 2017; pp. 207–228.
13. Mordatch, I.; Todorov, E. Combining the benefits of function approximation and trajectory optimization. *Robot. Sci. Syst.* **2014**, *4*, 5–32.
14. Abbeel, P.; Quigley, M.; Ng, A.Y. Using inaccurate models in reinforcement learning. In Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, USA, 25–29 June 2006; pp. 1–8.
15. Mordatch, I.; Lowrey, K.; Todorov, E. Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids. In Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 28 September–2 October 2015; pp. 5307–5314.
16. Liu, C.; Li, H.; Gao, J.; Xu, D. Robust self-triggered min–max model predictive control for discrete-time nonlinear systems. *Automatica* **2018**, *89*, 333–339. [CrossRef]
17. Yu, W.; Liu, C.K.; Turk, G. Preparing for the Unknown: Learning a Universal Policy with Online System Identification. *arXiv* **2017**, arXiv:1702.02453,
18. Schulman, J.; Levine, S.; Moritz, P.; Jordan, M.I.; Abbeel, P. Trust Region Policy Optimization. *arXiv* **2015**, arXiv:1502.05477,
19. Muratore, F.; Gienger, M.; Peters, J. Assessing Transferability from Simulation to Reality for Reinforcement Learning. *IEEE Trans. Pattern Anal. Mach. Intell.* **2019**. [CrossRef]
20. Bousmalis, K.; Irpan, A.; Wohlhart, P.; Bai, Y.; Kelcey, M.; Kalakrishnan, M.; Downs, L.; Ibarz, J.; Pastor, P.; Konolige, K.; et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 4243–4250.
21. Chebotar, Y.; Handa, A.; Makoviychuk, V.; Macklin, M.; Issac, J.; Ratliff, N.; Fox, D. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 8973–8979.
22. Salimans, T.; Ho, J.; Chen, X.; Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv* **2017**, arXiv:1703.03864.
23. Chrabaszcz, P.; Loshchilov, I.; Hutter, F. Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari. *arXiv* **2018**, arXiv:1802.08842
24. Rückstieß, T.; Sehnke, F.; Schaul, T.; Wierstra, D.; Sun, Y.; Schmidhuber, J. Exploring parameter space in reinforcement learning. *Paladyn* **2010**, *1*, 14–24. [CrossRef]
25. Ortega, J.G.; Camacho, E. Mobile robot navigation in a partially structured static environment, using neural predictive control. *Control Eng. Pract.* **1996**, *4*, 1669–1679. [CrossRef]
26. Åkesson, B.M.; Toivonen, H.T.; Waller, J.B.; Nyström, R.H. Neural network approximation of a nonlinear model predictive controller applied to a pH neutralization process. *Comput. Chem. Eng.* **2005**, *29*, 323–335. [CrossRef]
27. Ross, S.; Gordon, G.; Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Ft. Lauderdale, FL, USA, 11–13 April 2011; pp. 627–635.
28. He, H.; Eisner, J.; Daume, H. Imitation learning by coaching. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2012; pp. 3149–3157.
29. Laskey, M.; Lee, J.; Fox, R.; Dragan, A.; Goldberg, K. Dart: Noise injection for robust imitation learning. *arXiv* **2017**, arXiv:1703.09327.
30. Mordatch, I.; Lowrey, K.; Andrew, G.; Popovic, Z.; Todorov, E.V. Interactive control of diverse complex characters with neural networks. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2015; pp. 3132–3140.
31. Levine, S.; Koltun, V. Guided policy search. In Proceedings of the 30th International Conference on Machine Learning (ICML-13), Atlanta, GA, USA, 16–21 June 2013; pp. 1–9.
32. Levine, S.; Koltun, V. Learning complex neural network policies with trajectory optimization. In Proceedings of the International Conference on Machine Learning, Bejing, China, 22–24 June 2014; pp. 829–837.

33. Zhang, T.; Kahn, G.; Levine, S.; Abbeel, P. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In Proceedings of the 2016 IEEE international conference on robotics and automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 528–535.
34. Kahn, G.; Zhang, T.; Levine, S.; Abbeel, P. PLATO: Policy Learning using Adaptive Trajectory Optimization. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017.
35. Levine, S.; Finn, C.; Darrell, T.; Abbeel, P. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.* **2016**, *17*, 1334–1373.
36. Paul, S.; Kurin, V.; Whiteson, S. Fast Efficient Hyperparameter Tuning for Policy Gradient Methods. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 4618–4628.
37. Von Stryk, O.; Bulirsch, R. Direct and indirect methods for trajectory optimization. *Ann. Oper. Res.* **1992**, *37*, 357–373. [CrossRef]
38. Bock, H.G.; Plitt, K.J. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proc. Vol.* **1984**, *17*, 1603–1608. [CrossRef]
39. Biegler, L.T. *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*; SIAM: Philadelphia, PA, USA, 2010; Volume 10.
40. Wächter, A. An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2002.
41. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980,
42. Tallec, C.; Ollivier, Y. Unbiasing Truncated Backpropagation Through Time. *arXiv* **2017**, arXiv:1705.08209.
43. Ang, K.H.; Chong, G.; Li, Y. PID control system analysis, design, and technology. *IEEE Trans. Control Syst. Technol.* **2005**, *13*, 559–576.
44. Muralidharan, V.; Mahindrakar, A.D. Position Stabilization and Waypoint Tracking Control of Mobile Inverted Pendulum Robot. *IEEE Trans. Control Syst. Technol.* **2014**, *22*, 2360–2367. [CrossRef]
45. Dini, N.; Majd, V.J. Model predictive control of a wheeled inverted pendulum robot. In Proceedings of the 2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM), Tehran, Iran, 7–9 October 2015; pp. 152–157. [CrossRef]
46. Ha, J.; Lee, J. Position control of mobile two wheeled inverted pendulum robot by sliding mode control. In Proceedings of the 2012 12th International Conference on Control, Automation and Systems, JeJu Island, Korea, 17–21 October 2012; pp. 715–719.
47. Pathak, K.; Franch, J.; Agrawal, S.K. Velocity and position control of a wheeled inverted pendulum by partial feedback linearization. *IEEE Trans. Robot.* **2005**, *21*, 505–513. [CrossRef]
48. Kara, T.; Eker, I. Nonlinear modeling and identification of a DC motor for bidirectional operation with real time experiments. *Energy Convers. Manag.* **2004**, *45*, 1087–1106. [CrossRef]
49. Achiam, J.; Held, D.; Tamar, A.; Abbeel, P. Constrained Policy Optimization. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017.